A Proven Approach for More Effective Software Development and Maintenance

Rose Pajerski
NASA Goddard Space Flight Center, Code 552
Beltsville, Maryland

Dana Hall
Science Applications International Corporation
McLean, Virginia

Craig Sinclair
Science Applications International Corporation
McLean, Virginia

Abstract

Modern space flight mission operations and associated ground data systems are increasingly dependent upon reliable, quality software. Critical functions such as command load preparation, health and status monitoring, communications link scheduling and conflict resolution, and transparent gateway protocol conversion are routinely performed by software. Given budget constraints and the ever-increasing capabilities of processor technology, the next generation of control centers and data systems will be even more dependent upon software across all aspects of performance. A key challenge now is to implement improved engineering, management, and assurance processes for the development and maintenance of that software; processes that cost less, yield higher quality products, and that self-correct for continual improvement evolution.

The NASA Goddard Space Flight Center has a unique experience base that can be readily tapped to help solve the software challenge. Over the past eighteen years, the Software Engineering Laboratory within the Code 500 Flight Dynamics Division has evolved a software development and maintenance methodology that accommodates the unique characteristics of an organization while optimizing and continually improving the organization's software capabilities. This methodology relies upon measurement, analysis, and feedback much analogous to that of control loop systems. It is an approach with a time-tested track record proven through repeated applications across a broad range of operational software development and maintenance projects.

This paper describes the software improvement methodology employed by the Software Engineering Laboratory, and how it has been exploited within the Flight Dynamics Division within GSFC Code 500. Examples of specific improvement in the software itself and its processes are presented to illustrate the effectiveness of the methodology. Finally, the initial findings are given when this methodology was applied across the mission operations and ground data systems software domains throughout Code 500.

Introduction

A recent analysis conducted by the NASA Software Engineering Program found that over 30% of the NASA Goddard Space Flight Center (GSFC) Code 500 civil servants and support contractors spend the majority of their time directly involved

in the management, development, maintenance, and/or assurance of software (Reference 1). That represents over 1600 people out of the total of 5000 GSFC Code 500 civil service and support contractor community. Correspondingly, that same analysis found a tremendous investment in developed, operational software throughout the Mission Operations and Data Systems Directorate. Not including common off-the-shelf varieties of shrink-wrapped word processors, spreadsheets, and other typical tools, GSFC Code 500 is responsible today for some 21 million lines of operational code. This represents almost half of the 43 million lines of code currently operational throughout GSFC (Reference 2). Most of that is in one way or another involved in the preparation for, conduct of, or results analysis from spaceflight missions.

Given the importance of software in much of what the Mission Operations and Data Systems does and hopes to do, its not surprising that more attention is being paid to software; the tools and practices by which it is engineered; the management oversight by which it is coordinated and paid for; and the means by which products, tools, and know-how are disseminated and shared. The NASA Software Engineering Laboratory (SEL) and its software improvement methodology is a premier example of an attempt to understand the roles of software within GSFC Code 500 and to identify and promote practices that real experience shows are effective and beneficial.

Software Engineering Laboratory (SEL)

The Software Engineering Laboratory (SEL), located in the Flight Dynamics Division of GSFC Code 500, was developed to study the effectiveness of new software engineering technologies as part of the existing Code 500 software development projects. The SEL is a 300 person organization charged with producing operational flight dynamics software for each GSFC space mission, but it is also an organization that has intentionally and carefully for eighteen years experimented with languages, tools, and techniques to continually improve its software development and maintenance process. Within the SEL, every software project is considered to be an "experiment" where a new software technology is injected, its effectiveness measured, and if it proves useful the new technology is incorporated into the software development processes for the next project. The SEL organization works as a partner with the production organization's software developers to incrementally improve the software and its processes over time.

Figure 1 illustrates the three key components of the SEL environment that are critical to the success of improving an organization's software development process and software products. The first component is the development organization. This component is responsible for the software development of a real missions operations or ground data system application. This organization develops the software and documentation using the processes provided by the analysis organization. The development organization also provides software measurements, project characteristics, and lessons learned to the analysis organization.

The second component is the analysis organization. It uses the software measurements and project data to understand the developers' software and software process characteristics well enough to propose an improvement goal, analyze the effectiveness of the improvement, package the results, and feedback the result to the development organization for use in the current and future development efforts. The analysis component interacts with the development component to extract, examine, and compare the consequences of applying

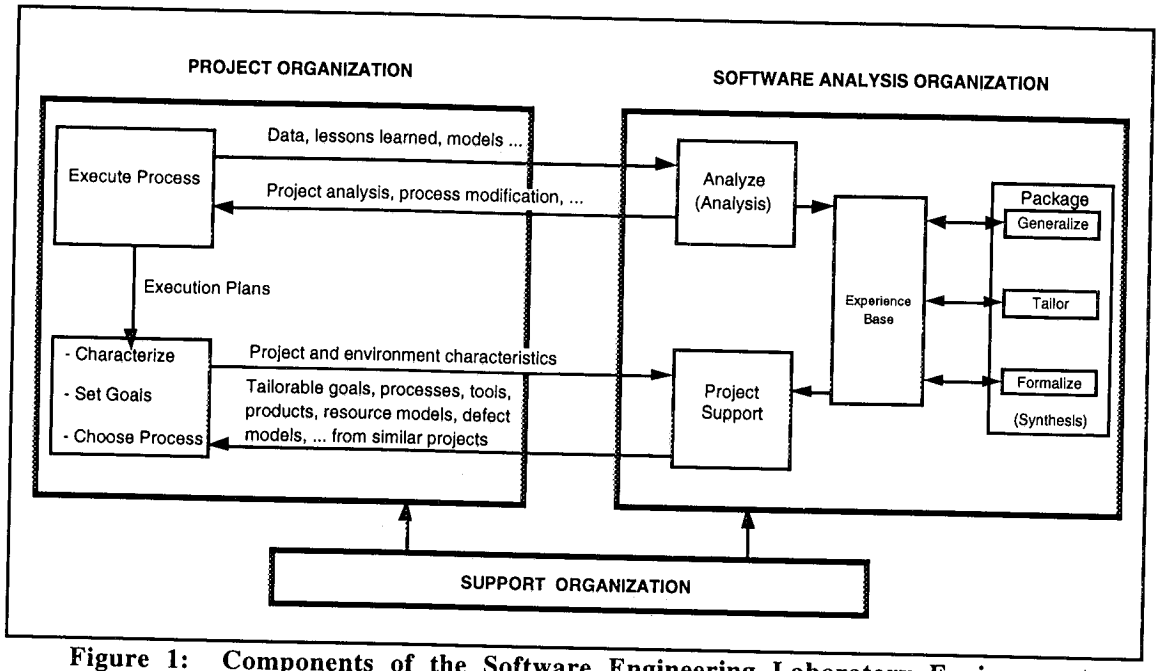

Figure 1: Components of the Software Engineering Laboratory Environment

specific methodologies, standards, and tools.

The third component is the support organization, which archives the information captured from the development and analysis organizations such as software measurement data, process models, training materials, and other documentation.

Over the last 18 years, the SEL has worked with more than 100 production projects where the software was used for mission operations and ground support of GSFC missions. In each of these, SEL analysts quantitatively assessed process changes on the developed software of real projects. These software projects ranged in size from 4 thousand lines of code to a million lines of code (Reference 3).

Software Improvement Methodology

The key distinguishing features of the SEL software improvement methodology are the following:

- Evolutionary not revolutionary
- Continuous
- Incremental
- Bottoms-up rather than top down
- Quantitative software measures
- Software experimenters work with software developers

These attributes are the key to success. Experience in many complex endeavors has shown that true process improvement takes time and commitment. The culture of an established organization must continually absorb and adapt to better ways of accomplishing its business. Experience has repeatedly shown that mandating a standard software engineering process from the top, for example, won't be accepted into an organization's culture. The people who comprise that organization must be part of the evolution of the process, the rules, and the techniques that they find work best for them in their particular environment.

The GSFC SEL software improvement strategy focuses around the simple three layer paradigm shown in Figure 2. This model recognizes that in-depth understanding must precede any attempt

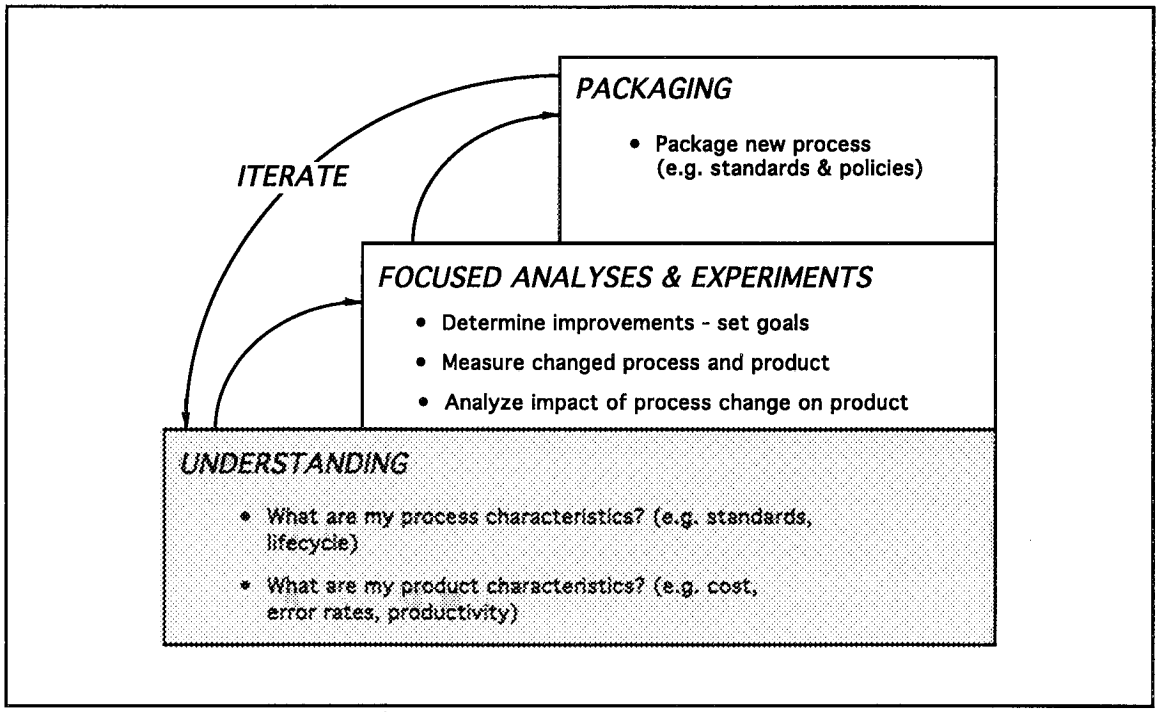

Figure 2: SEL Software Process Improvement Model

to improve. Detailed insight about the functions an organization performs, the dynamics of their interactions, the quality of their products, and the processes and tools they apply forms the basis for the second "layer" of the improvement model. This second layer benefits from the ongoing understanding activity to define focused, incremental improvement experiments. The term "experiment" is important, because change must be planned, instrumented, and compared. Many experiments may not prove helpful or at least not be beneficial in the ways or to the extent originally conceived. Further, the success of each incremental candidate improvement requires people "buy-in" which can only be gained through careful explanation and training, application, and results analysis. As stated above, true improvement takes time and is, by definition, bottoms-up. As improvements are shown to be helpful, they are packaged appropriately for ongoing use by the organization (the top layer shown in the figure). Usual examples of packaging are well-written user guidebooks and training materials. The state of the organization's business practice is thus altered. The packaged processes, tools, training, and guidance become that organization's software policies and standards. And those are effective policies and standards because they reflect what the organization really does (Reference 4).

Software Improvement Results

As an example of the application of collecting and analyzing of software error statistics at GSFC, the SEL collected data to determine the impact of the cleanroom approach on software error rates. Figure 3a shows the error rates of the baseline approach and two small (20-40 KSLOCs) development projects (Reference 5). Each time the cleanroom approach was used, the error rates decreased showing that the cleanroom approach had a positive effect on error detection rates and possibly should be adopted as part of the baselined development process for Code 552.

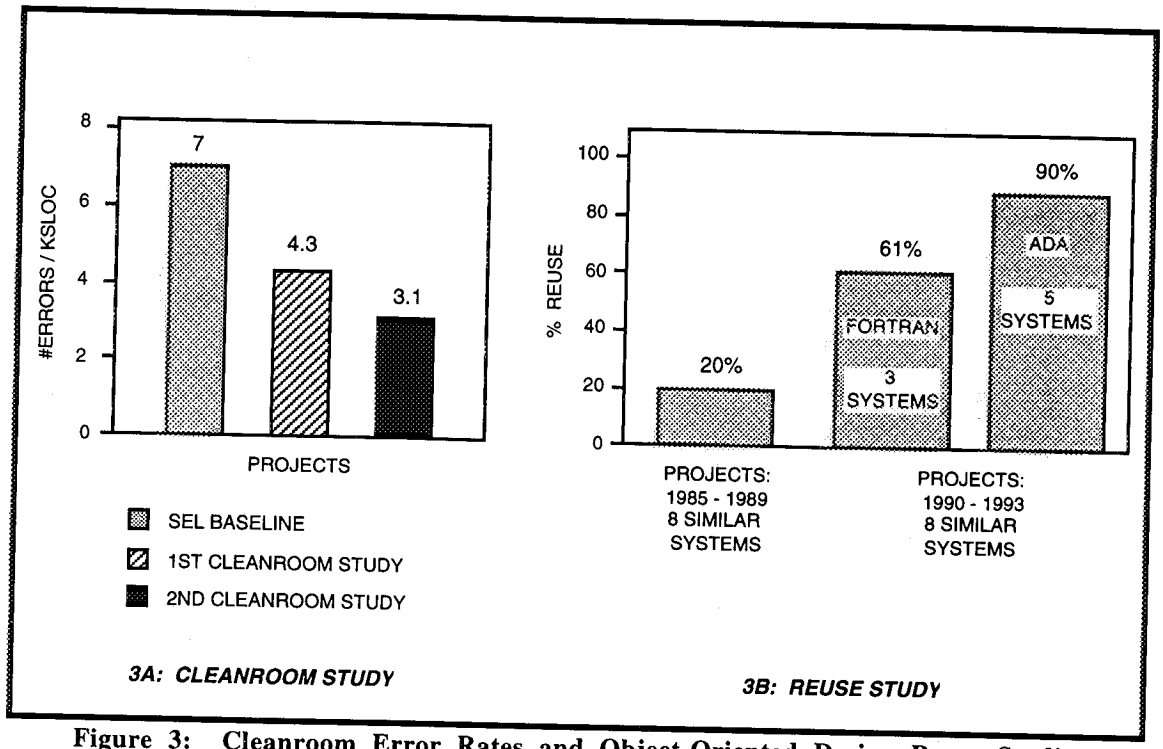

Figure 3: Cleanroom Error Rates and Object-Oriented Design Reuse Studies

Another effort focused on the use of object-oriented technology with the goal of increasing the reuse levels of software within the Flight Dynamics Division resulted in significant impacts as depicted in Figure 3b. The average level of code reuse increased from approximately 20% in the 1985-1989 time frame to over 61% reuse for FORTRAN projects and 90% reuse for Ada projects during the 1990-1994 period (Reference 3). Such improvements provide evidence of the benefits potentially derived from the application of evolving state-of-the-art software engineering practices such as object-oriented design.

Since the SEL has been monitoring and measuring the progress of GSFC mission operations and ground data software for over a eighteen years, the cumulative effect of the SEL software improvement technique on software error rates was analyzed. The software error rate is defined as the number of errors per thousand lines of code. The analysis shown in Figure 4 that the average software error detection rate decreased from 8 errors per KSLOC to 2 errors per KSLOC, a 75% decrease from 1977 to 1993. This type of information leads to well defined models and relationships of software parameters supporting improved management and control of future projects (Reference 3).

Improvement of the product within Code 552 (flight dynamics software) by changing and measuring results of software process changes on real projects has produced real gains in error rates, reuse, and productivity over the last 5 years. For a set of similar GSFC projects using this methodology since the late 1980s, the error rates decreased 75% (from 4 errors/KSLOC to 1 error/KSLOC), reuse increase from 20% reuse to 75% reuse, and the productivity increased 75% (from 440 staff-months to 110 staff-months) for an equivalent amount of software.

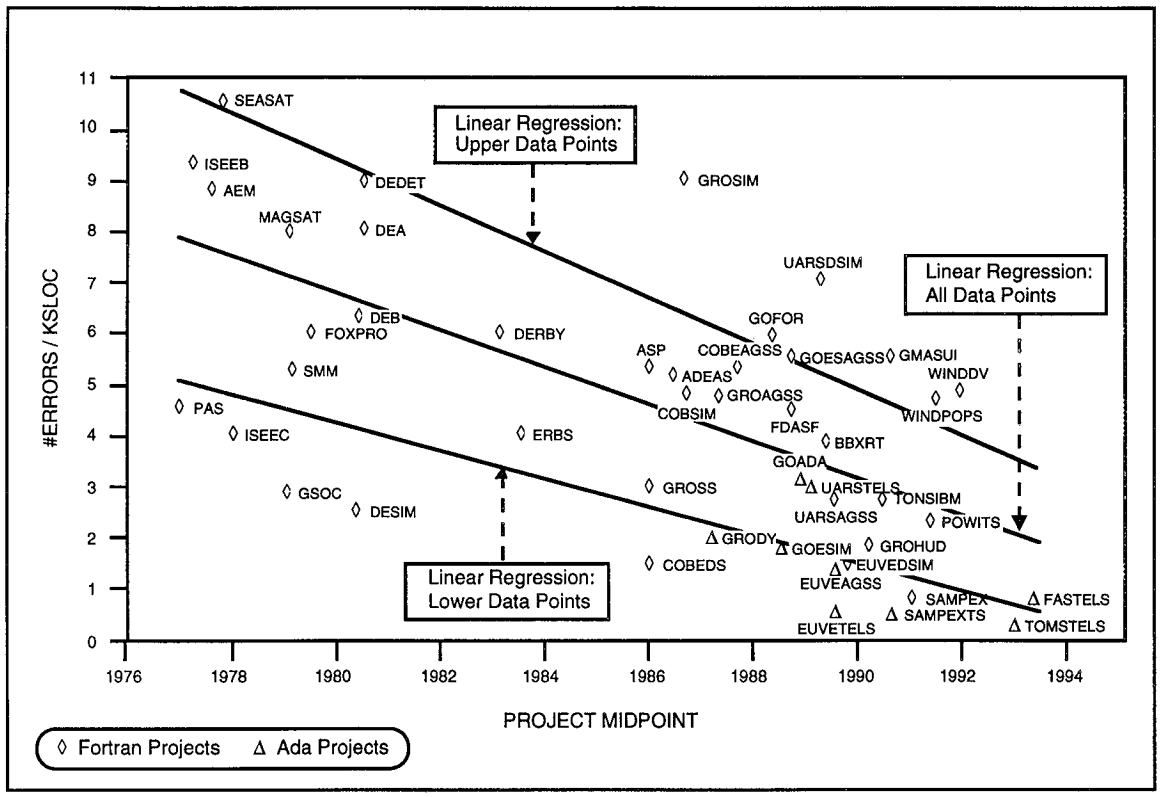

Figure 4: Software Error Detection Rate Over 16 Years for SEL Projects

Software Improvement Within GSFC Code 500

The basic methodology used by the SEL for flight dynamics software was applied to the Mission Operations and Data Systems Directorate as a whole. In accordance with the SEL software improvement model, a software "baseline" for Code 500 was established (Reference 1). The data for establishing an understanding of the Code 500 software and its development and maintenance processes was gathered via four integrated approaches: a comprehensive nine page survey, reviews of organizational and project data, informal roundtable discussions, and one-on-one interviews.

We sampled the GSFC Code 500 civil servant and support contractors by selecting respondents from organizations performing the majority of the software work. We then supplemented the insight gained through the surveys by reviewing software policies, standards, staffing data, project plans, and other organizational and project data. After initial review of the data, we conducted roundtable discussions with groups of respondents to refine the data and to obtain suggestions on how software processes might be improved. Finally, we conducted one-on-one interviews with selected managers to increase our confidence that our data was representative of the software work performed by that organization.

The published software baseline included both software product descriptions (amount of software, cost of software, software staffing, software error rates, etc.) and software process descriptions (standards used, metrics use, project volatility, development and test methodology use, training, etc.). This baseline data was then analyzed. Using the basic goals of decreasing cost, increasing productivity, and decreasing

error rates, specific areas for potential software improvement were identified. These areas were researched and a set of recommendations for improvement of the software product and processes within GSFC Code 500 were developed. Three of those recommendations are presented here.

Recommendations

Establishing our understanding baseline for GSFC 500 took time, patience, and careful analysis. We believe our understanding was sufficiently detailed and reasonably correct enough to move smartly into the second thrust of the improvement process; i.e., defining focused incremental improvements and experimenting with those improvements in controlled ways.

Three major areas that promise large near-term payoff for relatively small investments are:

1) Introduction of ongoing, continual software improvement into the culture of the Directorate

2) Establishment of an integrated software training program

3) Implementation of an effective software measurement program

These improvements can be accomplished for relatively little money and in a short time period because significant components of what are needed already exist.

Organizational Software Improvement Infrastructure

In order to implement and sustain any software improvement change across the Directorate, it is necessary to put in place a software improvement infrastructure throughout the organization. Upper management commitment and long term involvement is critical. So is the participation of everyone throughout the organization that has anything to do with software development and maintenance. People must be involved, have influence on, and help shape where their organization is going. Simply assigning another working group or holding an occasional meeting won't accomplish the software improvement goals. Improvement working teams or Software Process Groups must be established at all levels (Directorate, Division, and Branch). Both process improvement and software product improvement need to be emphasized. An initial task might be to develop a Code 500 approach (not necessarily standards) for the development and maintenance of software. With the help of improvement guidance such as the Software Measurement Guidebook (Reference 6), the Software Manager's Guidebook (Reference 7), SEL experience, and the materials from the Software Engineering Institute, this hierarchy of Software Process Groups could identify, define, and implement techniques designed to continually improve Code 500's software capabilities.

Software Training Program

GSFC Code 500 could benefit from an integrated software training program. Our findings indicate that software training tends to be focused on specific "hot" technologies as opposed to overall software process and development of personnel for key software positions. This goal could be accomplished with a bottoms-up approach by allowing project-level experiences to drive the content of the integrated training program. The needed disciplines are at minimum those of software project management, software requirements management, software contractor management, configuration management, quality assurance, and the software engineering life cycle. The courses must be consistent in approach, show the role of software measurement and feedback in the context of each discipline, and be

tightly integrated to the GSFC Code 500 approach to software development.

The curriculum will be most effective if each course has an overview version that is 3 to 4 hours in length and a full duration version (1 to perhaps 3 days depending upon the subject.) The overview version would be taken by everyone involved with software, but not directly responsible for that discipline area. For example, only software project managers and those people training to become such managers would take the full length software project management course. An effective enhancement to the basic training program would be on-line refresher modules accessible from any of the organization's networked workstations.

The basic elements of this training curriculum already exist at CSC, SAIC, and in the SEL at GSFC. This existing courseware and instructors can be tailored and enhanced and could be ready for use without a long delay or large additional investment.

Software Measurement Program

We found that little attention is given to software measurement in most GSFC organizations. Several contracts required metrics to be collected and forwarded to the government, but little or no analysis was being performed and even less in the way of improvement feedback into the actual projects. The consequence was the project and line management and staff had virtually no real insight about critical status indicators such as the number of errors in the delivered code, the amount of time any activity actually took, or how well the documentation matched the design, code, or testing.

Our recommendation is that GSFC Code 500 develop an effective, practical software metrics program to collect, analyze, and provide feedback for the following purposes:

- Continually decrease software cost
- Assist in the management of software projects
- Assure timely delivery of products
- Improve software reliability

Fortunately, practical solutions and experience are readily at hand. The SEL in GSFC Code 552 is one of the few nationally leading organizations that has proven, long term experience in the definition, analysis, and application of software metrics. The SEL-developed NASA Software Measurement Guidebook (Reference 6) will be released shortly. Code 500 should adopt a top level software measurement policy with the local organizations choosing their own specific goals for measurement, picking the minimum set of metrics needed to meet their goals, and performing metric analysis and feedback. The know-how in this Guidebook combined with an integrated training program are key improvement tools that are easily available.

Conclusions

The GSFC Mission Operations and Data Systems Directorate has successfully developed many millions of lines of code for ground systems of numerous spacecraft. Even so, our analysis of the ground and data software systems shows that are areas that could benefit from a sustainable, continuous software improvement program. The Software Engineering Laboratory is an example of this. In the past five years, the SEL saw a 75% increase in productivity and a 75% decrease in software error rates in flight dynamics projects. The SEL software improvement method of working directly with the development projects and using quantitative measures to test new software technologies can be applied throughout the GSFC Code 500 software domains.

The SEL approach of understanding, assessing, and packaging the assessment

results was applied to the Code 500 software domains in general. This study identified three areas in which GSFC Code 500 could enhance the success of their software development and maintenance projects: institute a Directorate-wide software improvement program, develop an integrated software training program, and develop a software measurement program. We believe that GSFC Code 500 has a superb opportunity to leverage the isolated experiences already existent in their organization to adopt a broad, experience-based software improvement program that could indeed be a model for both Government and industry.

References

1. McGarry, F. & Hall, D. (1994, June). *Profile of Software Within Code 500 at the Goddard Space Flight Center.* NASA/Code QE. NASA Software Engineering Report NASA-RPT-001.
2. McGarry, F., Hall, D. & Sinclair, C. (1994, June). *Profile of Software at the Goddard Space Flight Center.* NASA/Code QE. NASA Software Engineering Report NASA-RPT-002.
3. McGarry, F., Jeletic, K. (1993, December). *Process Improvement as an Investment: Measuring its Worth.* Proceedings of the Eighteenth Annual Software Engineering Workshop. SEL Report SEL-93-003.
4. Caldiera, G., et.al. (1993, December). *NASA Software Process Improvement Guidebook.* NASA/HQ/Code QE. NASA Software Engineering Report NASA-RPT-nnn. Draft.
5. Green,S.E., Pajerski, R. (1991, December). *Cleanroom Process Evolution in the SEL.* Proceedings of the Sixteenth Annual Software Engineering Workshop. SEL Report SEL-91-006.
6. Pajerski, R. & Bassman, M. (estimated 1994, July). *Software Measurement Guidebook.* NASA HQ/Code QE. NASA Software Engineering Report NASA-RPT-nnn. Draft.
7. McGarry, F., Waligora, S., Landis, L., et.al. (1990, November). *Manager's Guidebook for Software Development (Revision 1).* NASA/GSFC. Software Engineering Laboratory Report SEL-84-101.