Proceedings of the 1999 Particle Accelerator Conference, New York, 1999

SOFTWARE ENGINEERING PRACTICES FOR CONTROL SYSTEM RELIABILITY*

S. K. Schaffner, K. S. White#
Thomas Jefferson National Accelerator Facility, Newport News, VA

Abstract

This paper will discuss software engineering practices used to improve Control System reliability. We will begin with a brief discussion of the Software Engineering Institute's Capability Maturity Model (CMM) which is a framework for evaluating and improving key practices used to enhance software development and maintenance capabilities. The software engineering processes developed and used by the Controls Group at the Thomas Jefferson National Accelerator Facility (Jefferson Lab), using the Experimental Physics and Industrial Control System (EPICS) for accelerator control, are described. Examples are given of how our procedures have been used to minimize control system downtime and improve reliability. While our examples are primarily drawn from our experience with EPICS, these practices are equally applicable to any control system. Specific issues addressed include resource allocation, developing reliable software lifecycle processes and risk management.

1 INTRODUCTION

Jefferson Lab, a multi-use facility consisting of a Continuous Electron Beam Accelerator simultaneously serving three experimental halls and a Free Electron Laser outfitted with six user labs, is unique in its use of a single control system lab-wide. Both machines, along with their respective experimental areas and cryogenic facilities, use EPICS for their control and monitoring needs. EPICS is the result of a collaboration, which began in the early 1990's, between Los Alamos National Laboratory and the Advanced Photon Source at Argonne National Laboratory. Jefferson Lab is currently the second largest EPICS site in a collaboration that now encompasses over 100 laboratories and universities worldwide.

While EPICS continues to develop to include more capabilities, it is important to note that EPICS is a toolkit, providing development tools and runtime programs, to form the core of a complete control system based on the standard model.1] Known as an attractive alternative to the previously popular approach of custom coding all control system functions at each facility, EPICS embodies software sharing at its best. The functions provided by the collaboration represent the core requirements for most modern control systems: control algorithm development and execution, data monitoring, storage, retrieval, and visualization, and network communication between front and back-end processors. Specific control algorithms, invariably different between sites, and some device drivers, must be implemented by each facility using EPICS development tools. These tools allow programmers to generate a combination of code and configuration files used to direct EPICS runtime programs. Using this approach, EPICS provides a well-tested core for a complete control system that behaves according to site-specific instructions.

Using code generated by other laboratories saves programming effort by eliminating the duplication of effort present when recreating commonly used functions at each site. Additionally, all sites benefit from the experience and expertise gained by others through their use of EPICS. The base code provided to the EPICS community benefits from more development effort and testing than could be achieved by a single site. However, using EPICS does not eliminate the need for software engineering and programming efforts at each site. The EPICS installation itself, specific control programs, and configuration data must be supported. Most often, this support includes software development, maintenance, and upgrade activities. While managing this body of collaborative and site specific software in a way that provides consistent, reliable controls for machine operations can be challenging, it is not substantially different than managing other software projects of similar size and complexity. Such management requires reliable processes suited to the needs of the project and staff.

While Jefferson Lab is operational, there are still many upgrades in progress and planned for the future that require modifications to existing control programs or the development of new code. Additionally, these upgrades must be accomplished during brief scheduled maintenance periods with limited time for testing. These project requirements mean we must be able to quickly install new software, perform tests and restore the operational machine. These needs have driven the development of an appropriate Applications Development Environment and corresponding software management processes.

* Work supported by the U.S. Department of Energy, contract DE-AC05-84ER40150
# Email: karen@jlab.org

2 STANDARD PRACTICES

It is common for software organizations to develop their own internal standards and practices, but this effort can be streamlined by studying current best practices and incorporating this information in a way that is locally appropriate. Standard practices can provide proper guidelines, help prioritize the implementation of new processes, and be used as a checklist to ensure all necessary process areas are covered.

As the software industry grows and matures, much has been studied and written regarding software engineering practices. The most complete and definitive work on this topic, CMM, comes from Carnegie Mellon University's Software Engineering Institute (SEI).[2] This well documented model provides a framework for evaluating and improving key practices used to provide reliable software development and maintenance capabilities. Using a maturity based approach, CMM gives the software engineering community an effective and standardized means for modelling, measuring and defining software development processes. The model defines five levels of maturity, summarized in Table 1, and details the underlying principles and practices needed to improve process maturity and thus produce more robust, reliable, maintainable software systems. CMM recognizes that such improvements require ongoing effort and provides a framework to help organize and prioritize this effort. This model is very useful because the information is easily accessible and applicable regardless of the systems, projects, languages or techniques involved. The model allows software engineers to assess their systems to determine the current maturity level and the needed maturity level based on risk factors. Once this assessment has been done, process improvement can be introduced over a period of time, in parallel with project development. While it is ideal to have software processes defined and in use from the beginning of a project, in reality it seldom works this way. CMM allows processes to be added as the project grows and transitions from the development stage to an operational system.

Table 1:

Level	Characteristics	Focus Areas
1 Initial	Chaotic, few if any processes	None
2 Repeatable	Processes for management	Requirements Management Project Management
3 Defined	Processes for management and engineering	Training Process Definition Risk Management Project Performance
4 Quantified	Mechanisms to measure processes and quality	Quantitative Process Measurement Quantitative Quality Measurement
5 Optimizied	Process optimization	Innovation and Improvement

Most typical software efforts in non-commercial, research oriented environments, like laboratories, begin at Maturity Level 1. Employing even a few of the CMM techniques can ensure that the project does not also end there. Left to their own devices, software engineers usually develop some processes resulting in varying levels of product quality. The challenge is getting all members of a software development group to adopt the same reliable practices. Adopting standard practices, like those described in CMM enables this to occur consistently and at the correct priority. In order to ensure success, it is important that the software development staff develop the actual process definitions. This ensures efficient processes without unnecessary overhead. The role of management is to specify the process goals and support the development and use of the resulting processes. Taking this approach usually results in maximum adherence to the processes and standards since the developers appreciate the benefits of the methods. Additionally, a properly designed and implemented development process will increase the efficiency of programmers by providing tools to execute redundant tasks, thus saving time and ensuring consistency, a key to reliability.

3 PROCESSES AND TOOLS

As with most projects, Jefferson Lab's control system development began as a chaotic effort. Over time, and without the benefit of CMM, the Controls Group naturally implemented processes designed to aid in the software development lifecycle. The group first began looking at the process of requirements management. This arose out of a need to organize the haphazard approach to system specification that produced few if any useful requirements documents. Requirements were received from many different customers, sometimes verbally, some requirements conflicted with others, and all were labelled high priority. Written requirements often did not exist or did not contain the kind of information needed to proceed with a project. To put order to this chaos, we developed a template for software requirements that was used to structure the information received and ensure that all necessary information was acquired. Additionally, for software that was to be used by multiple groups at the lab, meetings were scheduled with representatives from each group to aid in resolving conflicting requirements. Establishing this standard template quickly improved the requirements gathering process and ensured there was a documented baseline for each major subsystem. In order to continue to benefit, the documents must be updated when new features are requested. Although updates can be difficult to sustain, especially with frequent upgrade requests, they pay off during system maintenance and subsequent upgrade work.

Faced with a seemingly endless list of tasks and a fixed number of software engineers, we next focused on

methods for project planning, tracking and resource management. These issues were addressed by the development of a software task database to document requests, assign priorities and resources, and track progress. For larger projects, preliminary meetings are held to help determine the scope and complexity of the software effort to make advance planning easier. The group leader prioritizes requests based on lab-wide goals, customer input, and resource availability, and keeps the database information up to date. Another tool is employed by the software engineers to track changes to software as upgrades occur. This information is integrated into the configuration management system accessible via the World Wide Web and is also used to facilitate software quality assurance.

A number of tools and techniques have been developed to improve software quality and minimize controls system downtime. Software testing can be difficult due to the lack of adequate off-line facilities and limited machine test time. Before any software is installed in operations, a testplan is written documenting the application, software test procedure, and roll back information in case problems arise after beam operations commence. The testplan is reviewed and scheduled by a team leader. In addition to the application specific tests, machine maintenance periods are followed by a comprehensive control system quality assurance procedure. This ensures all controls computers, applications and communications are functioning correctly. The Controls Group also provides 24 hours on-call support for operations. The software on-call person is always available via pager and is trained to either solve problems that arise or to contact the system expert if needed. Even if software quality assurance led to error free operational code, control system on-call support would be needed to handle hardware failures and situations where new operational procedures introduce disruptive resource loading. It is interesting to note that the frequency of off-hours calls has been significantly reduced since the testplan process has been put in place and control system downtime has dropped as well.[3]

Another aspect of our software process involves configuration management. A source code structure has been adopted making it possible to introduce automated tools to create, version, and install operational software. These tools enable software developers to quickly load new code and roll back to previous versions, greatly enhancing reliability by insuring that all process steps have been consistently applied to each application. In addition to reliability improvements, such tools make the developers' job easier by automating lengthy installation steps that were previously typed by hand.

4 CONCLUSION

At Jefferson Lab, the development and use of good software development processes has improved the quality of our software and reduced machine downtime due to control system problems. Because we inherited an unorganized development effort in progress, we have introduced new processes one at a time, phasing in supporting tools as time allowed. Our experience developing these processes and tools has led us to define the following list of characteristics of good software processes:

1. Processes are developed by those who will use them.
2. Processes are supported by management.
3. Processes take into account project and developer needs.
4. Processes make programmers' work easier, more efficient.
5. Processes produce repeatable, consistent results and automate repetitive steps.
6. Processes are documented and publicly available.

This list is useful to help design new processes and evaluate existing processes for possible improvements. We plan to continue developing and improving our processes using SEI guidelines such as the CMM framework. By standardizing our development approach and prioritizing work, we have been able to improve programmer productivity and reduce the level of stress and pressure on the group. The number of operational control system failures has been reduced along with the associated lost time. We have found the time invested in developing good processes has been well worth the effort, and believe that more improvements can be made.

5 REFERENCES

[1] M. E. Thuot, L. R. Dalesio, "Controls System Architecture: The Standard and Non-Standard Models", Proceeding of the 1993 Particle Accelerator Conference", Washington D. C.

[2] Mark C. Paulk, Charles V. Weber, Bill Curtis, "The Capability Maturity Model: Guidelines for improving the Software Process (SEI Series in Software Engineering), Addison-Wesley Publishing Company, June 1995.

[3] K. S. White, H. Areti, O. Garza, "Control System Reliability at Jefferson Lab", ICALEPCS'97 Proceeding, Beijing, China.