

Improving Software Engineering Practice at NCAR:

A Report to the NCAR Director

Lawrence Buja
Chris Burghart
Cecelia DeLuca
Jose Garcia
Richard Loft
John Michalakes
Russ Rew
Eric Scharff
Chris Webster
Gerry Wiener

April 19, 2004

Executive Summary

This NCAR Software Engineering Committee was charged in March 2003 to assess the status, needs, and priorities in the software engineering community at NCAR and to derive specific measures designed to create the cultural and organizational substrate that would permit incremental improvements to the software engineering practices across the organization. This report presents the findings and recommendations of the committee in response to that charge. Care has been taken that the recommendations are specific enough to form a basis for action by the NCAR directorate.

The findings and recommendations of the Software Engineering Committee are the distillate of many months of discussion and reflection. The committee's methodology for developing these recommendations began by divisional representatives obtaining input from their respective divisions and synthesizing them into an overall assessment of the problems faced by software engineers at NCAR. Prioritization and refinement of the committee's understanding of these problems gradually formed through extensive discussion and debate. The guidance and input of Bill Preeg in these early discussions was vital. A consensus was reached which transcends, we believe, the narrow divisional concerns with which the committee began the process. The committee then moved on to formulate possible solutions, refine them and finally vote upon them. The general goals of the committee's recommendations were:

- Improve project management practices.
- Augment HPC engineering resources.
- Improve the professional development of software engineering staff.
- Improve communication between software engineers.
- Encourage better software process practices.
- Address the issue of interdisciplinary software engineering projects.
- Provide oversight and feedback mechanisms to management.

To achieve these goals, the committee makes the following nine specific recommendations:

- To better manage software engineering resources and risks, senior software engineers with relevant expertise should be added as reviewers of significant new NCAR proposals with a large software engineering component.
- To improve their planning and execution, significant NCAR projects should be required, at inception, to provide a Project Plan to NCAR management. This plan should specify at least three items: a work breakdown structure, a project management structure, and a description of the software process to be used.

- To maintain their technical currency, we recommend that software engineering staff engage in at least one week of professional development per year. We also recommend that managerial reporting mechanisms or metrics be developed to track the professional development and its efficacy.
- We recommend that NCAR create and maintain a set of best practice and software practices guidelines for three levels of software development projects: R&D, Research Prototype, and Delivered System/Large Project. These guidelines should include guidance covering software language choice, development processes and tools.
- We recommend that a curriculum of professional development courses should be created by Human Resources, in consultation with software engineering staff, which would work to systematically advance the careers of software engineers, particularly in the areas of software engineering project management, technical leadership and mentoring of junior staff.
- To improve communication between software engineers, we recommend the creation of a software engineering knowledge management (SEKM) system. This system would be a web-based repository, maintained by a SEKM coordinator, that provides a road map to software engineering personnel and resources, best practices, project home pages and a project database, mailing lists, tool evaluations, seminars, etc.
- In order to foster resource sharing, enhance communication and promote mentoring in the science divisions, we recommend that an Application Group specializing in geophysical applications be created as a standing, cross-divisional engineering resource in the newly formed Sun, Weather, and Climate Laboratory (SWCL).
- To facilitate the implementation of these recommendations, and to provide a mechanism for feedback between NCAR management and the software engineering community, the committee recommends the creation of the UCAR Software Engineering Assembly (USEA).
- Finally, we recommend that implementation of these recommendations begin immediately, that it be well integrated with the on-going NCAR realignment process, and in place no later than April 2005.

We estimate the annual cost of implementing these recommendations to be approximately \$480,000/year. These costs could be recovered even if the investment in better software engineering practice resulted in only a 1% improvement in software engineering productivity NCAR-wide. These estimated costs break down as follows: a \$155,000 investment in a software engineer professional advancement curriculum; \$90,000 for a 1/2 FTE to fund the SWCL Application Group; a budget for USEA of \$35,000; and \$180,000/year for a full time SEKM coordinator and a materials and supplies budget for SEKM of \$20,000/year.

TABLE OF CONTENTS

Executive Summary	ii
Table of Contents	iv
1 Introduction.....	1
2 Committee Members.....	3
3 Committee Mission.....	4
4 Methodology.....	4
5 Assessment.....	5
6 Recommendations.....	9
6.1 Project Management.....	9
6.1.1 Proposal Review.....	9
6.1.2 Project Planning.....	9
6.2 Professional Development.....	10
6.2.1 Human Resources Professional Development Courses.....	10
6.3 Software Engineer Mentoring.....	11
6.3.1 Summary.....	11
6.3.2 Discussion.....	11
6.4 Knowledge Management.....	12
6.4.1 Finding.....	12
6.4.2 Recommendation.....	12
6.4.3 Rationale.....	13
6.4.4 Implementation.....	14
6.4.5 SEKM Coordinator.....	14
6.4.6 SEKM Milestones.....	14
6.4.7 SEKM Oversight.....	15
6.4.8 SEKM Resource Requirements.....	15
6.5 Software Process / Best Practices.....	16
6.5.1 Introduction.....	16
6.5.2 Finding.....	17
6.5.3 How is software process implemented at UCAR?.....	17
6.5.4 A basic taxonomy of software projects at UCAR.....	18
6.5.5 Software process elements in use at UCAR.....	19
6.5.6 Recommendation.....	20
6.5.7 Rationale.....	20
6.6 Application Group for Sun, Weather and Climate Laboratory.....	20
6.6.1 Scope.....	21
6.6.2 Benefits.....	21
6.6.3 Costs.....	21
6.7 Requirements-Based Choice of Computer Language.....	21
6.8 UCAR Software Engineering Assembly.....	22
6.8.1 USEA Mission Statement.....	22
6.8.2 Membership.....	22
6.8.3 Organization.....	22
6.8.4 Executive Committee Responsibilities.....	23
6.8.5 Seminar Series.....	23

7	Overall Cost Estimate	24
8	Acknowledgements	24
9	Appendix	24
9.1	Division Assessments	24
9.1.1	SCD Software Engineering Assessment.....	24
9.1.2	Research Applications Division Assessment.....	26
9.1.3	Atmospheric Technology Division Assessment.....	28
9.1.4	MMM Assessment	29
9.1.5	ESIG Assessment.....	29
9.1.6	Unidata Assessment	30
9.1.7	HAO Assessment	31
9.1.8	CGD Software Engineering Assessment	31
10	References.....	33
10.1	Bibliography	33
10.2	Web Sites	34

1 Introduction

What is the nature of software engineering at NCAR? Certainly there is no simple characterization. Under current NCAR practice, software engineering is undertaken by people with a broad spectrum of skills and training, working on a wide variety of problems, on projects ranging in scale from individual to major national efforts. Any of the following activities can reasonably be thought of as falling into the realm of software engineering at NCAR:

- a seasoned associate scientist coding parameterizations for a complex model that runs on parallel supercomputers

- an applied mathematician developing high-performance parallel numerical libraries

- an Earth science graduate student new to programming developing a model of atmospheric tides on a workstation

- a programmer drawn from the commercial domain working with a team on a networking project

- a software project manager responsible for structuring the technical aspects of a large modeling or infrastructure project, supervising the development team, and coordinating with management, scientists, and program managers

Depending on the task, the required expertise can be a combination of science, mathematics, computer science, software design, software process, and management.

Given this diversity, how do we approach the challenge of advancing software engineering in the atmospheric sciences?

We can begin by identifying a number of basic categories describing software engineering projects in our field, and key software engineering issues.

Perhaps the most useful categorization of software projects is by scale. Individuals involved with smaller software projects, like the new graduate student struggling with a computational task, are likely to benefit from basic training in software structure and development processes, and perhaps, depending on the project, high performance computing. Large software projects, whether focused on research or production software, require considerably more organization and staff expertise in both software structure and process to function effectively. For example, major climate and weather modeling efforts in the US and Europe, motivated by the need to manage a large set of contributors, have found it advantageous to create software Change Review Boards, to systematically identify, prioritize, and schedule changes to model software. Such formality would be out of place for an individual working on independent research. These projects also have a need for a number of “translators”, project members who are cross-trained in both the physical and computational sciences and who can traverse

disciplinary bounds. These “translators” are able to maintain a holistic view of the effort and help integrate the science and engineering tasks. Software structure, in the form of a clearly defined architecture, is critical for larger projects if the code is to be understandable and extensible. The new architectures emerging in the Earth sciences are being developed by computer scientists and software engineers with expertise in object-oriented design, component-based design, and multi-programming-language development. For an Earth science graduate student or beginning researcher, exhaustive training in these areas would be both impractical and of dubious value.

Another useful categorization is the set of scientific and technical elements required in the software project. For example, modeling efforts typically focus on scientific algorithm development and require expertise in both atmospheric physics as well as numerical analysis. Scientific visualization requires expertise in graphics programming. Cyberinfrastructure development requires expertise in data management.

In the software engineering literature, two key factors are repeatedly called out as critical to the success of software projects. These are an adequate understanding of requirements and competent project management. The first key factor, understanding requirements, is crucial because early in a project a statement of requirements bridges the gap between developers of the software and future users of the software. For example, requirements analysis can encourage communication between a group of computer scientists who are interested in developing cyberinfrastructure for modeling, but have limited knowledge about Earth science, and a group of Earth scientists who are interested in improving the capabilities of their model but are unfamiliar with the technology and terminology of computer science. Development of requirements can help to clear away jargon, identify the scope of the project, draw out assumptions, and set expectations.

The second key factor is management of software engineers and software projects. For larger projects, the collection of requirements, hiring and coordination of staff, oversight of software design, sequencing of development, and management of relationships with customers require attentive, dedicated management. The trend, in both infrastructure and modeling projects, has been to shift this responsibility away from project principal investigators and lead scientists to a full-time software project manager. As software projects grow ever more complex and critical for the advancement of NCAR science, identifying and training software project managers and engineering leads is a critical area of investment.

Physical scientists and computer scientists have different goals, each to pursue innovative ideas in their own domain. Those goals may not be entirely complementary, and even if they are, many preeminent scientists are specialists and have neither the time nor the inclination to devote considerable effort to understanding the principles and vocabulary of a substantially different field – or even the effort required to

successfully communicate with its practitioners. Resisting for a moment the democratic appeal of endeavors in which physical and computer scientists are equal partners, we realize that from a science point of view the ideal situation is often one in which NCAR scientists express their needs using terminology familiar to them and are handed a production-ready package, as quickly as possible, that satisfies those requirements.

With this context in mind, some strategic questions are:

What software practices and design methods are most critical for smaller projects in the Earth sciences, and how can those skills be taught?

What software practices and design methods are most critical for large efforts? When is it appropriate to train individuals within the Earth science community with the required skills and when is it appropriate to seek expertise from outside our community?

What can we learn from the commercial domain and other fields about the management of software projects?

How can software processes help to bridge the gap between the computer scientists and software engineers who develop software and the Earth scientists who use software?

How can we identify and train software project managers?

How can we best mentor and develop young software engineers?

We believe the strategies that are proposed in this report are ones that are most likely to be successful in a realigned NCAR which has recreated itself as a matrix managed organization in which interdisciplinary, cross-divisional interactions are the norm. From that standpoint, the current plan for NCAR realignment is a positive step that synergizes with the goals of the strategy outlined here.

2 Committee Members

The composition of the Software Engineering Committee was designed to be representative by division or program as noted below:

- Lawrence Buja, CGD
- Chris Burghart, ATD
- Tony Craig, CGD
- Cecelia DeLuca, SCD
- Jose Garcia, HAO

- Rich Loft, SCD
- John Michalakes, MMM
- Russ Rew, Unidata
- Eric Scharff, ESIG
- Chris Webster, ATD
- Gerry Wiener, RAP

3 Committee Mission

- Assess the status, needs, and priorities in software engineering and computer science at NCAR based on discussions with NCAR scientific and technical staff and external experts.
- Synthesize the results of those discussions in the context of the NCAR Strategic Plan for High Performance Scientific Simulation.
- Make recommendations related to software engineering that would require action.
- Write a report summarizing the findings and action items.

The committee's mission has evolved from the charge given to it by the NCAR Director at its inception. In particular, owing to the composition of the committee, it was determined in subsequent discussions with the NCAR Director to be most appropriate for the committee to focus on assessing the status, needs, priorities in software engineering community at NCAR and to derive specific measures designed to create the cultural and organizational substrate that would permit incremental improvements to the software engineering practices across the organization.

4 Methodology

The Software Engineering committee has held regular monthly meetings since March 2003. The first meeting was organizational in nature and led to a certain amount of free discussion regarding software engineering at NCAR. One very strong thread that emerged was the need to create a sense of community for software engineers within the organization.

A website for the SE committee was created, <http://www.scd.ucar.edu/secomm/>. This website contains pointers to source documents, minutes and findings of the committee.

At the request of the committee, in our April meeting, Bill Preeg described the Métier system employed at Schlumberger. The Métier system provides a source point for professional development, tools, and career development for software engineers.

To gather data about the community, SE committee members were asked to survey their divisions with regard to status, needs and priorities of software engineering. Members made presentations of the assessment information they gathered at the committee's May meeting.

In August 2003, the co-chairs met with Tim Killeen and Larry Winter to discuss current progress as well as future committee directions.

During the remainder of 2003, the committee focused on translating the May 2003 assessments into tangible findings and recommendations. A web site, <http://swiki.ucar.edu/secomm>, was constructed using the Swiki collaborative website technology, <http://minow.cc.gatech.edu/swiki>. This technology allowed all committee members to easily contribute input with regard to the evolving set of recommendations without having to edit the web site's html pages directly.

In December 2003, the co-chairs met with the Director's Committee and presented a progress report. The Director's Committee reiterated that it was essential for the software engineering committee to establish a set of actionable recommendations.

After establishing a tentative list of recommendations, the committee members were asked to post their comments on the Swiki site. We continued to meet and refine the recommendations. After numerous revisions, we finally created a draft set of recommendations and asked all committee members to vote on them. Once the voting was completed, the committee members called town meetings in their respective divisions to present the recommendations as well as to elicit feedback. The committee then reconvened to discuss the feedback from the town meetings and subsequently finalized the recommendations.

5 Assessment

The first step in the committee's assessment of software engineering at NCAR/UCAR was to collect the demographic information about the workforce. We found that there are 201 software engineers working across NCAR/UCAR/UOP. Of these, about 75% work within NCAR. This data, further broken down by program/division and by job classification is shown for NCAR in Table 1, and for UCAR/UOP exclusive of NCAR in Table 2. The data were collected in FY 2003. An important caveat here is that the data do not capture the employees with other job classifications who work in the area of software development, such as associate scientists and post-docs. Therefore, the results almost certainly underestimate the population of staff working in the software development area by an unknown amount. Nevertheless, there are some general

conclusions that can be drawn with regard to software engineering at UCAR simply by looking at the distribution of software engineers across the institution.

First, there is a wide variation in the distribution of software engineers across UCAR. SCD with 60 and RAP with 47 account for more than half the total, while ESIG and COSMIC have only one apiece. This simple demographic fact introduces dramatic differences in the mentoring and support environment software engineers experience across the institution.

Second, it is clear from the job classification data that NCAR/UCAR requires a highly trained and experienced cadre of software engineering professionals. Indeed, 68% of software engineering staff are either SE-3 or SE-4. The relative paucity of SE-1 staff indicates that there is almost no demand for entry-level software engineers in UCAR. The data indicate that attracting, training and retaining high quality software engineers should be a strong institutional priority, based on the apparent demand for this skill level across NCAR/UCAR.

As mentioned in the Methodology section, the committees also obtained input directly from the software engineering community by each member conducting assessments within their respective division or program.

We found from the assessments that UCAR software engineering problem domains are quite diverse in the various divisions and programs. Owing to the difference in problem domains, a wide variety of skill sets as well as engineering approaches are required. A brief list of focus areas includes creating software infrastructure for high performance computing, scientific model development, tools for data access and sharing, data analysis software, mathematical libraries, visualization software, user interface design, field project support, real-time system software, and operational system development.

Not only are there a wide variety of problem domains at UCAR, there are also a variety of different funding sources. Even though the National Science Foundation remains the most significant funding source for the institution, a large number of projects are funded by NASA, FAA, DOD and foreign governments. The requirements levied by the different funding institutions will influence the approach taken in various projects. For example, the level of reporting, documentation and testing required by the DOD is quite different from that required by the NSF.

Table 1. Breakdown of NCAR software engineers by division and job classification.

Division	SE-1	SE-2	SE-3	SE-4	Total
ACD	1	2	4	1	8
ATD	0	2	9	2	13
CGD	0	4	10	3	17
ESIG	0	1	0	0	1
HAO	0	2	2	1	5
MMM	0	2	1	1	4
RAP	3	16	16	12	47
SCD	2	6	29	23	60
Total	6	35	71	40	155

There is a wide variance in the set of software engineering management practices across UCAR divisions. In particular, some divisions have established organizational entities and management structures consisting of the software engineering staff while other divisions have not done so even when having a sizeable number of engineers in the division. In conjunction with this, the recognition and advancement opportunities given to engineers in the various divisions may also vary significantly.

We found that use of programming languages as well as software development tools is not uniform across divisions as well as across projects. Well-funded projects will have

the resources to purchase specialized tools as well as try approaches that would not be possible in projects that have more limited funds.

Table 2. UCAR /UOP software engineers by program and job classification.

Division	SE-1	SE-2	SE-3	SE-4	Total
COMET	0	2	1	0	3
COSMIC	0	0	1	0	1
DPC	0	2	1	1	4
E&O	0	1	0	0	1
G&A	1	3	3	0	7
GENFUN	0	0	1	0	1
GST	0	2	3	0	5
JOSS	0	4	3	1	8
NSDL	0	1	0	0	1
UNIDAT	0	3	7	4	14
VSP	0	1	0	0	1
Total	1	19	20	6	46

Generally speaking, we found that the large software projects at NCAR are growing in complexity. Often they consist of multi-component, coupled interdisciplinary models requiring large numbers of collaborators having differing domains of expertise. Such software projects will span divisions as well as institutions. With the growth of complexity, there is a stronger need for creating production quality software. In particular there is a need to have a better understanding of project requirements as well as a need for competent software project management including budget overview and resource assignment.

Owing to the diversity discussed above, there are a number of challenges to communication and knowledge sharing across projects and divisions. Also in those divisions having small software engineering staffs, career development issues are often a concern. Additional information with regard to division-specific assessments can be found in the Appendix.

6 Recommendations

6.1 Project Management

6.1.1 *Proposal Review*

Proposed NCAR projects and initiatives should be explicitly reviewed for both their scientific and software engineering merit. Specifically, a panel of senior software engineers with relevant expertise should be added as reviewers of those new proposals with a strong software engineering component. These proposal reviews would help ensure adequate software engineering resources are budgeted, that the proper technologies are employed or developed, and that medium and long-term infrastructural impacts of new projects are identified well in advance.

6.1.2 *Project Planning*

All internally-funded projects with three or more FTE software developers should be required, at their inception, to provide their NCAR Laboratory or UCAR Program management with a Software Engineering Project Plan.

This software engineering plan should specify at least three items: a work breakdown structure, a project management structure, and the software process to be used in the project.

This project plan must also be published on the project's web page.

External projects are also requested, at the discretion of their sponsor, to provide this material to NCAR management and publish it on their site.

6.2 Professional Development

To support career development and rapidly changing computer technology, all software engineering staff should set aside at least one week for professional development each year. Each software engineering employee and his or her supervisor should agree upon the relevant professional development based on the employees' objectives for the year as well as long-term career goals. The 2004 staff performance appraisals should begin including the development objectives and metrics for tracking training should be developed.

The UCAR Software Engineering job matrix should be updated to reflect current software engineering best practices and processes. The update should recognize the different skill sets needed by large and small teams working on HPC and non-HPC systems. Continuing professional education should be emphasized.

6.2.1 Human Resources Professional Development Courses

Software engineering curricula should be established by UCAR Human Resources in collaboration with the UCAR Software Engineering Assembly. These curricula should be tightly aligned with skills and competencies required at the different levels of the updated UCAR Software Engineering job matrix.

The curricula will consist of a sequence of courses hosted by UCAR Human Resources to provide coordinated career development for UCAR software engineers. Curricula courses would be offered in a 2-year rotation. Advancement to a higher SE level would require that the appropriate courses for the advancement level (or their equivalent) had been taken and passed within the last 2 years.

For example, some basic areas might be:

- SE-I Basic software engineering proficiency
 - Software engineering best practices
 - Software engineering standards
 - Writing documentation
- SE-II Advanced software engineering proficiency
 - Software testing and quality assurance
 - Software requirements and design
 - Data management best practices
 - Implementing code reviews
- SE-III Technical lead proficiency

- Applying lightweight or formal software processes
- Software project management
- Software engineering mentoring
- SE-IV Project lead or technical expert skills
 - Some subset of the leadership academy (see link below)

Budget: The committee has found that an instructor-led course costs in the region of \$1500-\$2000 per day per course. This works out nominally to \$150-\$200 per day per attendee. The above curriculum, once up and running, probably requires engineers to attend 5 days of this type of training per year. Thus, an additional investment of \$1000 per software engineer per year would allow each software engineer to advance professionally within this framework.

6.3 Software Engineer Mentoring

6.3.1 Summary

A software engineer mentoring program should be started, focusing particularly on new software engineers who are in groups or divisions that have few senior software engineers. The software mentoring program would be open to and would benefit new scientists, project scientists and post docs; i.e., anyone who spends a large fraction of their time in software development.

6.3.2 Discussion

Some software engineers in the organization are professionally isolated; they have few or no immediate peers doing similar work. Hence, for these engineers, there is little opportunity for professional growth by exposure to more skilled or more experienced coworkers. Similarly, there are engineers who are isolated not by lack of exposure to peers, but by lack of exposure to a broad range of software engineering realms. These engineers may miss opportunities for growth because of the focused nature of their work. In both of these cases, NCAR as a whole certainly has enough depth and breadth in software engineering to provide needed support for those who are isolated. What is lacking is a mechanism to bring together peers from a broader range of the organization. A formal mentoring program would address these issues, assuring that all software engineers would receive exposure to more experienced coworkers and be able to learn from their experience.

A mentoring program would focus on new engineers, especially those with few immediate peers. In addition the program should be open to all members of the organization who do software related work, even if they do not carry the formal title of

"software engineer". The program would have to address the following significant points:

- identifying potential mentors
- how to assign mentors to new staff
- mentor training
- recognition of mentors through performance evaluations

The mentoring program should be implemented through Human Resources, with their input on how best to proceed, as well as consideration of similar programs in other institutions.

6.4 Knowledge Management

6.4.1 Finding

We have found many missed opportunities for communication between NCAR software engineers. Awareness of the technologies employed elsewhere in NCAR is low, there are no mechanisms for software engineers to be aware of projects outside of their own, and although some attempts have been made to create skills databases and mentoring programs, awareness and perceived value of these efforts in the community has been low and their success modest at best. Development tool and software process decisions appear to be based on hearsay rather than credible reviews of software project technologies and skill with these tools and techniques are scattered. These problems stem in no small part from the observation that acquiring, evaluating, and learning software methods and tools is an expensive proposition and can easily seem overwhelming to the individual, and that potential sharing of expertise is not currently taking place.

6.4.2 Recommendation

A Web-based knowledge management system should be created to support software engineering efforts at NCAR. This Software Engineering Knowledge Management (SEKM) system will be headed by a coordinator. The responsibilities of the coordinator will be to promote the dissemination of knowledge about software engineering projects and tools and to assume a position of advocacy for software engineering issues within the organization. Access to the SEKM system will be limited to NCAR employees. The coordinator will be responsible for encouraging engineers to contribute content to the knowledge management system and organize events promoting software engineer communication and community building throughout the organization.

The proposed knowledge management system should at a minimum contain:

- best practice recommendations for software process
- profiles / home pages / links to software engineers
- a database of NCAR software projects
- evaluations of software engineering tools
- announcements of in-house and external SE-related events
- access to SE mailing lists
- software process documents
- style guides, and
- a search engine and site map.

6.4.3 *Rationale*

Software engineering pursuits at NCAR are diverse, and the organization's software engineering expertise is distributed throughout the organization. Many people engaged in software development activities are not classified as software engineers, further reducing the awareness of what peers may be pursuing. Few formal opportunities for collaboration exist and the organizational culture does not encourage software engineer collaboration or community building outside of the projects on which they are currently working. A knowledge management system would provide a technical tool for capturing and synthesizing the software engineering practice within the organization and creating an awareness of what other software engineers are pursuing.

Although a centralized information source about software projects and software processes and tools in use throughout NCAR may be valuable, it is insufficient to create a repository and mandate contribution of resources. Instead, a vibrant knowledge management system can only be created to the extent to which it serves the needs of a community of users. Contribution is the formalization of existing practice, but a desire to contribute and a realization that the resource may contain valuable content relies on a community with awareness of and trust in the knowledge management system.

Unlike other community building initiatives within the organization, such as the Early Career Scientist program, the current software engineering culture does not give career advancement credit for contributions to or participation in the engineering community. Large digital libraries (like DLESE) and knowledge management systems often have a dedicated staff that ensures the contents of the knowledge management are coherent and relevant. For this reason, we feel that it is essential that a SEKM coordinator be a full time position. We have also received informal feedback from the software engineering community, interviews, and town meetings that attribute the failure of other community and knowledge-sharing efforts to an over-reliance on volunteer

contributions and part-time commitments. Useful content that affects genuine cultural change requires a full time advocate.

The potential benefits of a knowledge management system are significant. Capturing experience of developers and awareness of other projects can lead to sharing insights into successful software engineering practice. Collaborative tools can help coordinate large distributed teams. There is little awareness of the number, scope, and goals of the large number of software projects within NCAR, and a knowledge management system could help organize this information.

6.4.4 Implementation

The proposed knowledge management system would require the hiring of a SEKM coordinator, the creation and population of the knowledge management system and the formulation of an oversight panel to evaluate the progress of the SEKM system.

6.4.5 SEKM Coordinator

The SEKM coordinator is responsible for the maintenance and continuous refinement of the knowledge management system. The KM software should support direct contribution by any NCAR software engineer, and the SEKM coordinator would not be solely responsible for generating content for the site. The coordinator would take an active role in soliciting content contributions from relevant members of the organization. Continuous contact with software developers and an active organization of the knowledge management system are essential activities for the SEKM coordinator. The SEKM coordinator would also be responsible for organizing community building events (from informal software engineering gatherings to workshops) to help create community cohesion and identify resources to populate the knowledge management system.

6.4.6 SEKM Milestones

The development of the KM system will involve both technical and community building activities. The technical development of SEKM will require a series of iterative developments of prototypes with rapid feedback from the community. Initially the KM system may be a static Web site and would grow through cycles of information gathering, presentation, and reorganization. The SEKM Coordinator will be responsible for researching knowledge management systems and implementing possible technical solutions.

Technical milestones for the first year would be:

- The creation of a Web site containing a database of software projects within NCAR including project names, descriptions, participants, code architects, and software tools and processes employed

- A mechanism for software engineers to contribute to the Web site
- The integration of the Web site with existing sources of information (such as SE mailing lists) and the possible creation of new mailing lists
- Research and prototype deployment of a knowledge management system

Community building milestones for the first year would include:

- Staffing an oversight panel of software engineers for SEKM
- Holding a series of town meetings with software engineers across the institution to introduce the SEKM concept and the SEKM Coordinator
- Organizing appropriate kick-off forums and gatherings for building cohesion and communication across the software engineering community

6.4.7 *SEKM Oversight*

We recommend that a small panel of software engineers be created to provide oversight of the SEKM project. This panel could be the Executive Council of USEA, if that recommendation is approved (see Section 6.8 for details regarding USEA). This panel will meet at appropriate intervals to monitor progress, provide feedback, and guide the future directions of the SEKM system. The SEKM coordinator will prepare an annual report to the panel about the status and use of the KM system. This report will be based on three sources of information:

- Statistics about library use (number of hits, number of contributions by each division, etc)
- An NCAR-wide survey of software engineers, asking about KM use, limitations, and strengths
- Interviews with software engineering representatives from each division

The SEKM annual report will be presented to the oversight panel and made available for public comment. The panel duties will be to produce, after deliberation, a written response with recommendations, which will be forwarded to the SEKM coordinator and the NCAR directorate.

Throughout the life of the KM system, the system will provide feedback technology including, but not limited to, a mechanism for reporting bugs in the KM system, requesting feature enhancements, content alterations, and new content. This feedback will be captured and available for inspection.

6.4.8 *SEKM Resource Requirements*

It is our estimate that the resources required to implement this system should be modest. In order to encourage strong applicants for the demanding position of

coordinator, we recommend that the NCAR directorate commit to funding it as a two-year renewable position. The coordinator should be an SE-III with a credible background in software engineering and a strong professional commitment to and interest in the advancement for software engineering practices. We estimate this position to cost \$180,000 with salary and benefits. Appropriate funds for technical resources should be marshaled, including funding for provisioning the required computer and data storage systems, software, and technical support services. This is estimated at \$20,000/year.

6.5 Software Process / Best Practices

6.5.1 Introduction

In the field of software engineering, there is increasing interest in general methods to assess and improve software development processes, in order to improve the predictable delivery of quality software within time and budget constraints. Researchers and practitioners have integrated key software development practices into several general methodologies and models for creating or evolving software systems. Some of the more well-known software process models include ISO 9000, the Software Engineering Institute's Capability Maturity Model (CMM), the Rational Unified Process, and Agile Software Development (including Extreme Programming).

These and other software process improvement frameworks aim to:

- reduce development cost or schedule
- improve the quality, reliability, usefulness, maintainability, adaptability, or reusability of software
- improve the predictability of costs and schedules
- support better management of software development and maintenance

Heavyweight methodologies, such as CMM and ISO-9000, are characterized by formally managed processes and plans, use of methodology-specific tools, comprehensive documentation of requirements and process, and gradually improving process maturity through assessment and feedback. At the other end of the spectrum, lightweight software processes such as Agile Software Development are characterized by developing "emergent requirements" along with software from user interactions, incomplete but working software at all development stages, and responding to change rather than following a plan.

Heavyweight processes address the needs of large software development organizations with an artifact-rich, team-based process that involves discipline, detailed measurements of project progress, cost, and return on investment. Agile methods address the needs of projects with shorter development schedules or less formal

requirements in smaller organizations, focusing on the creativity of the individual developer and offering projects the ability to continuously deliver software and to quickly respond to changing user needs and requirements. In between these two extremes are other methodologies that contain various elements of both.

Regardless of which software process is chosen, there are certain invariants that apply to successful software project outcomes: for example, it is difficult to overestimate the importance of obtaining an adequate understanding of requirements or of competent project management. Collecting and understanding requirements is crucial to the communication process between scientists and software engineers and requires project staff with a background in both arenas. Competent project management is obtained only by identifying and training software project managers in the skills necessary to hire and coordinate staff, manage budgets, collect requirements, oversee software design and development, and manage customer relationships. Following a formal software process is only one component of this complex task.

6.5.2 Finding

NCAR/UCAR employs hundreds of software engineers, computer scientists and scientific staff who are actively involved in software development. In addition to the staff involved in software development, NCAR/UCAR employs system's administrators, webmasters, database administrators and so on. Software development here at NCAR is performed within the context of numerous and diverse software projects. The more prominent projects at the institution are the large simulation and modeling projects (WRF, CCSM, and ESMF) that involve a significant number of the staff here as well as collaborators from other institutions around the world. A growing part of the software developed at NCAR/UCAR is not related to high performance simulation, but instead to data and metadata access, intelligent system development, field program support, data visualization, data mining, Internet clients and servers, Web services, graphical user interfaces, instruments, embedded systems, protocols, middleware, and infrastructure. Many of the related projects are small in size involving 1-3 developers.

6.5.3 How is software process implemented at UCAR?

Software process at UCAR is generally implemented on a per-project basis as opposed to a per-division or institutional basis. The type of software process employed is often based on a number of factors:

- Funding agency and their requirements/expected deliverables
- Type of project such as large simulation effort, field program, research effort, operational system, infrastructure development, etc.
- Number of developers involved (small 1-3 people, medium 3-10, large 10+)

- Number of lines of developed code
- Project lifetime
- Experience of the software project lead and the development staff

Owing to the wide variety of software engineering expertise here at UCAR, there is often uneven application of software process in software development projects.

6.5.4 A basic taxonomy of software projects at UCAR

The projects at NCAR that involve software engineering can very roughly be classified into three levels:

Level 1. Small research and development effort.

Such an effort would have one or more of the following characteristics:

- No software deliverables. Project goal would be a research paper or report.
- Project staff would generally be small in size consisting of one software engineer and one or more scientists.
- Software tools such as Matlab, R, IDL, etc. would often have significant focus.
- Software developed would be "prototypical" in nature.
- Minimal software process.

Level 2. Research system.

Such a system would have one or more of the following characteristics:

- Informal software releases or sharing of code outside the project.
- A user base larger than the project team.
- Software deadlines.
- Increased complexity over Level 1.
- Small development team not an individual.
- Moderate software process.

Level 3. Delivered system / large project.

Such a system would have one or more of the following characteristics:

- Software releases.
- Documentation and training.

- Requirements analysis and testing against requirements.
- Increased complexity over Level 2.
- Potentially large development team with collaborators outside the institution.
- More formal software process.
- Coding standards.
- Large user community.
- User support.

6.5.5 *Software process elements in use at UCAR*

As mentioned above, software process and software process elements are chosen here at NCAR on a per-project basis and are generally up to the discretion of the engineering lead and team members. The choice of software process elements is typically influenced by the level of the project. Here is a list showing a rough correspondence:

Level 1

- Software version control. Typically Concurrent Version Systems (CVS) is used.
- Focus is on code reuse as opposed to new development

Level 2

All elements of Level 1 plus

- Designated software project lead
- Organized task assignment and scheduling
- Design documents
- Build/Installation scripts
- Unit testing
- Bug tracking
- Code walkthroughs

Level 3

All elements of Level 2 plus

- Software team organization
- Risk assessment and mitigation
- Requirements management

- User documents
- User training
- Code freezes
- Nightly builds
- Design reviews
- Code reviews
- Ongoing project tracking and assessment
- Organized user support

6.5.6 *Recommendation*

Given the diversity of software projects at UCAR with the wide variety of complexity, size, lifespan, language, number of users and requirements for reliability, security, performance, and innovation, one lightweight process cannot fit all software development projects. Instead, we recommend that the institution commits to develop and maintain a set of best practices for each kind of software development, and encourages individual projects and developers to acquire the necessary training and select and adapt those best practices that are appropriate for their software development.

6.5.7 *Rationale*

Appropriate application of software process and project management principles is a key ingredient in producing quality software within time and budget constraints. Such application requires ongoing attention by management and project staff. Promoting the compilation and usage of best practices at UCAR will serve to support ongoing software process improvement.

6.6 Application Group for Sun, Weather and Climate Laboratory

We recommend that a group comprising computer scientists and software engineers specializing in geophysical applications be instituted and managed as a standing, cross-divisional engineering resource for projects within entities at NCAR whose mission is not primarily software engineering; in particular, the divisions comprising the new Sun, Weather, and Climate Laboratory. Application-specialist CS/SE employees within the proposed group would attach to and become integral members of project teams; on completion they would be reassigned to other projects. Assignment to projects would be for the duration of the need as determined by the progression of the project's development through phases. A reasonable span for involvement in a project would range from one to five years, but this should be decided based on the needs of the project and the desires of the engineer. It is not inconceivable that an engineer in this group could be assigned to a project indefinitely. Employees within this group would

maintain close ties with peer computer science and engineering professionals within the Computational and Information Systems Laboratory, and joint appointments may be appropriate between SWCL and CISL. Thus, employees within the proposed group would also serve important liaison roles between the scientific and engineering communities at NCAR. The group would also help oversee contract software engineering personnel brought in to augment CS/SE pool resources as needed.

6.6.1 Scope

The group is primarily intended to provide software engineering application support to divisions whose mission is primarily scientific. Under the current realignment plan, these are primarily within SWCL, though a wider scope might be appropriate.

6.6.2 Benefits

Formation of such a group would ensure projects access to adequate dedicated computer science and software engineering expertise while creating a persistent reservoir of computer science and software engineering talent and experience that is not lost to the institution as individual projects transition through their life cycles.

Engineers in SWCL would have a home within the institution. Training and mentoring of engineering staff as well as management of software engineering resources across projects would become easier and more consistent; the problem of lone or small groups of engineers working in isolation in some science divisions would be alleviated. The group would foster increased interaction, cooperation, and coordination between the projects and, also importantly, with the scientific computing division.

6.6.3 Costs

A conservative implementation scenario would involve the group being part of the new SWCL and incorporating existing staff from engineering teams already working in the scientific divisions. In this case, the incremental staff cost is for one half of a senior level software engineer (\$90,000, including benefits and overhead) to serve as the group leader, reporting to the SWCL director or a deputy, and for a reasonable fraction of an administrative person. The software engineering effort will be supported by the projects that use it. Base laboratory support would be required to support the manager and administrator.

6.7 Requirements-Based Choice of Computer Language

We recommend that decisions on computer language and other implementation technologies be made after careful consideration of a totality of software engineering project requirements, including:

- life-cycle and maintenance costs

- user base and mode of application (e.g., rapid-prototyping versus production use)
- availability of skilled personnel knowledgeable in the language
- interoperability with other languages
- libraries, and component models
- states of the art, stable market-supported software infrastructure
- general trends in software engineering development and language usage

6.8 UCAR Software Engineering Assembly

6.8.1 USEA Mission Statement

The UCAR Software Engineering Assembly (USEA) exists to foster a sense of community for software engineering professionals within UCAR, to facilitate effective participation of software engineers through cross-cutting interactions in UCAR's science mission, to enhance communication with management in matters of concern to software engineers, and to engage UCAR software engineers as full partners in setting priorities for the design, development and maintenance of the software infrastructure needed to realize UCAR's mission.

6.8.2 Membership

All UCAR staff working in the software engineering profession, regardless of their formal UCAR job classification, are invited to participate. Individuals with a strong interest in the topic and desire to contribute to the mission of the USEA are likewise encouraged to join.

6.8.3 Organization

The organization of the USEA is likewise modeled in most respects on the NCAR Science Assembly. An important difference is that the Chairperson of the USEA Executive Committee would not be a regular sitting member of the NCAR Executive and Director's Committee. Unlike the NSA, USEA Executive Committee members would serve at the pleasure of the constituencies they represent, and the committee chair would serve for just one year.

- Executive Committee - the USEA Executive Committee serves as the interface between UCAR and NCAR management and the software engineering community. It would have the power to form and dissolve standing and ad hoc committees and would be able to change the rules of governance with the approval of UCAR management. Its membership would consist of the SEKM Coordinator plus one representative and one alternate appointed from each

division, institute, or program in UCAR. The USEA EC would meet monthly or when requested by UCAR management.

- USEA Executive Committee Chair – chosen from the USEA Executive Committee members annually. The USEA Chair’s duties include:
 - Representing the USEA, when requested, at the NCAR Executive Committee and Director’s Committee meeting.
 - Preparing the USEA annual report.
 - Working with the SEKM coordinator to generate ideas for USEA sponsored software engineering discussions, workshops and events.
 - Serving as SEKM Coordinator – the coordinator of the Software Engineering Knowledge Management (SEKM) system.

6.8.4 Executive Committee Responsibilities

The USEA EC would be responsible for implementing the following recommendations:

- Review new proposals for software engineering content (See [Proposal Review](#), Section 6.1.1.)
- Review new proposals for software engineering initiatives (See [Proposal Review](#), Section 6.1.1.)
- Coordinate the software engineering seminar series.
- Serve as an advisory panel for SEKM).
- Organize one workshop or “special event” per year to advance software engineering at UCAR.

6.8.5 Seminar Series

USEA would establish a seminar series to feature topics of interest to the community of scientists, computer scientists, and software engineers at NCAR involved in the application of computer simulation, data management and exploration, and other applications of computing in the geosciences. The seminar series, intended to foster increased cross-disciplinary knowledge and cooperation between the scientific and engineering disciplines at NCAR, should feature a mix of presentations covering efforts in progress, success stories, technological trends and other relevant topics from science/engineering teams within the laboratory as well as invited experts and groups from outside institutions. A small program committee would be formed within USEA and would be given appropriate support for administrative, conference, and visitor expenses.

Resource Requirements

The resource requirements to host the seminar series: \$25,000 plus a like sum for funding the annual software engineering “special event”.

7 Overall Cost Estimate

- \$250,000 investment in software engineer professional development funds (\$1000/employee)
- \$90,000 for 1/2 FTE to manage the SWCL Application Group
- \$25,000 for the USEA seminar series budget
- \$180,000 for a full time SEKM coordinator
- \$10,000 for SEKM materials and supplies

Total: \$555,000/year investment in software engineering.

8 Acknowledgements

The Committee would like to thank Bill Preeg for his timely ideas that kept the committee moving forward and his perceptive observations about NCAR from a business perspective. We would also like to thank CSS section administrative assistant Jennifer DeLaurant for dutifully maintaining the SECOMM website and Eric Scharf for setting-up and maintaining the Swiki site. Finally, the Committee is especially grateful to the software engineering staff of NCAR, too numerous to cite here, for many useful and constructive conversations on the subject of improving software engineering practice at NCAR.

9 Appendix

9.1 Division Assessments

9.1.1 SCD Software Engineering Assessment

In the SCD assessment, we will first discuss the methodology used to obtain the input from SCD software engineering staff and managers, and then we will discuss the findings regarding software engineering issues important to SCD.

Methodology

At the SCD Management Retreat (May 5-7, 2003), SCD staff had a one hour discussion on the activities of the Software Engineering Committee. This discussion was led by Rich Loft, the committee co-chair. Those present included Cliff Jacobs of the NSF, the SCD director Al Kellie, all section heads, most SCD project leads, and many key senior software engineers. This discussion was designed to bring issues related to software engineering to the forefront and identify areas that were felt to be deficiencies or systemic problems. The discussions resulted in many extensive one-on-one interviews with staff both at the retreat and in the subsequent weeks.

Organizational Issues Uncovered at the SCD Retreat

Perhaps the most interesting and difficult problem uncovered in these discussions was the role the organizational structure of SCD itself plays in complicating the professional life of software engineers in SCD. In particular, the increasing emphasis by UCAR on interdisciplinary activities and the growing need for the modernization and integration of SCD computing services has begun to have an impact on the organizational and management structure of SCD. Many new software engineering projects are cross-cutting and call for a matrix managed organization, i.e. with horizontal as well as the traditional vertical management elements. Indeed some projects, such as Earth System Modeling Framework (ESMF) and the Earth System Grid (ESG) extend beyond divisional and even institutional boundaries. If management is increasingly done by projects, who then writes the software engineer's performance appraisal? Who arbitrates when demands are made on an engineer from both directions in the matrix managed organization? Al Kellie noted that a similar 'folder' concept he introduced was largely unsuccessful, but there was general support for revisiting the organization of the division to address these issues. Given these organizational challenges, how will SCD support users in an increasingly complex, Grid-enabled world? For example, SCD's consulting group experiences users working in new ways as a massive support issue. Indeed, many projects such as ESMF and Globus are creating new tools and new software infrastructure but no additional core funds are budgeted for technical support over the long term. The eroding support base for old technologies created by SCD through retirements and attrition is also threatening to increase the workload on the remaining software engineering staff. The prospect of degradation of the support for these aging technologies is a real prospect. Even worse, SCD is expected to support aging technologies created outside the division: for example, at a recent user forum, SCD was requested to teach FORTRAN because universities have dropped training students in this programming language entirely.

Software Engineering Technology Issues

SCD need to continue to identify and deploy industrial strength development, process, and project management tools. This will cost more money to support technological

evaluations, software licenses and software engineering training for professional growth. It was noted that there is little coordination across the division in these activities at the moment. Anecdotal evidence is that free or cheap software for these tasks doesn't scale to large projects with many participants.

Software Engineering Personnel and Management Issues in SCD

A discussion of the personnel and career issues faced by software engineers and their managers resulted in the following observations. SCD is not, primarily, a science division and does not suffer from many of the apparent clashes of culture that occur between software engineers and scientists in other divisions. Nevertheless, it is felt that software engineers are quite often still working in isolation and that a sense of professional community is needed. WEB developers within UCAR have started down this path by forming the Web Engineering Group (WEG) and the Web Advisory Group (WAG). A forum of this kind to present and exchange experiences between software engineers was also proposed. It was noted that similar forums have been tried and tend to die unless championed by an individual and that people see that they get something back.

The need for project management training was repeatedly cited. This is not a software engineering specific issue, but training tailored to engineers working on software projects would be most beneficial.

Other private one-on-one discussions held off-line led to the following general conclusions about career advancement. There is a common feeling expressed within the ranks of SCD engineers that there are few career opportunities for them. This situation was jokingly referred to as the Gilligan's Island effect: i.e. that "no one ever gets off the island". Training to maintain currency is experienced in a very uneven way throughout the division. Some staff take courses and even obtain degrees only to find there are no advancement paths available to them afterwards. Engineers in SCD often perceive themselves to be or are perceived by their immediate supervisors as being "too busy" to take time off to receive training. The initial overhead for training unrelated to the immediate task at hand is felt to be a hard sell to managers. There are poor opportunities for advancement in the management arena for software engineers across the organization as well; SCD's "stove pipe by function" organizational structure has a low turnover in managerial positions, few project management opportunities, and little exchange of personnel between sections. This could change under a new matrix managed organizational structure imposed on SCD by emergent, external technological pressures to modernize services.

9.1.2 Research Applications Division Assessment

The Research Applications Division has approximately 50 software engineers working in a variety of areas including the following:

- Data ingest and data management
- Data analysis and visualization
- Automated system development for hazardous weather
- Model development
- Scientific algorithm development
- GIS development

In spring 2003, RAP performed an assessment with regard to strengths, weaknesses and ways to improve.

Strengths

RAP has a talented group of software engineers and puts special emphasis on hiring quality engineers when new needs are called for. RAP engineers are committed to innovation and work closely with users employing an iterative development style. RAP's management has supported such innovation by giving engineers the necessary independence to work effectively.

RAP has a flexible software process that is instituted on a per-project basis. Typically the software process employed is dependent on the requirements imposed by the funding source. At a minimum, projects will utilize version control (Concurrent Versions System). In addition most projects will employ nightly builds or will have checkout and build scripts.

RAP developers utilize a variety of programming languages and tools. C++, Java, FORTRAN, Python and Perl are the languages that are in predominant use. Tools such as JBuilder, Insure, Purify, TogetherSoft, Bugzilla, etc. are also being used. RAP also makes use of specialized programming environments such as Matlab, R, IDL and more recently GIS software from ESRI.

Weaknesses

Even though RAP's project orientation has many strengths, there are also weaknesses associated with this type of organization. Typically these weaknesses are connected with communication and resource sharing across projects. Occasionally, software engineers will remain on a particular project and will not circulate to other projects in the organization. Also, there is an ongoing challenge to make sure that there is coordination among the projects in terms of software reuse and software development practice.

Generally speaking, RAP software engineering needs to improve in the software testing area. Internal and user documentation are often weak. Software development cost

estimation in proposal development is an area needing improvement. Often in wishing to please a project's sponsor, project scope is increased without necessary changes in resources or schedule. Better attention to requirements management is also an area needing improvement in RAP.

Training and professional development are an ongoing challenge in the general engineering community and are a challenge in RAP as well. There is an ongoing need to ensure that adequate time is allotted for RAP engineers to progress in these areas.

Improvement Areas

One way RAP is dealing with improving communication and sharing across projects has been to institute monthly coordination meetings of engineering project leads. These meetings have been quite helpful in attending to engineering issues across the division. RAP is in the process of creating a number of software process checklists. RAP projects will be encouraged to use these checklists on a regular basis in order to acknowledge the particular type of software engineering process being used on the various projects. Checklists will typically cover items such as design review policy, code review policy, version control policy, bug tracking policy, yearly project post-mortems, etc. The information contained in the checklists will be collected in a database in order to get a better handle on current software engineering practices as well as opportunities for improvement.

9.1.3 Atmospheric Technology Division Assessment

Who Are We?

~14 Software Engineers + many others who write code

Wide range of development:

- embedded code
- in-house applications
- externally used applications/libraries

ATD software development methods range from ad-hoc and seat-of-the-pants to much more formal means with design reviews and structured iterative development. A culture of software engineering, as opposed to just coding, is growing very slowly.

What Works Well?

- quality staff
- lack of formality reduces time needed for smaller projects

- broad range of development means there's lots of knowledge available down the hall
- use of CVS in some projects eases collaborative development

What Works Poorly?

- not much communication among software engineers (although this has recently improved greatly, with regular meetings of most of the SE's in the division)
- many single points of failure; we're very vulnerable if many of our individual developers get hit by a bus
- some projects suffer from not using revision control/ collaborative development/repository tools
- analysis/design steps have often been ignored or minimized

9.1.4 *MMM Assessment*

In the Mesoscale and Microscale Meteorology division, about twenty of the approximately fifty staff members in the division perform some amount of software engineering/programming as a significant part of their activities. There are only 4 full-time employees who are classified as software engineers; the rest of the software practitioners are Associate or Project Scientists. Several MMM codes and their associated analysis software -- MM5, WRF, WRF 3DVAR -- are developed and maintained for use by large external communities. Significant funding for these projects is provided from external sources including the U.S. Department of Defense, the Federal Aviation Administration, the National Oceanic and Atmospheric Administration, and several foreign sponsors, each with its own set of requirements, deliverables, and some form of oversight. MMM also develops and maintains other large codes, such as the Large Eddy Simulation code, that are used internally and by groups of external collaborators; there are also numerous single-researcher or small-group codes within the division. With the exception of the WRF project, which has a Software Architecture, Standards, and Implementation Working Group as part of its project structure, there is no formal organization or management of software engineering effort as a distinct activity in the division. There is no organized approach to training and mentoring for software engineers, and there is little or no formal software process management. Attracting and retaining trained software engineers has been a difficulty.

9.1.5 *ESIG Assessment*

ESIG lacks a formal software engineering infrastructure, although several individuals are engaged in software development. Dynamic Web site development, GIS projects, integration of atmospheric models with societally relevant models, and other software-

intense efforts are currently underway. With increasing pressure to create societally relevant software based knowledge products, and with potential expansion of ESIG's size, ESIG could benefit greatly from establishing a stable software engineering infrastructure. Critical mass does not yet exist, but the opportunity exists to adopt software engineering methodologies before informal software projects become too large and unwieldy. Informal interviews with ESIG staff revealed problems frequently encountered when SE practice is lacking, such as summer student projects created by people without software background that was unmaintainable by the staff once the project was complete. As a small division not focused on software development, ESIG faces two important challenges. First, ESIG software producers need a sense of community so that they can interact with peers. Second, future ESIG software engineering projects are likely to be highly interdisciplinary and collaborative, so software engineers grounded in ESIG but capable of interacting with NCAR's other large initiatives are essential.

9.1.6 Unidata Assessment

Unidata Software Development

- 14 developers
- Infrastructure and applications: NetCDF, LDM, GEMPAK, McIDAS, IDV, udunits, DODS/ OPeNDAP, THREDDS, data decoders
- Over 300,000 lines of Java developed and maintained (IDV, OPeNDAP, THREDDS, NetCDF)
- Standards and interfaces as well as software

What's Working?

- Unidata highly successful developing software, as perceived by community
- Factors in success:
 - Staff
 - Small projects (1-3 person)
 - Developers also provide user support
 - Users of our own software, infrastructure
 - Long funding cycles, stable community
 - Agile processes, incremental, test driven

What's Not Working?

- Insufficient knowledge sharing or capturing

- Wide reviews of APIs, interfaces lacking
- No backups for some areas of expertise

How Can We Improve?

- Process
 - Wider reviews of APIs, interfaces
 - Mentoring backups, to pass the torch
 - XP: pair programming, more frequent small releases, refactoring, ...
- Better Tools
 - Local GForge (remote CVS, bug tracking, forums) Bugzilla, Maven
 - Site licensing of other development tools

9.1.7 HAO Assessment

HAO's main programs research methodology is based in theoretical propositions along with empirical and analytical support, thus the creative artifact usually takes the form of a paper. This methodology for innovative research intrinsically implies limited software engineering funding. With the exception of some NSF-special community programs such as CEDAR or RISE in the field of data management and the Thermospheric General Circulation Models, computational science in HAO is performed directly by the scientist or associated scientist using some special purpose computational software, such as IDL.

There are 3 Software Engineers in HAO dedicating about 1/2 or their time to HAO specific software development. There has been a lot of collaboration in the last few years with other divisions within NCAR but especially with other institutions across the nation.

HAO has a formal software engineering infrastructure. This is mostly due because it is a requirement of the distributed efforts for developing applications in collaborative projects. The same infrastructure is then used for specific HAO projects. As collaborative projects had a limited time frame, HAO finds it difficult to retain its human assets in software engineering, however we have experimented some growth (from 1 to 3) in the last few years.

9.1.8 CGD Software Engineering Assessment

The Climate and Global Dynamics Division (CGD) has 17 software engineers, most of whom are working on the Community Climate System Model (CCSM) project. There are few CGD software engineers who have been professionally trained as software engineers. Much of the experience base is in the area of scientific programming. With

the formation of the CCSM Software Engineering Group, CGD has begun transitioning from the traditional scientist/programmer pairing characterized by low levels of software engineering practice to a much more advanced software engineering environment. Critical practices such as code review boards, code and data gatekeepers, requirements gathering, formal testing are now being used. However, this is just the initial step. In order to reap the long-term gains that come from clean code development by professional software engineers working within an effective, coordinated process, a number of items remain to be addressed. This includes increased training of the CGD SE's in professional software engineering processes and practices, dedicating the necessary time and resources needed to make significant software engineering (as opposed to scientific) improvements to CGD software and implementing a "right-sized" software engineering process "right-sized" to CGD's needs and environment.

Specific CGD software engineering areas to be addressed include:

SE Process

After looking across the spectrum of software processes, from lightweight to formal, it is clear that there is a need to choose the right process for our environment, project, and size. The intent is to support a universally useful process when practical (CVS) in NCAR. Streamlined processes are needed. Implementing an effective software process is difficult, requiring significant training and infrastructure. Our experience is that scientist led projects usually means no SE process will be put in place. Too little planning is done due to the need to constantly respond to fires. We need good examples of success and to publicize successes. We need to add 'demonstrating SE best practices' to the SE job matrix.

SE Role

We need to better define the role of the SE in CGD. Is it to be scientific development, infrastructure, code design, or a customer service organization? SE is a "new religion" Should it be organized bottom up or top down? What should be done? How should it be sold? There is a lack of recognition of the job and a lack of respect. NCAR has an organizational bias towards science and scientists. We need to build a culture of professional SE's at NCAR. SE groups do not have equal status in a division.

SE Work Issues

There is a conflict between refactoring and redesign. We're always jamming new stuff into old code. A clean-sheet is required on occasion, but we never seem to have the time or the management will to make it happen. A coding standard is lacking. Models are very complex. Performance portability is a big problem. We need to have time to do

thoughtful code design and implementation. Codes do not perform particularly well right now due to low priority and not enough time. The machine portability capability is very powerful due to compilers, debuggers, memory management, performance, etc. These are tools for HPC. Code reuse and sharing needs to be improved. Design and review is mostly not being done now.

Management

A closer look at management revealed that priorities, timelines, and resources (money and people) are lacking. Project organization coordination, skills and tools are also lacking. Communication also needs to be improved.

Career Development

An SE promotion is formally based on the job not on ability, while scientists have a career ladder. SE III's are stuck in CGD. There appears to be a serious equity issue across divisions with respect to SE classification. There should be a technical ladder and a management ladder. The SE III and SE IV reclassification a few years ago turned out to be a bad thing in CGD. Mentoring is lacking. The salary difference between SE's and Scientists is a sore spot for some scientists.

Training/Interactions

The Human Resource courses have proven to be of limited value. There is little incentive for SE's to improve their skills. We need to define an SE education ladder and make training a formal part of the annual reviews. We also need a budget to be able to do training for SE's. This training should be SE driven. There is also a need for education of scientists in SE and programming, for example SE-lite classes for scientists, etc.

Communications

Poor coordination and communication with other UCAR SE's has been noted. We need to share technical developments and resources better (F90, tools, emacs, shell, etc.) What are the latest tools and tricks? There's a need for education and communication of these. The role of SCD could be tools, training and consulting, however there is little interaction between SCD and the science projects in the divisions.

10 References

10.1 Bibliography

Fowler, Martin. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

Guzdial, Mark; Realff, Matthew; Ludovice, Pete; Morley, Tom; Kerce, Clayton; Lyons, Eric; Sukel, Katherine. Using a CSCL-Driven Shift in Agency to Undertake Educational Reform. In C. M. Hoadley and J. Roschelle (Eds.), *Proceeding, Tthe Computer Support for Collaborative Learning (CSCL) 1999 Conference* (pp. 211-217). Palo Alto, CA: Stanford University [Available from Lawrence Erlbaum Associates, Mahwah, NJ].

Hunt, Andrew and Thomas, Dave, *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, 1999.

Jones, C. *Patterns of Software Failure and Success*. International Thomson Computer Press, 1996.

McConnell, S. *Software Project Survival Guide*. Microsoft Press, 1997.

Meyer, Bertrand. *Object-oriented Software Construction, Second Edition*. Prentice-Hall, 1997.

Weigers, Karl. *Creating a Software Engineering Culture*. Dorset House Publishing, 1996.

Demarco, Tom and Timothy Lister, *Peopleware: Productive Projects and Teams, 2nd Ed.* Dorset House, 1999.

10.2 Web Sites

Douglass Post. "Expert Opinion: The Coming Crisis in Computational Science."

- <http://www.hpcwire.com/hpcwire/hpcwireWWW/04/0402/107343.html>

Extreme Programming

- <http://www.xprogramming.com/>

Joel Spolsky, The Joel Test: 12 Steps to Better Code.

- <http://www.joelonsoftware.com/articles/fog0000000043.html>

People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods by Richard Turner and Barry Boehm, CrossTalk, *The Journal of Defense Software Engineering*, Dec 2003.

- <http://stsc.hill.af.mil/crosstalk/2003/12/0312Turner.html>

Software Engineering Process Sites - Commercial

- <http://www.processimpact.com/index.shtml>
- <http://www.rspa.com/>

Software Program Managers Network

- <http://www.spmn.com/16CSP.html>