LA-UR- 09-03479

Approved for public release;
distribution is unlimited.

Title: The Application of Formal Software Engineering Methods to the Unattended and Remote Monitoring Software Suite at Los Alamos National Laboratory

Author(s): John C. Determan
Joseph F. Longo
Kelly D. Michel

Intended for: Proceedings of 50th INMM Annual Meeting
Tucson, AZ
July 12-16, 2009

Los Alamos
NATIONAL LABORATORY
EST. 1943

The Application of Formal Software Engineering Methods to the Unattended and Remote Monitoring Software Suite at Los Alamos National Laboratory

John C. Determan, Joseph F. Longo, Kelly D. Michel
Los Alamos National Laboratory
P.O. Box 1663, MS-E540, Los Alamos, NM, 87545

ABSTRACT

The Unattended and Remote Monitoring (UNARM) system is a collection of specialized hardware and software used by the International Atomic Energy Agency (IAEA) to institute nuclear safeguards at many nuclear facilities around the world. The hardware consists of detectors, instruments, and networked computers for acquiring various forms of data, including but not limited to radiation data, global position coordinates, camera images, isotopic data, and operator declarations. The software provides two primary functions: the secure and reliable collection of this data from the instruments and the ability to perform an integrated review and analysis of the disparate data sources. Several years ago the team responsible for maintaining the software portion of the UNARM system began the process of formalizing its operations. These formal operations include a configuration management system, a change control board, an issue tracking system, and extensive formal testing, for both functionality and reliability. Functionality is tested with formal test cases chosen to fully represent the data types and methods of analysis that will be commonly encountered. Reliability is tested with iterative, concurrent testing where up to five analyses are executed simultaneously for thousands of cycles. Iterative concurrent testing helps ensure that there are no resource conflicts or leaks when multiple system components are in use simultaneously. The goal of this work is to provide a high quality, reliable product, commensurate with the criticality of the application. Testing results will be presented that demonstrate that this goal has been achieved and the impact of the introduction of a formal software engineering framework to the UNARM product will be presented.

INTRODUCTION

The Unattended and Remote Monitoring (UNARM) system is widely used by the International Atomic Energy Agency (IAEA) to provide nuclear safeguards at many types of nuclear facilities all around the world, as depicted in Figure 1. UNARM is in use at reactors, storage facilities and reprocessing plants, for a total of 75 sites in 14 countries across 4 of the 6 inhabited continents. While radiation data, such as neutron and gamma ray data, are at the core of the UNARM system and of utmost importance at nuclear facilities, UNARM also provides support for other associated data types, such as image data, balanced magnetic switch data, operator declarations and GPS data. Analytical capabilities are also provided for determining such things isotopic ratios and Pu and U masses.

UNARM is usually described as consisting of three logical components: data acquisition, collection and review. UNARM is implemented as a combination of hardware, firmware and software elements, as depicted in Figure 2. The hardware and firmware are largely associated with the data acquisition; for example, radiation detectors such as He3 tubes connected to miniGRAND (Gamma Ray And Neutron Detector) instruments running firmware to acquire the data. Instruments such as the miniGRAND are in turn connected to a computer known as the Collect computer via some specialized networking hardware known as intelligent local nodes (ILONs). The Collect computer

and associated ILONs host a constellation of software and firmware components that collectively function to collect the data from the instruments and safely and securely maintain it. Finally, elements of the collect system software may be used to transfer the data from the Collect computer to another computer, the Review computer, where several data review applications may be used by IAEA inspectors in either "stand-alone" or "integrated" mode to review the data. Integrated mode is best described by an example. When an IAEA inspector sees an event of interest in the radiation data using the Radiation Review application (RAD), that inspector may view images taken at about the same time as the radiation event in the Digital Video Review application (DVR) by simply clicking the mouse. In stand-alone mode, links to other review applications are not enabled.

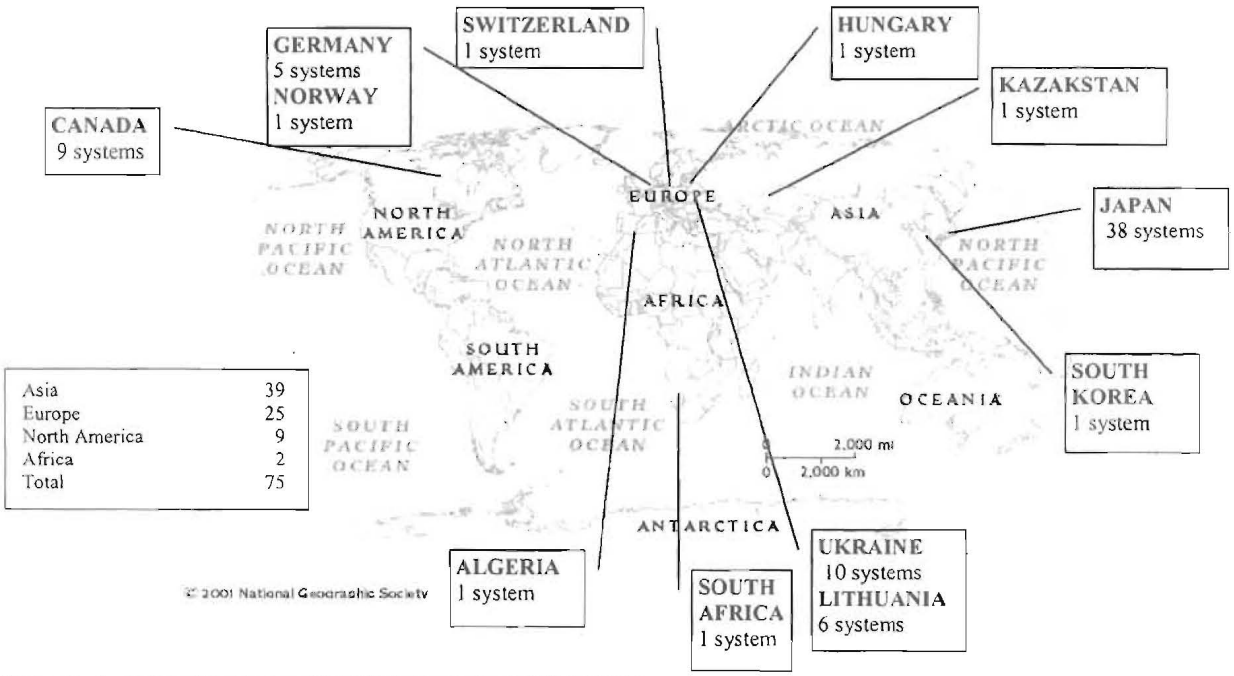

Figure 1. Distribution of IAEA installed UNARM systems.

UNARM development began in the early 1990's, and after one decade had grown to a large collection of hardware, firmware and software components. Around 2002 the awareness was growing that increased formality of operations was necessary to properly manage the development and maintenance of the UNARM system, particularly its many firmware and software elements. Greater formality was also called for because the UNARM system had been adopted by the IAEA and put into use around the world. Since 2002, several key formal software engineering practices have been adopted by the UNARM software development team, and several formal software tests have been conducted on the UNARM software. This paper will describe those software practices and the present the results of several rounds of software testing.

FORMAL PRACTICES

Several key formal software engineering practices have been adopted by the UNARM software development team. The UNARM development team is and always has been lean, so there is not a highly diversified structure of designers, implementers and testers; our typical budgets cannot support that level of specialized effort. Software engineers handle the design, implementation and

maintenance in fluid teams as needed for each new effort, and technicians perform testing to provide a measure of independence. The adopted practices are adapted to this lean environment. These practices include: software configuration management, issue tracking, a software control board, formal testing and managed releases. Together, these formalities allow the UNARM software team to manage its resources, software issues, and delivered products.

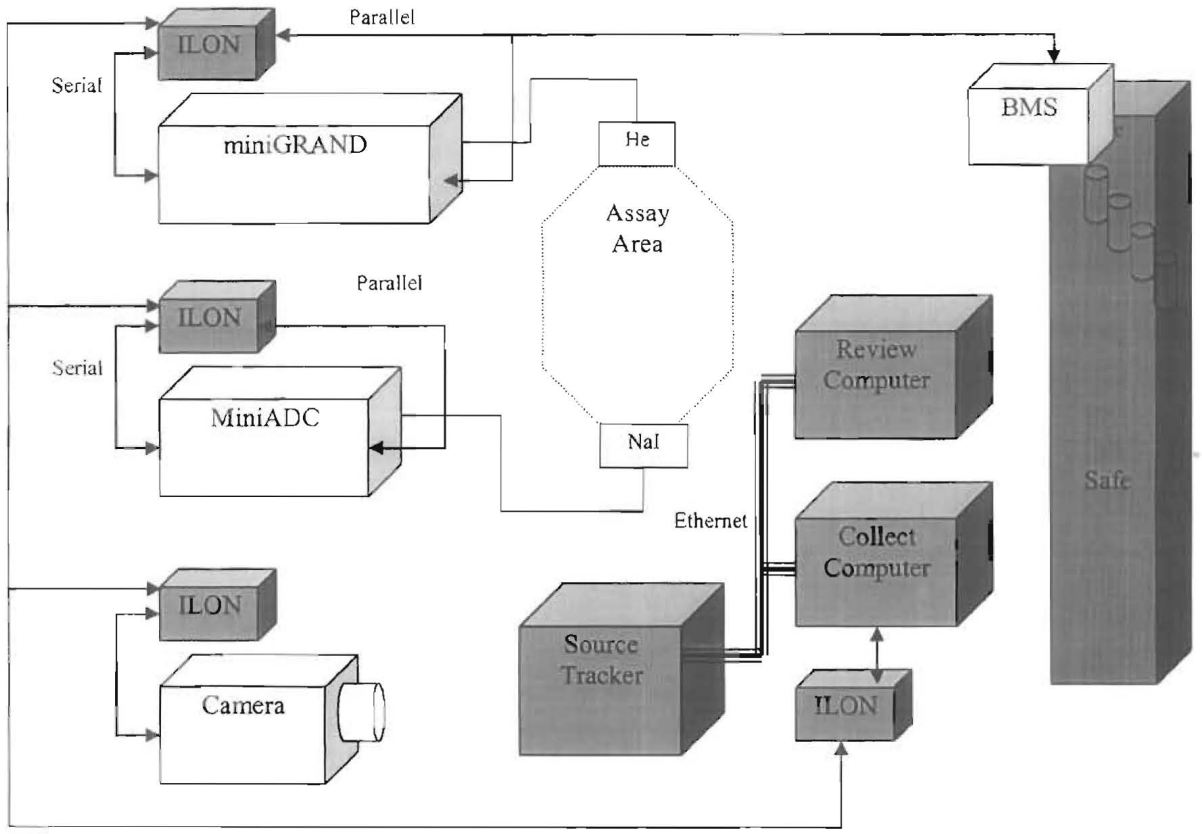

Figure 2. Schematic of demonstration UNARM system maintained at LANL.

Software configuration management was the earliest practice to be adopted. Our configuration management efforts are centered around the Microsoft Visual Source Safe (VSS) application, but also include recommended procedures for various activities, such as source code check-in, software versioning, and others. Proper use of VSS provides many advantages, including backup of all source code, version control, historical queries on code modifications, and ease of project deployment to back-up developers when needed. The associated procedure for source code check-in is particularly useful in making sure that all necessary modified or newly created components have been captured by the check-in. An activity log is part of the check-in process, and this has been especially useful for situations where resource demands have dictated that different developers are involved serially in the support of a single application.

A software control board was the next practice to be adopted, and in turn, through the control board's deliberations, the remaining practices were initiated. UNARM comprises dozens of software and firmware elements, and the ripple effect of changes in one part of the overall system

often have to be managed. For example, one utility, FacilityManager, manages the data about facilities – what instruments and detectors are installed at a facility, where facility-related directories are located on the computer's hard-drives and more. All of the review applications depend on this utility for information about the facility currently being reviewed. A significant change to FacilityManager could well require all dependent software to be recompiled, at the very least, so a high level of coordination amongst the developers is required, and this is one thing that the control board provides. The control board is also a forum for product planning and process improvement discussions. Our current systems for issue tracking and release management evolved under the guidance of the control board.

Issue tracking has always been of importance. Prior to the control board, issue tracking was performed in an ad hoc manner, by individual developers maintaining isolated lists, often in spreadsheet files. The control board recognized very early that a more comprehensive system for issue tracking was vital to product planning and delivery. A first stab at the problem was made by adapting a Microsoft Project Server system already in use for other purposes, to maintain a database of issues for the entire UNARM suite of software components. This database was a huge step forward, but limitations of the system were also obvious. The control board did some research and determined that a work-flow engine was needed and settled on the TeamTrack product. A work-flow engine, while at its heart a database, is much more than a just a database. A work-flow engine allows for the custom specification of work-flow processes, graphic visualization of the process, and automated email notifications of team members involved in work-flow processes, when their actions are required. Custom reports can be defined to provide reports on issue resolution per product or suite of related products. The issues we track are categorized into Software Change Requests (SCRs) and Discrepancy Reports (DRs). SCRs are for requesting new or modified software behavior, while DRs are for reporting what are suspected to be bugs.

A charter problem for the control board was the issue of software releases. Like issue tracking, software releases prior to the control board formation were ad hoc. This issue has been formalized by developing two levels of software release: Baseline Deliveries and Emergency Service Packs. The intent of the two delivery mechanisms is very different. A Baseline Delivery is a coordinated collection of changes and fixes across the full suite of products, and requires some level of verification prior to delivery. While an ESP also typically delvers a suite of software components, they are typically based on less sweeping changes, require less verification, and are targeted at the special needs of a specific installation, which cannot wait for the next baseline delivery. Baseline Deliveries require formal verification, and what this verification consists of has evolved with each delivery. Testing has evolved from documented unit testing to integrated testing in the presence of a third party observed selected by the IAEA, and is the subject of the next section of this paper.

PRODUCT TESTING

Installation and testing of the UNARM product has always been somewhat of a cooperative effort between the IAEA and the UNARM developers. In the early years of UNARM development a facility installation and checkout would be the joint responsibility of both the IAEA and the UNARM developers. In recent years, however, specialization of responsibilities has been developing. Product development starting around 2002 focused on making software installation easier, and as a result developers were no longer essential during system installation, putting that responsibility more squarely on the shoulders of IAEA personnel. As a response to shifting IAEA

responsibilities, product testing has shifted much more to the development team. There is still and may well always be some degree of overlap in the installation and testing activities between the two organizations, but these roles are much more specialized and clear-cut than in the early days.

Several types of testing have been performed on different version of the UNARM product in recent years. In the initial Baseline Deliveries, Los Alamos National Laboratory (LANL) personnel provided documented unit testing and IAEA personnel provided integrated testing. For older, well established products, unit testing of modifications and fixes was performed and documented, while for new products, test plans were written, performed and documented. Two recent products, the Nondestructive Assay Review – Nuclear Components (NDAR-NC) package, modules and Baseline 2 Revision 1 (B2R1) have been and continue to be tested very thoroughly, and testing of these products will be discussed in detail.

NDAR-NC DCOM MODULE TESTING

The NDAR-NC DCOM modules represent an intermediate step in the evolution of the UNARM product. The data import and analysis capabilities of the UNARM software are currently being repackaged as Component Object Model (COM) and Distributed Component Object Model (DCOM) modules, so that the capabilities of the UNARM software suite can be leveraged in more versatile software architectures than simple applications. The Rokkasho Reprocessing Plant (RRP) in northern Japan is the prime example of where a more versatile software architecture was required. The IAEA is currently developing a review system, NDAR, with the nuclear computations being supplied by a subset of the UNARM capabilities referred to as NDAR-NC. NDAR-NC is a collection of COM and DCOM modules. COM and DCOM modules are dynamically-linked libraries with a generalized interface that can be easily linked to from a variety of programming languages. DCOM is a variant for distributed applications, and is required at RRP because the processing of the voluminous data may occur across multiple nodes.

Initially, the NDAR-NC testing consisted of functional and performance testing of the entire set of COM and DCOM modules. All data import options and analyses used at RRP were covered in this testing, and where the same analysis could be applied to different types of data, that variation was also tested. Where capabilities overlapped between the products, the "gold standard" against which comparisons were made was the most recently accepted Baseline Delivery of the software at that time, Baseline 1, Revision 2 (B1R2). In some cases, however, new capabilities existed in NDAR-NC that did not exist in B1R2, and in those cases, spreadsheets capable of executing the needed calculations were created and verified, and used to double check the NDAR-NC calculations. Interactive test platforms were created to produce external calls on the COM and DCOM libraries; high-level parameters could be interactively specified for quick, flexible testing, but where large amounts of detailed information was needed by the interface, pre-configured XML files containing detailed parameter specifications were employed.

The functional testing of the NDAR-NC product has been regressed several times, with the results shown in Table 1. The issues discovered in the initial testing included one improperly specified test that could not be successfully completed, and the inability to meet an arbitrary performance criterion. In the second round of testing, the improperly specified test was corrected and successfully completed, but the arbitrary performance criterion was still not met. In the third round of testing, the performance test was eliminated because a better, less arbitrary performance test had

been developed as part of the iterative testing described below. Throughout this testing all parties were in agreement that despite the failure of the performance test, the results indicated an acceptable performance level and that more meaningful criteria needed to be developed.

Testing Date	# Tests Performed	# Successful Tests	Success Rate
December 2006	26	24	92%
August 2007	26	25	97%
January 2008	25	25	100%
November 2008	14	14	100%

Table 1. Results of functional testing, over time.

Despite this thorough testing, the developers of NDAR still reported anomalous behavior under certain circumstances. There were two notable differences in the circumstances of how they used NDAR-NC, versus how we used it; one difference turned out to be largely irrelevant, and the other significant, but still somewhat mis-leading. NDAR-NC is produced in C++ and our testing done from a C++ platform. The NDAR developers were using C# to link to NDAR-NC. The platform difference seemed suspect at first, but more significantly, the NDAR developers were performing highly iterative tests where a series of COM calls were performed in a loop for hundreds or thousands of iterations. To further complicate the picture, the NDAR developers typically ran two or more such iterative tests simultaneously. Under these circumstances, unexpected failures of COM operations would occasionally occur. Although the code had been verified correct, under stressful conditions seemingly random failures could occur.

To attack this problem, we first verified that we could reproduce the results using a C# test platform provided to us by the NDAR developers. Next, we demonstrated that the same issue could be produced running multiple C++ performance tests simultaneously, proving to ourselves that C# / C++ interactions were not part of the problem. Finally, a specialized test rig was built for studying this problem in detail. To simulate actual usage of the NDAR-NC modules, we needed to be able to simulate specific sequences of NDAR-NC library calls known as process sequences. A generalized process sequence can be described as follows: find a radiation event, find a period of background radiation close in time to the event, perform a quantitative analysis on the event and its background to arrive at a Pu or U mass value for the event. The test rig had to be able to represent any of the major process sequences, had to be fully automated for iterative testing, and needed to be able to collect and analyze large amounts of data on the unexpected failures of COM operations. To simulate NDAR running multiple parallel processes of process sequence computation, multiple test platforms would be run simultaneously. In addition, it was necessary to introduce random variability in the timing of the calls to avoid iterative parallel testing from falling into cyclic operation.

A system for easily configuring and fully automating process sequence calculations was developed. One type of XML file allowed specification of the components involved in a process sequence calculation, while another type of XML file allowed for the complete specification of all parameters needed to control any of the NDAR-NC components. To collect data, the test rig recorded to a file the success or failure of every call made along with the complete text of error messages in the case of failure. Typical tests could involve several hundreds of thousands of such entries. To minimize the impact of data collection on the testing, large buffers of output were collected and infrequently

written to the disk. Because of the large amounts of data collected, an automated analysis capability was developed. C++ exceptions were the fault of interest. The random variability on call timing and data size interval had also introduced the possibility of "valid failures". Valid failures were error conditions induced by random variations and an acceptable system response, but C++ exceptions were always a problem and never valid. The error analysis could distinguish C++ exceptions from valid failures, and count them. This result was presented as the percentage of calls resulting in C++ exceptions out of the total number of calls made. An error rate of less than 0.1% was deemed acceptable. Also, the error analysis reported the growth of unreleased memory across the lifetime of the test rig.

The test rig took some time to fully perfect, but intermediate versions were sufficient to run iterative parallel testing for several months prior to the completion of the project. A mocked up distributed test bed already existed for the functional testing of NDAR-NC, and with slight modification this test bed was re-used for the iterative testing. For months, automated tests were run day and night and through the weekends. The test rigs were run in windows capable of catching and reporting on C++ exceptions, such that the causes of the failures could be tracked down and corrected. Memory growth could be watched, related to particular components, and alleviated. Two general categories of problems were eventually isolated: minor memory usage anomalies that had always existed in the "legacy" code of the older applications, but that were not a problem until tested in an iterative fashion, and problems in the usage of so-called "smart pointers" an element of the COM infrastructure. Ironically, "smart pointers" are intended to make programming with C++ pointers easier and more transparent, but until you have studied them deeply, they tend to do the opposite.

Several different tests were planned and executed using test rig described above. Two are of particular interest, from the standpoint of reliability testing. A set of five process sequences was identified that between them tested every NDAR-NC component, including testing any given component on multiple data types, where applicable. This test was considered the acme, and was therefore run for 500 and 10000 iterations of all five sequences simultaneously, at different times, with results of this test presented in Table 2. The rate at which unexpected C++ exceptions occurs has improved over time to the point that only a small handful occur in a test that makes approximately 750,000 method calls over a period of about 1 week. Furthermore, it is now suspected, but not conclusively confirmed, that the few exceptions that do occur are due to an oversight in the test rig that allows for resource conflicts between simultaneous processes under very rare circumstances, circumstances that would not occur in actual NDAR operation.

Date	Iterations	Error rate (%)	Errors (abs # errors)	# Method calls
Aug. 2007	500	.01	6	~50000
Nov. 2008	500	0.0	0	~50000
Jan. 2008	10000	.0005	3	~750000
Nov. 2008	10000	.0004	4	~750000

Table 2. Error rates for iterative testing of 5 test sequences simultaneously.

The most challenging test was one in which a severe network outage was simulated. The NDAR-NC components would not be able to receive data under that condition, would not be able to do their analyses, and so needed to gracefully record the network problem to emergency log files and then await the network's recovery to resume activity. The ethernet connector between the data and

analysis nodes was disconnected in the middle of a long iterative test. As expected a flood of error messages were recorded in emergency logfiles on each node, and then activity reduced to a minimal level until the network was re-established. Normal activity automatically resumed.

BASELINE 2 REVISON 1 TESTING

Describe how SCR / DR testing was done and plans for integrated testing.

SUMMARY

The US Support Program has worked with us over the years to implement the formal software engineering efforts and associated testing described in this paper. The quality of our products and services has demonstrably increased thanks to this support. MORE

REFERENCES

LA-UR test plans?