The work described in this report was supported by the National Science Foundation under Grant number CMS-0117853. Any opinions, findings, or conclusions are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Cyberenvironment Project Management:

Lessons Learned

September 5, 2006

B. F. Spencer, Jr.[1]

Randal Butler[2]

Kathleen Ricker[2]

Doru Marcusiu[2]

Thomas Finholt[3]

Ian Foster[4]

Carl Kesselman[5]

[1] Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, Urbana, IL
[2] National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana, IL
[3] School of Information, University of Michigan, Ann Arbor, MI
[4] Argonne National Laboratory, Argonne, IL
[5] Information Sciences Institute, University of Southern California, Marina Del Way, CA

Suggested citation: Spencer, B., Butler, R., Ricker, K., Marcusiu, D., Finholt, T., Foster, I., Kesselman, C. "Cyberenvironment Project Management: Lessons Learned." (September 2006). http://neesgrid.ncsa.uiuc.edu/documents/CPMLL.pdf

Acknowledgments

This work received support from the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) Program of the National Science Foundation under Award Number CMS-0117853; from the National Science Foundation Middleware Initiative; and from the National Science Foundation's NCSA CORE award.

Many former NEESgrid collaborators and current colleagues provided input and feedback crucial to the writing of this document. We could not have proceeded without their assistance and support.

This document could not have been conceived at all, had it not been for the years of close collaboration of the NEES System Integrator Team that went into making the NEESgrid a reality. We gratefully acknowledge the extensive and, in many cases, continuing contributions of the following people to NEESgrid: Daniel Abrams, Kazi Anwar, Sung Joo Bae, Jean-Pierre Bardet, Cristina Beldica, Joe Bester, Jeremy Birnholtz, Michelle Butler, Randal Butler, Chris Cribbs, Mike D'Arcy, Shirley Dyke, Jim Eng, Gregory Fenves, Filip Filippou, Thomas Finholt, Ian Foster, Joseph Futrelle, Jeff Gaynor, David Gehrig, William Glick, Glenn Golden, Scott Gose, Gullapalli, Joseph Hardin, Tomasz Haupt, Erik Hofer, Dan Horn, Paul Hubbard, Erik Johnson, Anand Kalyanasundaram, Carl Kesselman, Sung Jig Kim, Young Suk Kim, Samuel Lang, Robert Lau, Kincho Law, John Leasia, Lee Liming, Doru Marcusiu, Francis McKenna, Dheeraj Motwani, Nancy Moussa, Jim Myers, Narutoshi Nakata, Laura Pearlman, Gojkhan Pekcan, Jun Peng, Chase Phillips, Joel Plutchak, Kathleen Ricker, Lars Schumann, Charles Severance, B. F. Spencer, Jennifer Swift, Suzandeise Thome, Von Welch, Terry Weymouth, Guangquiang Yang, and Nestor Zaluzec.

For helping give us a clearer picture of where NEESgrid fits into the bigger landscape of research community cyberenvironments, as well as for their patient willingness to provide us with the occasional "sanity check," we would like to thank Danny Powell and Jim Myers of NCSA. For taking the time to read and offer helpful comments and suggestions, we'd like to thank Jerry Hajjar of the Department of Earthquake and Civil Engineering at UIUC and Charles Severance of the School of Information at the University of Michigan.

And finally, for providing us with her unique and invaluable point of view as a funding agency representative, as well as her personal support for the writing and eventual dissemination of this document, we would like to thank Joy Pauschke of the National Science Foundation.

Overview

This paper describes important lessons we have learned through our experiences in community cyberenvironment development, and specifically, through our experience developing one of the first large-scale community cyberenvironments. That network was the NEESgrid, which connected earthquake engineering researchers throughout the United States and the world with each other and with experimental apparatus, enabling them to breach disciplinary and geographical barriers to deliver innovative solutions to seismic safety problems.

Like the contents of so many books and articles about project management, most of the lessons set forth here may seem, at first glance, to be common sense principles, and we do not wish to imply that our conclusions are somehow entirely new. But cyberenvironments *are* breaking new ground. What we learned with NEESgrid and subsequent similar projects is born of practical experience. We hope that, as more research communities see how they can benefit from cyberenvironments, they will also benefit from our experience with these large-scale, community-driven projects.

As an early cyberenvironment, NEESgrid was intended to address the goals of the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) initiatives. Specifically, these goals included 1) transforming the nation's ability to carry out earthquake engineering research, 2) obtaining information vital for developing improved methods for reducing the nation's vulnerability to catastrophic earthquakes, and 3) educating new generations of engineers, scientists and other specialists committed to improving seismic safety. Fifteen earthquake engineering equipment sites with advanced testing capabilities were established under NEES to achieve these goals.

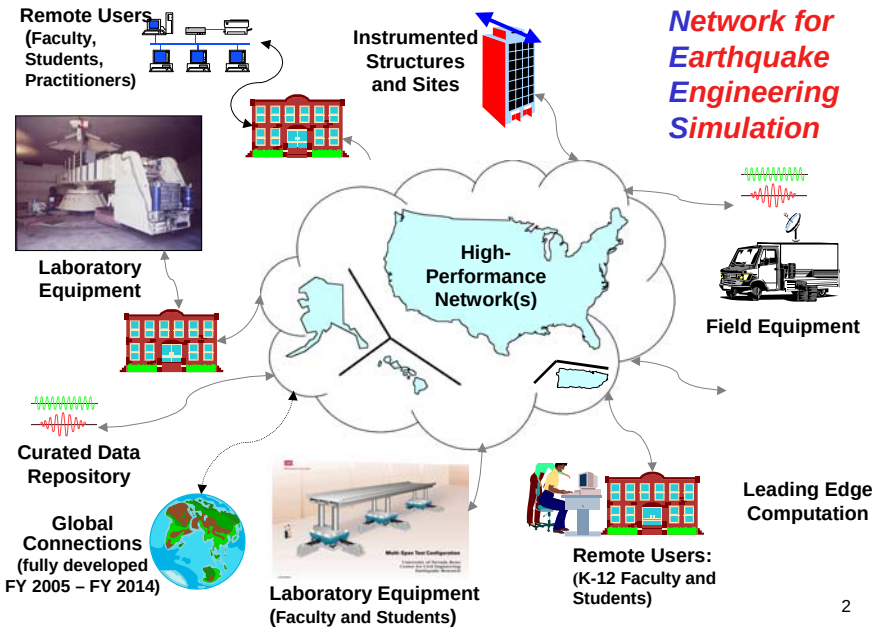

Figure 1: The NEESgrid concept.

To insure that the nation's researchers could effectively use this equipment, equipment sites were to be operated as shared-use facilities, and NEES was implemented as a network-enabled collaboratory. The overall goal was to enable members of the earthquake engineering community to interact with one another, access unique, next-generation instruments and equipment, share data and computational resources, and retrieve information from digital libraries without regard to geographical location. The portfolio of equipment included new or upgraded shaking tables, reaction wall facilities, geotechnical centrifuges, tsunami wave tanks, and mobile and permanently installed field equipment. At each site, participation by off site collaborators was to be encouraged through advanced teleobservation and teleoperation. Invitations were issued to other national and international research facilities to join NEES.

Two characteristics that made the NEES system integration (SI) team unusual were its size and makeup. While NCSA and the Department of Civil and Environmental Engineering at UIUC (CEE-UIUC) provided overall project management and administration, the SI team was very much a geographically distributed collaboration, with key project teams located on the West Coast and throughout the Midwest and the South. However, the heart and soul of NEESgrid was the cooperative relationship between more than sixty applications developers, Grid and cyberinfrastructure researchers, social networking experts, and earthquake engineers at Argonne National Laboratory, Mississippi State University, Pacific Northwest National Laboratory, Stanford University, the University of California at Berkeley, UIUC and NCSA, the University of Michigan, the University of Nevada at Reno, the University of Southern California and the Information Science Institute at USC, and Washington University in St. Louis.

NEESgrid brought together a group of prominent, accomplished technologists with experience as principal investigators and team members on research projects of their own. The team included researchers who had developed significant, cutting-edge cyberinfrastructure components, such as Globus and CHEF, as well as similarly novel applications such as eNotebook and OpenSees. However, for most of the team, developing a community-driven cyberenvironment was, in many ways, uncharted territory. NEESgrid was not a conventional single-PI research project. The technologists were being asked to create something that did not exist, to do so on a massive scale, to leverage the expertise of diverse, geographically-distributed individual research teams, gather requirements, analyze them, develop and deploy a system in just three years and on a modest budget, and—perhaps most challenging of all—to do so in close collaboration with a community (the earthquake engineers) that was struggling to understand how it all fit into their world. For this community, the idea of cyberenvironments was entirely new and presented an entirely different, and even more daunting, set of challenges.

In this paper, we have tried to distill what we learned in confronting these challenges. We discuss what worked, what did not, and, in some cases, how we applied these lessons in later projects. Community cyberenvironment development is a rapidly changing field,

and each project and relationship will be unique. We hope that others who find themselves facing similar challenges will find our experience—sometimes painful, but always valuable—to be of use in navigating these unfamiliar waters.

1 Users and technologists need each other to succeed.

The most important principle of community cyberenvironment development—and one which we stress repeatedly throughout this document—is that **the development of a community cyberenvironment needs to be a full partnership between the user community and the cyberinfrastructure technologists.** Because the point of the project is to advance the user community's ability to conduct research and education, the technologist community must respond to that community's needs. If the users are to adopt the software and use it effectively, they must have a primary role in determining its capabilities and functions.

Careful technologists will take the time needed to understand fully how users currently work, and why, rather than simply assuming that the innovations they propose are an inevitable improvement. It can be tempting to skip past this exercise to focus directly on how things will be in the future, but this step is necessary to help keep the technologists well-grounded and to aid in helping the two groups better understand and communicate with each other. Users understand what they need and, moreover, *why* they do things the way they do, which is not always apparent to others outside the community. Technologists need to understand this point, as well to understand that most researchers are not technologically naïve. In other words, cyberenvironment development is a two-way street—users need to be able to describe to technologists how they work, and technologists need to be able to explain to users how a community cyberenvironment can enable them to do even more.

Integrating notions of user-centered, user-driven design into cyberinfrastructure development can be problematic, however, because cyberinfrastructure and cyberenvironment development often advances so rapidly that it is far removed from the everyday experience of most domain researchers. It can be unrealistic to expect potential users to provide constructive feedback on a technology that doesn't yet exist and that doesn't even have precursors that provide users with even a vague idea of what they could be getting, but that, if executed successfully, could have unimaginable and unpredictable transformative effects on their research and, possibly, their field as a whole.

Thus, it's important to keep in mind that the developers are much more intimate with both the limits and the potential of the technology itself and are therefore in a better position to be able to tell the users what they will and will not be able to do with a given component, feature, or release. Technologists will also be able to make end users aware of already-existing tools, resources, and services that can be leveraged for domain-specific applications. Furthermore, it is the technologists' responsibility to design and build systems that are implementable, extensible, and supportable. Technologists must use their expertise to build systems that are flexible and that will do more than just meet today's needs.

Partnership not only need to begin in the early planning stage, but should be nurtured throughout the process. Over time, both groups should benefit from the relationship—the

technologists will understand the user's needs better, and the users will become much more knowledgeable about the technology. The following chapters describe in detail how the need for partnership informs every aspect of the cyberinfrastructure project as a whole, as well as every stage of development.

2 You must have a target and know how to reach it.

Take planning seriously. "It's not the plan that's important, it's the planning," goes an observation often repeated by those who have confronted projects of apparently overwhelming complexity, from engineers and developers to entrepreneurs to military strategists. While a well-constructed project plan is valuable, it would not be an exaggeration to say that the planning process itself is one of the most crucial aspects of the development of community cyberenvironments. Indeed, it may well be the single most important stage of the project. Sufficient planning is important for two reasons: it helps prevent serious execution problems from developing further down the line, and it helps establish that all-important partnership between technologists and users. It is during the planning process that effective communication and management strategies are developed, and it is also during the planning process that the real issues—what the community wants, what the development team can provide, and what the obstacles are—can begin to be identified.

At the beginning of the project, technologists and users need to jointly establish and document what the overall goal of the project will be: what needs to be accomplished, what can be accomplished, and how it can be accomplished. What may not be entirely intuitive is that what is to be done requires just as much, if not more consideration (and, consequently, planning time) as how it should be done. The project's goals need to be defined clearly. It shouldn't be as broad as "make homes safer" or "prevent hazardous ocean spills." While these are goals that every earthquake engineer or environmental scientist likely shares, they are not helpful in shaping the purpose of a community cyberenvironment project, or ensuring that all participants in the project remain focused on the goal. During the course of NEESgrid's development, the lack of a set of goals and expectations commonly shared with the sponsor, the community, and the developers resulted in shifts which turned out to be costly both in terms of time and energy.

One way to avoid such problems is to develop a **Requirements Traceability Matrix (RTM)**, discussed more fully in Section 4. Because this document explicitly lays out the needs the users originally specified, it serves to remind everyone of the project's initial goals and helps both users and technologists ascertain what a major change in direction will cost in terms of time, effort, and funding. In our experience, it is easy to get distracted and add new ideas or change old ones; the RTM is a great tool to keep discussions between the customer and the technologists focused.

An important part of this decision, and one that may not be entirely obvious, is **clearly identifying who the stakeholders in the project are and what concerns they have**. It may seem easier to simply identify the whole user community or a subset of that community as the group of people with the most interest in the project's advancement. However, in many cases the research community is itself evolving, and it's hard to tell who is likely to be most invested in the project and will therefore actively engage with

and commit to it. We learned this lesson through hard experience: because NEESgrid was NSF's first engineering cyberinfrastructure project, and most earthquake engineers had very little experience with cyberinfrastructure and therefore weren't sure what could be done with it, it was difficult to ascertain who had a vested interest in the project and what they expected from it.

Expect the planning period to take considerably longer for community cyberinfrastructure development than for a single-investigator research project, and expect it to be more formal. Much of the planning for single-investigator research projects often takes place while the funding proposal itself is being formulated, with adjustments and revisions occurring later, particularly if the funded project requires hiring postdoctoral researchers or graduate students. Consequently, individual PIs often see the planning period of a single-investigator project as a mere extension of the process that began during the writing of the proposal, and as a result, it may require only a few months to get the bulk of the details hammered out.

However, because the scale for community cyberenvironment development is so much larger, and the hurdle for production software so much higher, there is a great deal of difference between writing a proposal and writing a project plan. Project planners should expect to take anywhere from six months to two years to formulate a solid blueprint for community cyberinfrastructure development, expect it to continue to evolve throughout the project, and have strategies for managing the plan's evolution. Even so, there will still be may be many factors that can only be nailed down after the project begins.

Expect the planning period, like the development period, to be iterative. In section 6 we discuss the advantages of the spiral development plan for developing software. Through the use of mockups and prototypes to demonstrate to potential users what the project is all about, the spiral approach is as useful for determining *what* is to be accomplished as how it is to be accomplished. Community members can then provide feedback which can be used to make modifications in successive iterations until all reach agreement on the final design.

In developing the initial timetable, it's important to plan—generously—for the planning period itself, as well as to ensure that there is more than adequate time built in for the development phase. In large part because NEESgrid was one of the first projects of its kind, the SI team underestimated the amount of time required for both the initial planning period and the project as a whole. The eventual duration of this planning phase was about a year—considerably longer than expected—and in retrospect, those involved in the original planning process believe that it may have been better spent generating pilots and prototypes and soliciting user feedback. The lack of emphasis on allowing users to interact with early prototypes resulted in major changes in project focus further down the line, all of which decreased the time remaining to develop the software. The NEESgrid project was intended to reach completion in less than four years. Projects of similar size and scope, however, now commonly take anywhere from six to ten years from planning to completion.

Ultimately, the NEESgrid roadmap became one of our most effective planning tools. Done in Microsoft Project and based on a Gantt chart, it was instrumental in helping the project participants determine the order in which to undertake development and deployment efforts, manage risk, and ensure that the integration effort would meet its objectives and schedule. The roadmap was critical for coordinating a large, distributed group consisting of many geographically remote teams and individuals. In developing and reviewing the roadmap, the team confronted critical issues such as the amount of effort required for any particular element and a meaningful risk assessment that helped clarify where significant contingency funds should be assigned during the development process.

Whatever form planning documents take, keep in mind that they are not static entities, but living documents. Changes, both major and minor, are inevitable. Planning documents should be used to note where the plan is less concrete and define in a process and schedule to work those parts out. All revisions, as well as decisions for software directions/selections, should be carefully documented and should always be transparent. In the case of NEESgrid, the roadmap was revised several times and reflected changes in leadership and approach as more people came to participate both in NEESgrid and in the planning process, and as more domain scientists—members of the user community—took a larger role.

Like the development process, the planning process must also be agile and interactive. A planning document may be exhaustively detailed, but if it is developed without benefit of continual user feedback, it loses much of its value. In fact, slavish adherence to a project execution plan without regard for critical changes that would make the system under development more worthwhile for users is a serious mistake. Early in the project, the NEES SI Team produced a Project Execution Plan intended to set forth the organization, systems, and overall plan for managing the project. In the end, it was abandoned for more flexible, less time-consuming planning tools, and in retrospect, many former SI Team members agree that the time spent developing the plan could better have been spent in developing prototypes that would give the user community the opportunity for more engagement in the planning process.

Before finalizing your project plans, a number of increasingly complex pilot or prototype projects should be conducted that build upon and advance the infrastructure and its capabilities. This approach can help establish that the technologists and users are working toward the same goals, clarify what will work and what the community will need, and help determine who should make up the development team before committing large amounts of resources. These prototypes have an additional benefit of demonstrating progress to the community and to the funding agencies and are useful in attracting other members of the community.

SI team members have concluded that NEESgrid could have benefited significantly from the development of such prototypes during the planning process. As it happened, more than halfway through the project, NEESgrid did develop two experimental setups that could be considered prototypes: MOST (the Multi-Site Online Simulation Testbed) and a

smaller-scale version of this setup dubbed "Mini-MOST." MOST was a joint project between UIUC, the NEES SI team, and CU-Boulder that involved simulating, over a large geographic distance, the response of a two-bay frame structure to an earthquake. The two external physical supports of the structural frame were at UIUC and Boulder; the central, inner support, however, was virtual and located in a computational server running simulation software at NCSA. Despite its late appearance in the project's duration, the experiment gave the user community a much clearer sense of what NEESgrid could do for them—and just as importantly, what the limits of the project were.

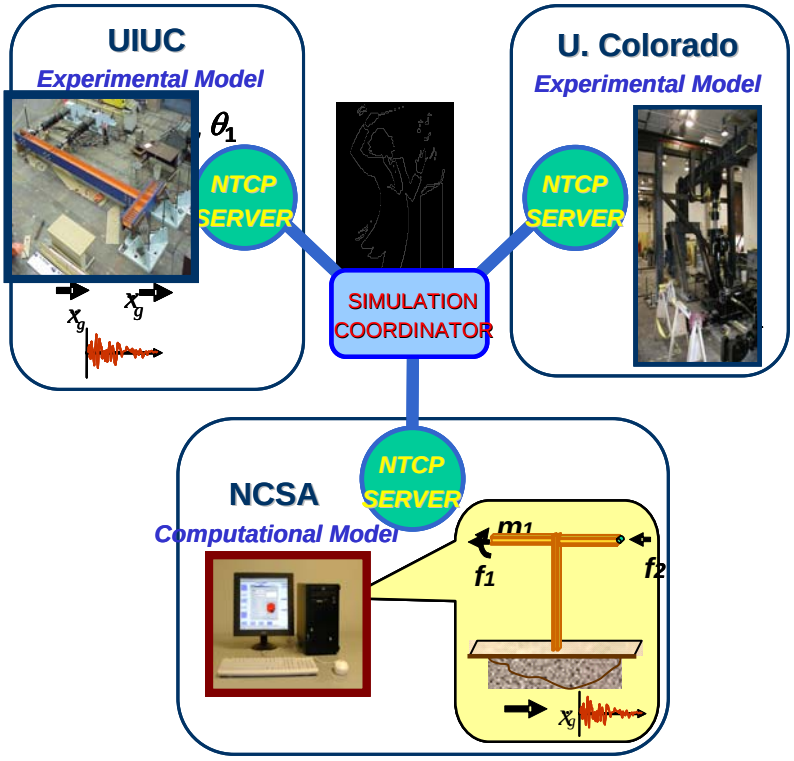

Figure 2: The MOST Experiment.

Mini-MOST was also run successfully as an international experiment with one part of the experimental frame in Tokyo and the other at UIUC. Not only did its portability and relatively low cost make it possible to give demonstrations easily at the NEESgrid site, but it also provided an excellent training tool for new users. In 2004, students at the Colorado School of Mines and USC traveled to Japan to help construct and conduct a mini-MOST experiment which connected equipment sites at their respective institutions with an equipment site at Keio University in Tokyo. The experiment was the first NEES multi-site simulation test to incorporate a structural control device. The students were participants in the Research Experiences for Undergraduates in Japan in Advanced Technologies (REUJAT) program organized by Prof. Shirley Dyke at Washington University in Saint Louis and Prof. Makola Abdulla at Florida A&M, both of whom also

were extensively involved in adapting the mini-MOST for use by non-NEES institutions for research, education, and outreach.[6]

NEESgrid benefited considerably from making projects such as MOST and mini-MOST part of the planning process. **However, in retrospect, it is now clear that because of the project's overall complexity, earlier and more frequent pilots could have greatly benefited the project**.

More recently, and in a different context, a pilot study was used effectively by developers at NCSA and biologists at Michigan State University involved in developing the Long-Term Environmental Research Network (LTER). They explored how grid technology could be used to create a web-based environment for analyzing acoustical data. They created a "Biophony Grid Portal"[7] that demonstrated how researchers could match up the digital signatures of sounds present in a recording against those in a digital database. The Biophony Grid Portal was demonstrated at the LTER Coordinating Committee meeting in September 2005 and again in November at Supercomputing 2005. This demonstration accomplished several things: it demonstrated to the LTER community that grid middleware technology could be used to support the community's research goals; it served as an exploratory study that identified which middleware components could best meet the requirements of a production version of an LTER Grid; and it helped identify scaling as a specific, important issue that Grid technologies could address. Most importantly, however, it drew the technologists and researchers closer together and helped cement a partnership crucial for undertaking a longer-term, larger-scale project. In other words, it served to break the ice: a pilot project provides a way for technologists who have no prior relationship with the community not only to knock on the door, but also to walk in and introduce themselves.

Take more time for documenting and developing the design up front. Doing so can prevent major headaches down the road. During the planning process, take time to plan the entire lifecycle of the project, including requirements gathering and analysis, design of the software, development activities, software documentation, unit and systems testing, the software release strategy, packaging and distribution, and deployment and support strategies. Don't forget to build in plenty of time for integration at different stages of the software release cycle.

Leverage where possible. You will save yourselves an enormous amount of effort and cost—and your users an equally enormous amount of grief—by employing any existing software which does what you want it to do and fits into your design. NEESgrid took advantage of many already-available suites of services and applications such as Globus, developed at Argonne National Laboratory and the University of Southern California, which provided Grid support; CHEF, developed at the University of Michigan, which provided collaborative tools that proved extremely user-friendly; OpenSees, a powerful computational framework for developing earthquake simulation applications; and Data

[6] http://cive.seas.wustl.edu/wusceel/minimost/Keio.htm
[7] http://www.grids-center.org/news/GCNdocs/biophony.asp

Turbine, a commercial product developed by the New Hampshire engineering firm Creare, allowed users to scroll back through data while the experiment itself was running.

Leveraged software, whether open-source or commercial, still must be integrated and can present its own set of complications. A cost-benefit analysis of all alternative solutions in terms of both development and longer-term maintenance should be conducted. If integrating the software requires too much time, or its requirements end up drastically limiting the functionality originally promised to the users, it may not be worthwhile. Finally, a key factor to address when looking at such software is whether it has long-term support. If the software you choose to integrate is not updated on a regular basis to repair bugs and resolve incompatibilities, it may in the long run be more of a problem than a solution—and you do not want to leave that kind of headache for those eventually responsible for maintaining the completed system.

3 Leadership should be a partnership between technologists and domain specialists.

As we emphasized in Section 1, it is crucial to envision the project as a partnership between developers and the user community from the earliest planning stages through the final phase of transition. In the end, however, one person generally makes decisions regarding important issues such as priorities and distribution of resources to meet documented goals. Here, the project director plays a critical role.

We must emphasize, first, that the project director must demonstrate general leadership qualities. Is this person strong and authoritative, with credibility within his or her field? Can this person articulate a vision for the project that can integrate the domain needs with the cyberinfrastructure and see it to its realization, or does he or she tend to be reactive? Can this person make a significant commitment of time and resources (our recommendation is as close to 100% as possible) to the project without having to deal with the distractions of other demands on his or her attention? Does he or she have the communication skills necessary to make sense of confusing technical explanations for an audience new to the technology? Is he or she willing and able to make effective, timely decisions when faced with multiple technology choices and/or technical problems? What kind of leadership style does he or she possess, and does it mesh well with the group dynamic and serve the best interests of the project as a whole? These are attributes important for leading any large project, and it is crucial that whoever takes on the leadership of a community cyberinfrastructure project possesses them.

Project management is a team effort, and there is no substitute for technical experience. The project director should possess a strong understanding of computational technology, available resources, and the development process. However, in bridging the research and technologist communities, the project director must also possess a strong understanding of the needs of the user community. Where there is significant overlap between the user community and the technology community—for example, in certain of the physical sciences where scientists are often themselves also technologists—a member of the technologist community can often easily take on the role of director. However, our prior experience has suggested to us that where users and technologists initially have little in common, a domain specialist is a more effective leader, mainly because he or she, as a member of the user community, has a great deal of credibility among fellow engineers or scientists. NEESgrid, we found, turned out to be just such a case in point.

Where the user and technology communities need to be bridged, the domain specialist can provide the best of both worlds: an intimate familiarity with the specific needs of the community and a familiarity with the technical development process, which helps ensure more effective and realistic development activities. In his or her capacity as director of a community cyberinfrastructure project, this person's most important goal is to lead effectively while creating and sustaining a partnership between research community members and technology developers.

- **A domain scientist—someone well-respected in the community—is often a more credible advocate for the technology than an "outsider" technologist.** Members of the community need to feel that the project leader—often its most visible representative and chief cheerleader—is someone who shares their priorities, someone who has the same things to gain or lose by adopting the proposed technology, someone they can trust readily because he or she is also invested in the technology and the effect that it will have on his or her future research.

- **There are some issues of perception with which the leader of a project involving both IT and domain specialists must grapple to maintain harmony between the two communities.** The project leader needs to be aware of issues such as best-practice disagreements or perceived discrepancies between funding levels needed to do software development and faculty research and must be able to explain and clear up misunderstandings that can cause rifts and resentment. As we emphasized in Section 2, the community has got to see the greater good in the project and understand how they will ultimately benefit, both individually.

- **A domain scientist is more likely than a technologist to be able to explain to the user community what the technology is, how it works, and what it is supposed to do, using language, ideas, and a communication style that make sense to them.** In Section 5, we discuss the importance of good communication between the development team and the user community in more detail, but let us just point out here that each research community has its own common vocabulary that may mean different things to other research groups. Users may tend to find the explanations of how a particular component works much clearer and to the point when provided by one of their own, who can "translate" unfamiliar technology concepts into familiar scenarios more effectively than technologists from outside the community—or, conversely, who can act as interpreter for the technologists.

- **A domain scientist can identify the community's most serious questions, concerns, and criticisms.** Inevitably, there will be resistance and even skepticism regarding the new technology from some parts of the community. Within a given community, the people who are the most vocal do not necessarily express the problems that are most important to the community. For outsiders (such as technologists), it is often hard to discern the community's real priorities. An expert in the field can more readily recognize which issues are most important to a community and help develop appropriate strategies to ensure the success of a project.

- **A domain specialist in charge can help technologists to see things from the community's perspective and do things in ways that may seem counterintuitive to the way a development culture usually approaches projects.** It is in the nature of developers to keep a project on the cutting edge, to keep adding new features and capabilities. At first glance such new features and

capabilities may seem positive and exciting, but they may be counterproductive for a community's current needs. More importantly, they risk slowing the project down or derailing it altogether as the developers become bogged down in their own vision of what the project should be.

- **A leader from the user community will understand how other community members conduct their research, and, as a result, will be aware of what kinds of software, tools, and services are out there that might be leveraged.** Such awareness can, during the planning and design period, prevent technologists from inadvertently reinventing the wheel. At the same time, a leader who is fully aware of the rate at which cyberinfrastructure development advances can better communicate to the user community the need to keep the project at the forefront of computational advances. This point is especially important when one considers that the development period for a given cyberenvironment project can be three to five years, with an expected operation/maintenance period of ten years or more.

As the user community comes to better understand the project and its potential, its members may start to view the project's visions and goals differently. A certain amount of change and adjustment is to be expected. However, a domain scientist who is intimate with the development process can help negotiate these possible shifts in requirements so that other members of the user community understand the consequences of changing goals in mid-cycle.

Having a technologically savvy expert from the earthquake engineering community in charge of NEESgrid helped both the earthquake engineers and the developers reach a working middle ground on a number of issues, such as the spiral development model, which we discuss in more detail in Section 6. Project managers need to have a view of the entire development process and to be comfortable with the knowledge that it will involve uncertainty as parts of the design either are proven to work or need to be discarded altogether. The spiral development model involves a substantial degree of uncertainty over a longer period of time. In the case of NEESgrid, this process proved frustrating to those members of the earthquake engineering community for whom getting equipment sites on line was a priority, and who needed earlier, relatively precise answers about the system as a whole. To help alleviate their concerns, the NEESgrid director pushed for more quality control and quality assurance, as well as for more testing and validation of the integrated software than is often implemented during development. This approach greatly increased the confidence of the earthquake engineering community in NEESgrid's overall reliability.

Ultimately, however, leadership is still a partnership between domain specialists and technologists. We must also stress that a strong technology team is necessary, with sufficient breadth, depth, and commitment in both the cyberinfrastructure and research applications fields to keep up with rapid changes in cyberinfrastructure development and to design the cyberinfrastructure that will best meet user requirements. This team must include technical leaders within the overall management team who have solid project management skills and who are able to see above and beyond the technology itself

without being married to a specific solution. Even more importantly, they have to be able to analyze the problem with which they are presented without reformulating it to fit their preferred solutions.

NEESgrid's director shared much of the decision-making process with a deputy director for the project who represented leadership for the project's technologists. While these were two of the project's most visible individual leaders, there was, as we mentioned in Section 1, a large group of subproject leads. Coming as they did from the earthquake engineering community and the technology community, and from institutions across the country, they provided specific kinds of leadership in different aspects of the project, and they were consulted on a regular basis. The project's chief architect orchestrated the enormous work of technology integration, and social networking experts spearheaded the crucial task of requirements gathering within the user community and developing and maintaining the Requirements Traceability Matrix, which kept everything on track. Other project leads worked closely with the user community, helping the sites get their NEESgrid nodes up and running, collaborating on the new focus on data management, and providing advocacy for both the project and the users. The project manager oversaw the day-to-day activities of the project and the administration tasks associated with it. And beneath this top layer of leadership, other SI team members took on crucial leadership roles in areas such as developing applications and creating the final NEESgrid package. NEESgrid's success depended on the crucial skillsets these members brought to the project as a whole.

The bottom line: choose carefully when selecting people for leadership roles. Making changes is always difficult down the road. This is true not only for the project director, but also for people who are tapped for management and advisory positions. Disruption is always likely when there are changes at or near the top. Partly because NEESgrid was one of the earliest major community cyberinfrastructure projects, lessons about effective leadership had to be learned through trial and error, and the disruption that followed errors created some setbacks. The appointment of a project director who had the attributes necessary both to lead NEESgrid effectively and to strengthen relationships within the earthquake engineering community was pivotal to putting NEESgrid on the road to success.

4 Effective project management is essential at all levels.

Project management should include the project leadership team, external advisory groups, funding agency liaisons, and representative user community members who provide input.

Of all these groups, the management team—your leadership team—is the most important. NEESgrid's management functions were coordinated through a multi-level management strategy. At the highest level was a Project Director from within the earthquake engineering community, a Technical Project Director experienced in large software development and deployment who provided leadership and guidance on IT issues, and a Project Manager who oversaw the day-to-day activities and responsibilities of team members. Interacting constantly—daily, or even several times a day—this team made decisions based both on their own experience and on the input of the entire management team. The management team, in turn, included lead investigators from all of the major project areas—a range of technical activities such as data control, data acquisition, documentation, and collaboration, and outreach activities such as site deployment, training, and demonstration/experiments. This larger management team, which met on a weekly basis throughout the length of the project to discuss project status and issues, was key to the coordination of activities across the entire project.

The combination of technical leadership from the area leads with direct management of staffing and budgets from site leads allowed each team to focus on specific responsibilities. It also freed the technical discussions from site politics and funding issues. Often, the area leads were able to identify needs for additional work and staffing, issues that the Project Manager documented and worked out with the site leads. This separation of responsibilities allowed both teams to focus exclusively on the issues on which they could act directly.

Make effective use of project management tools. While the success of the project depends on many things, not least of which is skillful leadership and coordination by the management team, a variety of invaluable tools are available to help you stay on track, on deadline, and on budget. NEESgrid benefited from the use of several such tools, including:

- **A Gantt chart.** We used a variation of a Gantt chart, a timeline or road map, which kept us on track, telling us visually what tasks needed to be completed when and how. Reviewed on a weekly basis and constantly updated, it allowed the management team to track outstanding or uncompleted tasks. The Gantt chart provided an effective visual tool that was targeted at the management team and an effective way to provide progress updates to NSF.

- **Effective meetings.** For the NEESgrid management team, which consisted of ten individual project teams scattered across the country, getting all progress reports heard in a reasonable amount of time on a biweekly basis was a challenge. A

"green-yellow-red" quick-hit list format was instituted where issues were summarized before the bi-weekly teleconference for everyone to review. During the teleconference, each project lead got three minutes to talk about serious problems, glitches, and delays (red), work in progress or potential problems (yellow). Successes and completed tasks (green)—the kind of subject most people would prefer to talk about at meetings—often were simply never broached; there wasn't time. When a presenter's three minutes were up, their time was cut off. The remaining meeting time was set aside to address problem issues.

- **A Requirements Traceability Matrix (RTM)**[8]. The RTM provided a formal representation of user requirements aligned against system functionality and was used to ensure that all requirements are being met by system deliverables. NEESgrid's RTM listed community requirements down one side and system capabilities across the other, allowing for a straightforward discussion of what requirements were being met, and how, as well as identifying those that were not addressed. The RTM captured, in a simple document, all driving requirements and served as a point of communication between the SI team, NEES community, and NSF. Most importantly, it served as a way to manage expectations on the part of the user community, in that it served as a constant reminder of what the users had originally expected.

It is important, however, to keep in mind that project management tools are only as good as the project management team's commitment to them. Schedules and roadmaps mean nothing if there is neither accountability nor an incentive to continue using them. Publicize your schedule—and keep to it.

Project	Contact	Status May 12	Current Status
Experiment-based Deployment	Doru/Sridhar	Yellow	Yellow
Education/Training	David	GREEN	GREEN
Mini-MOST Outreach	Shirley	GREEN	GREEN
Acceptance Planning/Testing	Lee/Sridhar	Yellow	GREEN
Data Turbine	Laura/Chuck	RED	Yellow
Computational Simulation Component	Greg/Tomasz	Yellow	Yellow
PNNL E-Notebook	Jim	GREEN	GREEN
Data Models	Kincho/JP	GREEN	GREEN
Experiment-based Development @ Uminn	Chuck	Yellow	Yellow
Experiment-based Development @ Fast-MOST	Sridhar/Laura	GREEN	GREEN
Data Infrastructure	Joe/Randy	Yellow	GREEN
Packaging/Releases	Doru	GREEN	Yellow
Documentation	Cristina	GREEN	Yellow
Transitioning	Bill	GREEN	Yellow
Operations	Doru	GREEN	GREEN

Figure 3: Items on the "red-yellow-green" hotlist, which helped to keep project management team meetings short and effective.

[8] http://neesgrid.ncsa.uiuc.edu/documents/TR_2003_13_v1.1.pdf

5 Communication is crucial.

Effective communication is important, among development partners, between developers and the community, and between developers and outside groups. Good communication on the part of all parties and an effective feedback loop are extremely important for the success of a collaborative project.

Communication between Developers and the Community

A major challenge in creating cooperation within a community cyberinfrastructure development project is bridging the communication gap between the development and user communities. One consists of computer scientists, programmers, and software engineers; the other may consist of scientists, engineers, or other researchers to whom cyberinfrastructure may be a largely new and unfamiliar way to conduct research. Each community possesses its own expertise and, more significantly, its own work culture and language, with little overlap.

Clear channels of communication must be established. Both the technologist and research communities are extremely diverse groups in terms of needs, roles, and skills, and it's important early on to identify leaders and representatives on both sides who are able to answer questions and articulate concerns knowledgeably. Within NEES, this was not entirely apparent early on, which generated much confusion regarding the needs of the user community and, more specifically, the equipment sites. The funding agency, the user community, and the technologist community should determine who speaks for which interested group.

Users are not all the same. Developers throw around the term “end-user” as a way to identify all customers of their product. The problem with this approach is that it does not recognize the differences between categories of users. Such an approach is almost guaranteed to frustrate efforts at communication.

During the NEESgrid project a wide variety of different kinds of users were identified, each with its own distinct set of goals and requirements. To give you a sense of how complex the user community can be, we describe different types of users briefly below.

- **Site resource providers** are responsible for the operation, maintenance, and support of site experimental equipment. Their needs include understanding the observational and remote control capabilities of their equipment to remote researchers, as well as data capturing and archiving.

- **Site IT staff and site researchers** are responsible for the operation, maintenance, and support of both computer hardware and software comprising the NEESgrid infrastructure, these users fall into two subcategories. *System administrators* need information about computer hardware requirements, upgrades, and maintenance, as well as software issues such as maintaining and supporting both the system

software and the NEESgrid software stack. *Site researchers*, on the other hand, are concerned with the interfaces between the NEESgrid software and the site's experimental equipment; need well-written documentation describing the NEESgrid software application programming interface (API) of the NEESgrid software services.

- **Remote researchers and practitioners** are interested in sharing experimental data, or viewing or actively participating in a distributed experiment. This community needs to understand the capabilities and use of the NEESgrid software in order to participate either as observers or active participants in an experiment. Researchers involved in participating in distributed experiments need to coordinate with staff at each of the collaborating sites to prepare for and coordinate an experiment in real time.

- **NSF program officers** represent the funding agency staff interested in viewing an experiment or in observing community interactions and collaborations. NSF program officers also must have an understanding of the cyberinfrastructure capabilities under development. Because some people in this community may have only a more general understanding of the technologies involved, user documentation must be of the best quality. Do not underestimate the time and effort required to communicate effectively with the project funding agency. They have different needs and, often, a different perspective that technologists need to understand clearly and appreciate.

- **Application developers** develop software that interacts with and/or makes use of NEESgrid capabilities and/or services**.** For instance, in order to develop software to control a distributed experiment among multiple sites, application developers needed a more detailed understanding of what the NEESgrid components were, how they worked, and why they were necessary. Good documentation and training aimed clearly at application developers was therefore provided.

While not every cyberinfrastructure project may include every category of users described above, it will likely include several of them, as well as some not mentioned. Technologists should work to document all the kinds of users with whom they will need to communicate and how to best serve their unique needs.

Communication is a two-way street. Developers and users often approach the same problem from two different frames of mind. Developers may strive for elegance of design, while users want something that is practical and can address their needs as soon as possible. By themselves, both approaches can be flawed: one may result in a "perfect" design that may simply never be practical or useful, while the other may result in a set of disparate tools that turns out only to be a quick fix and becomes outdated too rapidly to be useful. It is important for technologists to make sure that applications researchers have a clear understanding of what a given project can or cannot deliver, and that both are using the same terminology. Likewise, it is absolutely crucial for members of the application community to ask for clarification when descriptions are unclear, to

give considerable thought to what needs the project should address, and to express those needs clearly to the development team.

The NEESgrid collaboration experienced initial tension because it was, in a way, a collision between two such disparate cultures. As a result, the project team made a number of assumptions that turned out to be erroneous, creating difficulties in defining the community's initial requirements. In some cases, language was at fault—specifically, differing terminologies and differing use of common phrases. A word that means one thing to a computer scientist may mean something very different to an earthquake engineer. Consequently, we learned that understanding, mapping, and using the language of the applications community was crucial to successful community building.

As we discussed in Section 3, judicious selection of a qualified project director from the user community can go a long way in bridging this cultural and linguistic gap. But there are other aspects of communication that need to be reinforced for a successful relationship between the development and user communities.

Choose carefully who speaks for the technologists. The technologists among us freely acknowledge that many software engineers are less noted for their communication skills than for their development prowess. As a result, they may not always be aware of the impact their words have on user community members. A common miscommunication is the confusion of what, in the abstract, is technically "doable" versus what is "doable" within the context—and constraints—of the project itself. When a developer says, "Yes, that can be done," he may merely mean that it is theoretically feasible to implement a particular feature. However, without the intercession of the management team—who is both aware of the priorities and constraints of the project in a way that the developer may not be—and who also has the power to say no—the user may then interpret this as a commitment, which can cause conflict within the community when it proves impossible to make good on the promise.

At the outset, users need to be able to see the end goal as clearly worthwhile. Developers are paid to create the software, but users are compensated little, if at all, for their efforts in adapting, learning to use, and testing the software. Users need to perceive that they are getting something important out of the project and that their input is valued in order for them to be fully engaged from the beginning.

If users do not know what to do with the software, or why they should use it, they will never use it, because they do not know they need it yet. Ideally, you want the users to know what the benefits will be to them early in the project in order to initiate a feeling of excitement and anticipation about the project. We cannot emphasize enough the importance of early and frequent pilot projects, which can most effectively give users an idea of what the software can do for them, as well as a starting point for improving design and functionality.

Do no harm. Users are often committed to existing solutions. They want to know what they can do with the new tool that they cannot already do with existing tools.

Technologists, on the other hand, often approach the users with existing solutions in mind. As a result, technologists often force on users a set of solutions that the users neither understand nor need, ending in frustration and wasted effort for all concerned. Technologists need to understand why and how users already do things. With the new tool, users should be able to do what they already do and either make that easier or, at the very least, not make it any more difficult. The new tool should only provide new capabilities and improvements to the current way of doing things. If no way is found to improve on the existing technology, it might be a sign that the project's overall goals and objectives need to be adjusted. As discussed in Section 2, before beginning the design process, existing tools should be identified that will provide the functionality the user community needs. The rationale for building a new tool should be clearly articulated.

Developers need to ensure that users are trained sufficiently in the use of a tool before assessing whether the tool meets the needs of their community. While users should be responsible for making a sincere effort to try the tool out, technologists should be sure to allow for enough time for users to learn and apply new tools before determining that the tool simply doesn't work and discarding it. Rigorous user testing is also necessary to eliminate bugs and glitches and ensure that users are satisfied with the final product.

Users need to be able to take ownership of and responsibility for their infrastructure. Mechanisms to support the user community as they take responsibility for their infrastructure need to be developed in advance. Requirements of what will be needed for the community to take ownership, maintain, and evolve the project need to be clearly defined early in the project in order for the community to hire, train, and involve their own technical staff while expertise from the IT team is available.

Do not overestimate the community's grasp of the new technology and the development process. Technologists need to ensure that the community understands fully both the purpose and the use of the software. At the same time, do not underestimate their ability to learn the new technology quickly.

Demonstrations of prototypes should be planned to solicit feedback, to demonstrate progress, and to build the community's confidence in the implementation. Providing a research community with concrete examples of what a system will be able to do is essential to increasing the likelihood that it will be successfully accepted and adopted. This is the first step in this iterative approach to software development, discussed in Section 5, which involves implementing a prototype system early on at a cooperating early adopter site and, through collaboration with that site, making incremental adjustments and upgrades. Enabling the applications community to see what they will be getting and what it will be able to do for them can strengthen their relationship with the developers; in addition to demonstrating progress, it also provides a way for the user community to give feedback and help shape the final product to a much greater extent, particularly in the case of data and collaborative tools.

Several such demonstrations took place throughout the development of NEESgrid, beginning with a prototype demonstration at the University of Nevada at Reno in late 2002 that remotely tested three shake tables; a three-site test in July 2003 called the Multi-Site Online Simulation Test (MOST) that combined physical experiments at the University of Colorado at Boulder and UIUC with numerical simulation at NCSA; and the NEESgrid Reference Implementation,[9] which consisted of two miniaturized versions of MOST that sites could use to test their installations.

In retrospect, we believe that there should have been more such prototypes and demonstrations—and earlier. **We recommend, in fact, that on the very first day of the project, the project team should make a "mockup"—in as simple a form as a set of presentation slides—available to the user community.** Doing so helps not only to give the users a good sense of what they will be getting, but also to clarify what users need at the beginning of the design process.

As community engagement increases, continuously gather and update user requirements in a formal and systematic manner that allows for sound decisions about future software directions. Map the user requirements into your requirements traceability document (described in Section 2) to show specifically how each user requirement is mapped into the software's functionality.

Workshops where the users participate to learn about project status and prototype demonstrations provided valuable ways in which to build the relationships between the IT and the scientific community. During NEESgrid a few training workshops were held, even before the final software release, to start familiarizing the system administrators of the local NEES hardware with the software and to get them better acquainted with the software developers. Such workshops are valuable also for gathering feedback on the system and understanding documentation needs.

[9] http://cive.seas.wustl.edu/wusceel/minimost

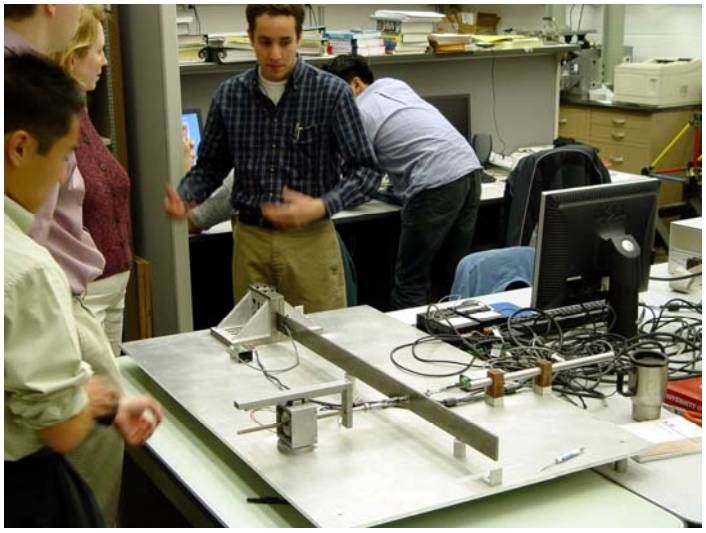

Figure 4: Workshop held at Newmark Laboratory, UIUC, April 14, 2004.

Communication within the Development Community

It should go without saying that a close-knit team of talented developers is essential to any project. Brought together in 2001, the NEES System Integrator team was just such a group, counting among its numbers the key developers and architects of several important technologies that would play major roles in the evolution of cyberinfrastructure development, such as Globus, CHEF (now Sakai), and many of the tools developed under the NSF Middleware Initiative.

Integration should be a focused effort: so much so, in fact, that these projects should emphasize integration; development is secondary. Obviously, development will be needed to fill technology gaps and to aid in integration, but all team members need to understand that they are building an integrated "system," and that close cooperation is required. Including all developers, testers, documenters, trainers, and software release engineers in the integration effort ensures that software versions are uniform, accomplishes the component integration tasks more efficiently, and facilitates testing the release as a complete system. In short, the team must be working from a common architecture and design principles with the goal of integrating the collection of software. Documentation is important, but equally so is effective communication among the development teams.

Do not let dates slip during development phase. Admittedly, this principle can be hard to follow. Despite the best planning, unforeseen obstacles can still prevent the delivery of a specific feature by a particular date, and such delays can strain the relationship between the technologists who are providing the functionality and the users who are expecting it, whose research may depend on its being released by a certain date. Such problems can be avoided by approaching software releases in two different ways. The first approach is a timed release, in which you target a specific date and release the software on that date regardless of what new functionality it includes. The second

approach is a functionality release, in which a particular date is not specified, and the new functionality is released when it is ready. For our part we found that a timed release was a better strategy for keeping the project on schedule. We augmented this strategy with feature releases (as they became available) interspersed between the timed releases.

Focus is extremely important; immersion works. One important way NEESgrid developers succeeded in sticking to deadlines for final releases was to hold week-long development sessions we called "integration weeks." These sessions allowed the team to immerse themselves in the project without distractions. All members of the team would gather in one place to finish a release, with the goals to be met by the release clearly in mind. Integration week was very structured: each morning and evening, the team would meet to talk about what had been achieved, what the next objectives would be, and whether priorities needed to be readjusted. During the day coffee and meals would be provided—the team often quite literally would not leave the building. Everyone was entirely focused and dedicated to meeting the deadline. As an important side benefit, the week-long immersion strengthened relationships within the IT group and promoted a team approach to resolving problems and making progress toward the goal.

Testing and validation is a key aspect of integration. Each integration week had as its target the production of a well-tested release that met certain criteria established in advance. For NEESgrid, the software and documentation was tested in the Mini-MOST testbed as a way to validate the release. This sort of real testing meant that just getting the software to build and deploy was not good enough; it also had to work as documented. In addition, we had a set of well-defined acceptance tests that were developed in collaboration with the community. The team utilized these acceptance tests during the integration week as a way to test modules (unit tests) and test the entire system (workflow tests) more fully.

Communication between Developers and Relevant Groups Outside Project

Both developers and the funding agency need to understand the broader significance of what is being done. Cyberinfrastructure projects are often not merely stand-alone projects, but important models that are of interest to other communities. In the case of NEESgrid, developers were dealing with earthquake engineers, but discovered that the project was being watched with interest by other communities, such as environmental scientists, seismologists, and oceanographers, who saw NEES and NEESgrid as the first of many such "observatories" to be built. Members of a cyberinfrastructure development team should think carefully about how their project is being viewed by other communities, and how they can leave a lasting, positive impression.

Relationships with external groups should foster an environment of support and mutual trust. While the project management team's leadership is central, good relationships with other management groups are important. In the case of NEESgrid, the Project Director also coordinated formal communication between the project and the PIs at the equipment sites, the NEES Consortium Development Team, and the National

Science Foundation.[10] The Consortium Development Project, a project of the Consortium of Universities for Research in Earthquake Engineering (CUREE), was responsible for the creation of the NEES Consortium and the submission of the original proposal for building and operating the NEES collaboratory.

The Executive Advisory Board, an independent entity comprising prominent domain scientists and IT experts, served in an advisory capacity to the NEESgrid Project Director. The Board's makeup ensured that both the interests of the earthquake engineering community and perspectives regarding technical fields central to the NEESgrid system architecture and user interface components were well-represented. The EAB met twice a year to review and make recommendations on NEESgrid technical directions, strategies, and project management; recommend strategies for improving communications with the community and with the National Science Foundation; and advise the Project Director as needed on overall administrative issues.

An Executive Advisory Board with sufficient depth and breadth in both the cyberinfrastructure and application fields can be a valuable ally to the management team and to the project team as a whole, providing honest assessments of the team's progress while advocating for the project to sponsors and the community. For this to occur, however, the members of the EAB have to be as committed to the project as the project team themselves. And, just as importantly, there needs to be close, regular interaction—and, most important, trust—between the EAB and the management team. A regular and frequent review of the cyberinfrastructure project by a project external advisory committee can keep the project from becoming too insular and ensure that the team keeps up with new developments in cyberinfrastructure capabilities.

It has also been suggested that having representatives from other user communities on the EAB might prove advantageous, in the sense that tools and applications could be leveraged from those communities to the current project. Likewise, these representatives could explore the possibility of applying the technologies currently under development to technological problems confronting their own communities. In any case, broader communities with whom the user community may need to interoperate should be identified—these may be not only research communities, but educational and international communities as well.

The project lead is not just responsible for managing "down" (within the project) but also for managing "up"—making sure that the sponsor and others are satisfied with the project's progress. A close, positive relationship provides strong impetus for NSF to help move the project forward and provide direction. For that to happen, the sponsor needs to know that progress is being made and that the things that need to be done are getting done. The sponsor needs to stay in touch not only in the matter of status and direction, but in order to help the project leadership to succeed through promoting relationships with other communities and providing awareness of current issues.

[10] http://www.nees.org/About_NEES/History/

Genuine partnership in this context can be challenging; it is sometimes hard to distinguish from micromanagement. The funding agency representative who wants a good working relationship makes frequent site visits, takes tours, and is intimately familiar with everything the project does; at the same time, he or she does not get buried in details but maintains an effective high-level relationship.

6 Good software development practices need to be established.

In any project, managing software development must address issues related to version control, backups, integration, testing, and packaging to name a few. The larger the software development project the more critical these issues become. **Releasing a quality product requires disciplined use of testing procedures and infrastructure.**

Software development practices vary from one institution to another as well as from one developer to another, which means that identifying and imposing a single standard for software development can often be challenging. A more feasible approach is to achieve agreement about some common practices such as modular design, software management control policies, quality assurance through unit testing, development of appropriate documentation, definition of APIs (or Web services interfaces), identification of dependencies, and proper integration testing of components as a whole system.

Some of the common practices that we emphasized for NEESgrid included:

- **Testing and quality assurance.** The testing framework for the software components, as well as the integrated system, is critical to project success. A feature may function as expected one day; make a small change, and it may not work at all the following day. Moreover, all features may work independently, but a change in one component by a single developer may cause trouble with the integrated system. Tests must be repeated frequently—ideally, daily or even multiple times a day if necessary.
- **A modular design approach.** It is important to be able to adopt part of the package and scale up to the full-blown package over time—users may want one part of the bundle, but should not have to take the whole thing. Do not give people too big a bite to handle.
- **Documented code.** All software code must be documented in sufficient detail that it can be readily supported by other developers. Standard code documentation guidelines should be enforced.
- **Documentation of design philosophy and changes to original plans.** Design decisions change for good reasons, but those reasons are often forgotten. If not documented, decision processes tend to be repeated unnecessarily.
- **Source code revision control.** All code must be accessible by all developers in a proper source code revision control system. Otherwise, time is wasted dealing with inconsistent versions and other problems.
- **Additional documentation, including documentation geared to the technical maintainers of the systems/hardware and manuals for the system's users.** During the course of NEESgrid development each component was meticulously documented, and manuals were updated regularly as necessary. The documents were made available online through a sortable database at the NEESgrid website.

- **A spiral development strategy** should be employed to manage expectations in the user community and to get appropriate feedback about prototypes that can be incorporated into future design and implementation. Technologies essential to the development of cyberenvironments evolve both rapidly and perpetually. An iterative, or spiral development model, uses existing implementations to develop prototype solutions while preparing to port prototypes to newer implementations that will be available in the future.

Based on the NEESgrid software development plan, the release schedule of dependent software, the need to develop early prototypes, and the recognition that some requirements were likely to be revised or significantly changed, a plan for the distribution of the NEESgrid software was established. The software distribution plan consisted of three major releases, with minor release in between major releases as required.

- The first software distribution consisted of the basic NEESgrid functionality and was distributed only to a subset of the community identified as *early adopters*. **Early adopter sites were identified as those sites that were prepared to work directly with the system integration team to deploy, test, and provide feedback about the NEESgrid software.**

- The second software distribution was designed to represent a nearly complete set of functionality of the NEESgrid system. This distribution was targeted to the entire NEESgrid community as a way to get them early exposure to the software and capabilities.

- The final distribution was targeted to the entire NEESgrid community and was to differ from the second release only in offering a *hardened* version of the NEESgrid software. In reality, the final distribution included some capabilities that were intended for the second release but were not included for various reasons.

Experiment-based development (use scenarios) enables developers to anticipate revisions to requirements and engage parts of the user community early on in the project. This approach includes the identification of real-world experiments and workflows that can be used to test integrated software services and capabilities throughout the development process. During this phase, members of a system integrator team work closely with a small number of user community sites to develop early capabilities and demonstrate them, while at the same time involving real users closely in the development. The prototypical example of this process was the MOST experiment. Not only did this approach help validate the first prototypes, but new and revised requirements could also be identified early enough in the project to prevent a significant impact on final implementation. These new or revised requirements could be used in the spiral software development model to provide more useful software in the next release.

For NEESgrid, this approach also led to the development of the NEESgrid Reference Implementation, also known as Mini-MOST, which was a physical model representative of a simple experiment that could be used to validate future software in a lab environment. This relatively inexpensive physical experiment was built and used to test later version of NEESgrid software by the system integration team developers without the need to use full scale equipment at any of the early adopter sites. This same apparatus could be used by other equipment sites to validate their deployment of the NEESgrid software before using their expensive full-scale equipment.

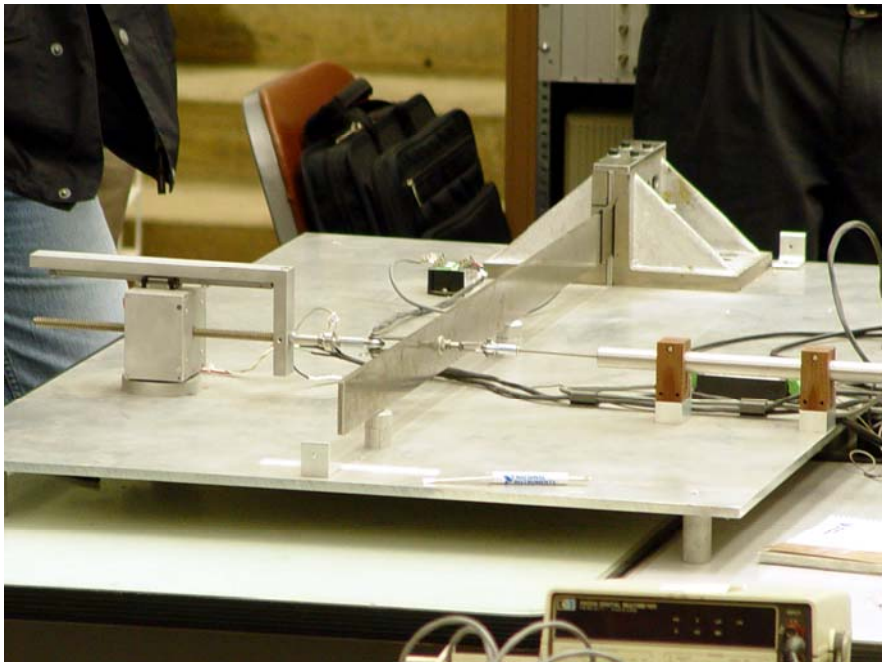

Figure 5: The Mini-MOST implementation in use at a NEESgrid workshop for system administrators in April, 2004.

7 Experiment-based software deployment is effective for helping users to own the software.

Experiment-based deployment provides an effective method for site integration and functionality validation and can help kick off software handoff. The process involves experiments proposed by equipment sites or individuals within the user community on which members of the technologist community collaborate. Experiment-based deployment allows the SI team to test the system's architecture and components, identify new requirements as they arise, and engage a larger part of the community in the deployment effort while at the same time giving them the opportunity to begin doing science.

NEESgrid 2.0 was the point where the NEES SI team turned from experiment-based development to experiment-based deployment, with the goals of understanding and gaining experience with site integration with the NEESgrid software and validating the software using real site-specific experiments. The selection of equipment sites for early deployment was determined by their willingness to work with us as well as their level of technical and IT sophistication. As the target date for the final release approached, deployments became more and more routine. Our objectives for each site included 1) a successful deployment, 2) validation that the software was working at a site, and 3) site acceptance of the software.

For each experiment, the equipment site involved would plan and communicate the basic experiment to the SI engagement team, including such information as the identification of a project lead, technical contacts, experiment requirements, and an estimated time frame for the experiment. Once the SI had confirmed the proposal's feasibility based on the available capabilities and features of the NEESgrid, the SI engagement team worked directly with the ES project lead to coordinate the implementation tasks in preparation for the experiment. After the experiment was successfully run, the ES was expected to provide a report of the experiences and results of the experiment, and to release software that was developed in collaboration with the SI. Reports and software were posted on the NEESgrid web site and were used to validate the status of ES engagement in NEESgrid. These also provided valuable feedback to the engagement team to improve the experiment-based deployment process at the remaining ES sites.

An example of one such experiment involved a tsunami basin apparatus, located at Oregon State University. This was hooked up to both video capture and the NEES data acquisition mechanism, which then stored the experimental data along with metadata in the site's NEESpop server. Both the development team and the earthquake engineers at OSU were pleased with the experiment as a whole. The experiment used the existing SI components available in the NEESpop in a new and different way, the engineers observed that the setup permitted them to do in a very brief time something that would have been very time consuming if performed manually, and the SI team received useful

feedback about improvements that would need to be made to the NEESpop hardware specifications. A more detailed description of this process may be found online.[11]

As sites deployed the NEESgrid software, they had to integrate it with their local environment. The early sites were encouraged to make their implementations available to the community. Often, other sites had similar equipment and setups; these additional reference implementations directly helped other sites get online faster. It was an important advance in that it was the community's first step towards the self-sufficiency that we describe in the final section.

[11] http://neesgrid.ncsa.uiuc.edu/documents/NEESgrid_epd_strat_req_v1.0.doc

8 Cyberinfrastructure is a living entity.

When a development team delivers the finished technology to the community, the community must be prepared to support and possibly extend it. Good communication (Section 5), experiment-based development (Section 6) and experiment-based deployment (Section 7) help ensure successful adoption of the software by the community as a whole. Ideally, the developer should also be the operator.

However, there may be circumstances where, for whatever reason, the cyberinfrastructure development team hands the technology off to a third party to maintain, support, and upgrade it. **In the case of such a handoff, a smooth transition is crucial.** If the developer will not be responsible for long-term maintenance, then there should be **at least a year's overlap** between the developer and the operator to fix bugs and to ensure that the operator understands the software capabilities.

Those involved in the handoff need to work closely together to define a transition plan to define tasks, timelines, and responsibilities among the technologists and the community to which the transition is being made. In October, 2004, the NEES SI team handed off responsibility for maintenance and support of the NEES network to the NEESit Services Center, operated by the San Diego Supercomputing Center at UCSD. Six months ahead of time, the two teams began working cooperatively to put together a detailed transition plan that took into account all the areas that would require attention and catalogued in painstaking detail all the tasks associated with those areas and those people who would be responsible for them. Until the formal handoff on October 1, 2004, the transition was managed as a project in and of itself. With the exception of a modest number of bug fixes and adjustments, the handoff was a success. However, were we to suggest improvements in the process, we believe that closer involvement between the development team and the maintenance team could only help. Specifically, during the first year of operations, both the cyberdevelopment team and the cyberoperations team should jointly operate the software, with the former transitioning into the full operation in year two and the latter helping the user community to use and adopt the software features.

Because the cyberenvironment is a living entity, it is important for the infrastructure support team to remember that the lessons of the original development process continue to be applicable. The original SI team fulfilled all of its deliverables, but neither the technologies on which the system was based nor the needs of the user community stopped evolving. Furthermore, user community adoption does not happen within a year or two. Software needs to be tested and tried by the community to assess what works and what does not. Technologists within or external to a community who are maintaining the software/system need to be proactive in keeping that partnership alive. They must seek out and identify users who can benefit from the software and help them adapt the software to their needs and learn to use it. If no one knows the software is there, or what it can do for them, or how to use it, it will cease to be useful. It is our experience that technology as complex as NEESgrid, as an example of a cyberenvironment, requires solid support and, more importantly, advocacy to build acceptance and enthusiasm and to continue to evolve the cyberenvironment to meet the

needs of the community. Thus, a plan for using for the cyberinfrastructure as delivered and leveraging its acceptance in the user community to extend its functionality has to be well-defined.

The relationship between communities is also a living thing. We began this paper by emphasizing the importance of a strong partnership between technologists and users, and we emphasized that this partnership is important from beginning to end. However, in closing we should emphasize that while it's important that users are able to "own" the technology and become self-sufficient, the benefits of maintaining ties to the CI technologist community are very worthwhile.

Involvement between the earthquake engineering community and the technologist community did not end with the NEESgrid handoff to SDSC. For example, members of the Mid-America Earthquake Center (MAE) are currently involved with researchers at NCSA in developing the MAEViz cyberenvironment,[12] a tool integrating spatial information, data, and visual information into a framework that allows users to evaluate and analyze seismic losses. NEESgrid opened the door to new ways of doing research for the earthquake engineering community.

Cyberinfrastructure projects may come to a close, and the relationship may, over time, undergo changes in the communities' level of involvement with one another and in the nature of technology, but members of the user and technology communities continue to collaborate in other ways as rapidly-developing cyberinfrastructure opens up increasing possibilities for doing research that was never before thought possible. The relationship can benefit both sides for a long time to come.

[12] http://maeviz.cee.uiuc.edu/