Report No. BNL-107824-2014

The U.S./IAEA Workshop on Software Sustainability for Safeguards Instrumentation:

Report to the Office of Nonproliferation and International Security (NA-241)

Susan E. Pepper[1], Chris A. Pickett[2], Al Queirolo[1], Katherine M. Bachner[1], and Louise G. Worrall[2]
[1] Brookhaven National Laboratory
[2] Oak Ridge National Laboratory

April 2015

BROOKHAVEN NATIONAL LABORATORY

OAK RIDGE National Laboratory

NNSA National Nuclear Security Administration

DISCLAIMER

Table of Contents

Appendices:

US/IAEA Workshop on Software Sustainability for Safeguards Instrumentation

Report to the DOE NNSA Office of International Nuclear Safeguards (NA-241)

1. Introduction

The U.S Department of Energy (DOE) National Nuclear Security Administration (NNSA) Next Generation Safeguards Initiative (NGSI) and the International Atomic Energy Agency (IAEA) convened a workshop on *Software Sustainability for Safeguards Instrumentation* in Vienna, Austria, May 6-8, 2014. Safeguards instrumentation software must be sustained in a changing environment to ensure existing instruments can continue to perform as designed, with improved security. The approaches to the development and maintenance of instrument software used in the past may not be the best model for the future and, therefore, the organizers' goal was to investigate these past approaches and to determine an optimal path forward.

The purpose of this report is to provide input for the DOE NNSA Office of International Nuclear Safeguards (NA-241) and other stakeholders that can be utilized when making decisions related to the development and maintenance of software used in the implementation of international nuclear safeguards. For example, this guidance can be used when determining whether to fund the development, upgrade, or replacement of a particular software product. The report identifies the challenges related to sustaining software, and makes recommendations for addressing these challenges, supported by summaries and detailed notes from the workshop discussions. In addition the authors provide a set of recommendations for institutionalizing software sustainability practices in the safeguards community.

The term "software sustainability" was defined for this workshop as ensuring that safeguards instrument software and algorithm functionality can be maintained efficiently throughout the instrument lifecycle, without interruption and providing the ability to continue to improve that software as needs arise.

A working paper was prepared by the workshop organizers as a read ahead document for the workshop participants. The working paper is included here in Appendix 3. A report to the workshop participants was prepared and distributed in August 2014. The report is included here as Appendix 4 and is archived under accession number BNL-105966-2014.

2. Workshop Objectives

The United States and the IAEA convened the workshop on *Software Sustainability for Safeguards Instrumentation* to identify strategies for improved software development and maintenance practices for IAEA safeguards instrumentation software. The organizers assembled a cross-section of diverse safeguards instrumentation software stakeholders, including users, developers, vendors, and sponsors, to identify strategies for ensuring that critical safeguards instrumentation software products continue to be available for use by the IAEA and the international safeguards community as required, that relevant software is sustainable, and that software functionality does not degrade over time.

3. Software Sustainability Challenges and Solutions

During the workshop, the participants were presented with information from a variety of experts and then divided into three facilitated breakout sessions. In the breakout sessions the participants discussed nine case studies that were developed by the workshop organizers to promote discussion of safeguards software challenges and to elicit suggestions for improving practices for safeguards instrumentation software development and management. Each breakout session had two facilitators and a note taker. The participants identified a number of problems faced by the individuals, groups, and entities that develop, use, and maintain safeguards instrumentation software. The major challenges and frequently proposed solutions are discussed below. Combined results of the breakout sessions are documented in Appendix 3.

3.1 Sustainability Practices

Challenges: While sustainability culture was not specifically discussed, there were many comments from the participants regarding the lack of standard institutional practices that are necessary for software sustainability. Knowledge management, chain of custody, and software stewardship practices are all examples of sustainability practices that are absent, and are all areas in which the IAEA and other stakeholders can improve the status quo moving forward. The lack of a software inventory, established and/or effectively distributed standards and requirements, and lifecycle planning are all indications of weak institutional commitment[1], contributing to poor sustainability practices.

Solutions: All projects should be initiated and led by a user champion who is responsible for the particular code. Code-focused user groups should be established to socialize the code and share best practices. Knowledge management practices should be incorporated to ensure that

[1] "Institutional commitment" means that the IAEA is committed to a project as an organization and that the project will survive a reorganization or the reassignment or departure of a staff member who is the project's champion.

no code is upgraded or significantly modified without widely disseminating the necessary knowledge to other stakeholders. The IAEA, the vendor community, and Member State Support Programs (MSSPs) should support each other by setting standards for software development, sharing them with each other, and adhering to them. Professional societies can play a role in forming groups of users or other stakeholders who are interested in sustaining software. Software sustainability should become an institutional priority for the organizations that depend on the software. The first step in understanding the requirements for sustaining software for any program or community is identifying the existing software. The participants urged the IAEA to conduct a software audit for this purpose.

3.2 Intellectual Property

Challenges: There is intellectual property (IP) associated with almost all safeguards software. The algorithms that are used to perform data analysis via physics calculations and other scientific functions are associated with achievements that are patented or otherwise protected by the national laboratory or company where the method was first put into practice. In many cases, the programming style has resulted in algorithms being embedded in software in such a way that the software is deemed proprietary in its entirety. As a result, the software may have licensing fees and other requirements that restrict its use and prevent the IAEA from obtaining access to the source code.

Solutions: The workshop participants suggested modular software development that separates the algorithms (the proprietary parts of the software) from the graphical user interface, security, communications, and other nonproprietary components of the software. This would make it easier for the IAEA to obtain access to elements of the software for simple bug fixes and upgrades, and ultimately result in software that is easier to sustain.

A suggestion for working with proprietary code is the "black box," or wrapper, approach.[2] A wrapper enables a user to embed proprietary code and interact with it through an interface. This would allow use of software with defined inputs and outputs and prevent competitors from obtaining knowledge of the proprietary aspects of the code. These approaches require a sophisticated set of tests to ensure the code operates as declared, but precedents exist or are being evaluated by the arms control community (i.e., information barrier concepts). Further investigation of this approach should be considered by the IAEA to fully understand its potential.

The IAEA wants to have an in-house capability to make minor software modifications that do not warrant the time and expense associated with a typical MSSP task.[3] As a cost and time

[2] This is the approach being used by Los Alamos National Laboratory in 2014-2015 in updating the INCC code.
[3] If the IAEA had significant in-house capability for modifying software, they would have to establish a version control management system, assume responsibility for maintenance of their versions of the code, and assume the risks associated with "forks."

savings measure, the MSSPs should investigate the IP contained in safeguards instrumentation software to understand who owns it and consider ways (i.e., nondisclosure agreements) to make source code available to the IAEA without compromising the IP. For new software, IP issues should be addressed prior to the start of development and planning for them should become a software sustainability practice.

There was significant discussion about the potential of open source software. Open source software might give the IAEA access to source code, but it would introduce other challenges such as version control, quality assurance, and security. A proof of principle open source software development project should be conducted to demonstrate the effectiveness of this approach for the IAEA. As part of the project, a cost benefit analysis should be conducted and the cost of ownership of open source software should be assessed. A well-managed open source software product could be used as a benchmark. Specific standards for open source development would be required. The open source community could be engaged to promote collaboration for the development and maintenance of safeguards instrumentation software.

The participants also discussed the advantages and disadvantages of using LabVIEW for instrumentation software; while there was no strong endorsement of LabVIEW, some participants recommended a study to assess its benefits and identify the projects for which LabVIEW might be useful and where it may be inappropriate.

3.3 Development

Challenges: The workshop participants cited lack of standardization and poor requirements and poor project management as concerns related to the development of safeguards instrumentation software. Lack of IAEA standards for software development can result in the developers not understanding or misinterpreting the IAEA's requirements, the IAEA receiving software products that are written in different programing languages and that produce incompatible data streams, and software that cannot be maintained effectively over the full lifecycle of the software. Software that is written in obscure software languages can be difficult to sustain. Poor project management can result in lack of lifecycle planning, miscommunication between stakeholders, delayed delivery of or incomplete software products, cost overruns, and products that do not meet the IAEA's needs.

Lack of interaction and/or communication between developers and other stakeholders was also identified as a weakness. The IAEA is often treated as a third party and their input may not be valued. Moreover, stakeholders tend to work independently and not share their work.

One case study prompted the participants to explore the efficacy of software development by the MSSPs (see Appendix 3, Case Study 2). When an MSSP contracts directly with the vendor, it can be difficult for the IAEA to interact directly with the vendor and participate effectively as the end user. The vendor may not understand the importance of working with the IAEA since its legal obligation is to the MSSP. MSSPs sometimes consider their contribution complete upon

delivery of the software and do not make provisions for the software over its lifecycle. National laboratories are research and development entities and are not for profit organizations; those that develop software do not benefit substantially from software sales and are only incentivized to provide technical support if a sponsor will fund their work.

Another case study highlighted the specific issues related to working with small software developers (see Appendix 3, Case Study 7). The primary risk is the loss of the main or sole software developer due to change in work status, illness or death. An independent developer is also more likely to make nonstandard architectural choices. It would be difficult for another developer to assume responsibility for or understand products resulting from such development.

Solutions: The workshop participants recommended that the IAEA develop software standards and advertise them widely. RAINSTORM, an IAEA standard for remote monitoring interfaces, is a good model for standards but also demonstrates the difficulty that the IAEA has in promulgating its needs and requirements. It was a startling discovery that the majority of U.S. vendor participants interviewed prior to the workshop had not heard of or been made aware of RAINSTORM. The IAEA should use formal requests to MSSPs, or SP-1s, as one means of distributing its requirements; MSSPs should not begin a project if requirements are not provided. Software features, such as user interfaces, can be standardized to avoid duplicative programming effort and to reduce the need for training. Developers should be required to use mainstream programming languages. IAEA standards and requirements should be updated periodically to ensure they reflect the state of the art and new measurement approaches.

With respect to project management, there should be IAEA management approval of, an IAEA champion for, and active IAEA involvement in all projects undertaken on its behalf, including tasks performed by the MSSPs and projects in which the IAEA is a party to the contract. Direct IAEA involvement is necessary to ensure that IAEA standards and requirements are addressed, that the IAEA is involved in all related decision making, including change control, and that changes in the IAEA's planning are taken into consideration.

With regard to MSSP contributions, each MSSP should consider and develop a policy regarding the development of instrumentation software. The policy should require establishing a lifecycle plan that is reviewed and updated periodically, periodic reviews of the software to determine if software updates are necessary, inclusion of the IAEA in the software reviews, and notifying other MSSPs of the results of software reviews. The periodicity of reviews will vary between codes based on the application and level of use of the code. MSSPs should agree with the IAEA in advance of development who will be responsible for maintenance and upkeep of the code, as well as who retains IP rights at the end of development. The policy should address the participation of national laboratories in software development projects to help decision makers understand the ramifications of subcontracting with a national laboratory versus a commercial

entity.[4]

A project plan, schedule and budget should be prepared and all changes should be approved by a change control board. There should be periodic project reviews during the software development to review progress and to reevaluate the need for the software. Project reviews are not intended to be excuses to change requirements or increase the scope of the development, but they are important opportunities to review the status of the project. Shortened development schedules will reduce the possibility of schedule slippages or changes in the environment that would necessitate changing direction or terminating a project and for communication between project participants. Final deliverables should undergo acceptance testing by the IAEA. There should be an institutional commitment to software sustainability that can survive the rotation of the sustainability champions. In addition, the establishment of success metrics would be useful in managing future projects.

A phased approach[5] to software development can help to mitigate some challenges such as project delays or the delivery of a product that does not meet the IAEA's needs. Good planning is necessary to ensure that resources are available to complete the project as specified and that the state of the art of software development and associated technical fields are mature enough and understood sufficiently to reach the desired result. Limiting the scope of the software to bare essentials can help to minimize the development schedule. Breaking the scope down into manageable modules will assist in planning and enable developers to successfully complete parts of the project that can then be implemented by users while later modules are under development. Both approaches can increase the likelihood of success and minimize the chances of cost overruns and schedule delays.

An important part of the overall project management is the development of a software lifecycle plan, which is discussed below, under *Maintenance*. The choices made in the development phase will have profound effects on the maintenance phase of software development. Likewise, the effort invested in developing high quality software (such as planning and project management) will reduce the effort required for maintenance. The lifecycle plan can help to understand those tradeoffs at the beginning of a project.

The workshop participants identified opportunities to learn from internal and external experience. The CRISP joint development and RAINSTORM initiatives should be documented

[4] For example, national laboratories may be more stable than companies, need consistent funding streams, and can sustain software that is not commercially viable; companies own software and have an incentive to sustain it if there is a market for it.

[5] A phased approach, such as the waterfall model, is a sequential process in which software is developed in phases and one phase is completed before the next begins. The common phases are Conception, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance. Agile programming is another model that utilizes cross functional teams to work on the various phases concurrently and provide input to each other. Critics of the waterfall model say that one phase cannot be fully complete before another begins because additional information is learned throughout the process. For the purposes of this paper, the phased approach refers to the Production/Implementation phase, which can be broken down further into manageable steps that result in preliminary products prior to completion of the entire code.

and monitored as they mature so that experience can be used in future projects. Participants thought that the CRISP project is the result of a unique opportunity for two entities with similar needs to collaborate and that it demonstrates the importance of communication with stakeholders and related communities. The participants recommended that the IAEA develop success metrics for CRISP, evaluate the project's success, and document the lessons learned. Similarly, the participants recommended documenting lessons learned from the implementation of RAINSTORM and benchmarking the advantages and disadvantages of these approaches.

The participants recognized that there are unique challenges associated with small companies, but they can be overcome with proper project management techniques, such as using software escrows. A modular, phased approach with frequent reviews will ensure that the developer understands the requirements and the IAEA has sufficient opportunities to provide input. When working with small companies, it is important to practice due diligence with respect to the contractor, for the IAEA to be involved at a technical level, and to have a contract that outlines the responsibilities, scope, and requirements. Long term support for maintenance and upgrades by a small company developer may be cost prohibitive, but must still be planned and provided. The participants recommended that the IP be held by a stakeholder other than the developer, such as in an escrow, to ensure that the code remains available to users. The IAEA and MSSPs may consider establishing requirements for the selection of vendors for IAEA instrumentation software projects to ensure appropriate quality standards are met and risks are reduced and to avoid disreputable or incapable vendors.

Other scientific communities are likely to have experience dealing with software sustainability issues from which the IAEA and the MSSPs can learn. The authors suggest that the 2016 MSSP Coordinators' meeting be used as an opportunity to discuss and address MSSPs' roles in software development.

3.4 Legacy Code

Legacy codes are codes that have been in service for an extended period of time and whose users have difficulty finding experts who can provide support.

Challenges: Legacy codes can be difficult to maintain due to loss of institutional knowledge that results from attrition of personnel and obsolescence of software interfaces. Output from legacy codes may not be compatible with newer software interfaces. Outdated programming languages, syntax, and algorithms are also major challenges that must be overcome.

Solutions: The formation of user groups, periodic workshops, and other efforts that support socializing the codes will help to establish a larger community of knowledgeable individuals. Incompatible output can and has in many cases been addressed by using file format converters, but a more sustainable approach would be to establish and promulgate data file standards to the developer community. There is no known guidance, other than this report, that informs the

international safeguards community on the decision making process related to sustaining, retiring, or replacing legacy codes.

Periodically during the lifecycle of software, decisions must be made by individual stakeholders or groups to update, overhaul, retire, and/or replace software. User groups can help to make necessary updates or perform overhauls during the useful life of a software product. Updates can be made to make the software compatible with newer hardware. The decision to retire software can be made by individual users but will affect the entire community by reducing the number of users and, by association, resources that can be applied to the legacy code.

3.5 Maintenance

For the purposes of this report, maintenance is defined to include all activity from implementation through retirement of a software product, including upgrades.

Challenges: Safeguards instrumentation software has a long lifecycle that can span multiple generations of hardware. The maintenance period can be 20+ years. Software that is not properly maintained becomes a burden to its users due to incompatibility with newer hardware, pervasive bugs, inefficient routines, and out-of-date algorithms. Software developers and subject matter experts (SMEs) are often reassigned following implementation and are not available for the maintenance phase. Safeguards instrumentation software usually does not have a warranty that protects the IAEA by requiring the developer to fix coding errors or provide technical support.

Solutions: As a preliminary step the IAEA should prepare an inventory of codes and associated data such as its programming language, developer, primary purpose, safeguards purpose, years in service, users and level of use, IP situation/considerations, and cognizant personnel/groups. The inventory should be updated routinely. A system for prioritizing the codes, from the IAEA's perspective, would also be useful. This will enable the IAEA and MSSPs to make decisions regarding the allocation of resources for maintenance, keeping software in use, determining when to remove a code from service, and assessing whether a code should be rewritten or replaced. The workshop participants stressed the importance of lifecycle planning for successful maintenance of safeguards instrumentation software. A lifecycle plan should be prepared for software prior to the beginning of development as a project management tool that can be used to determine if adequate resources will be available for the lifecycle of the software. As a minimum, the lifecycle plan should include an estimate of financial and human resources required for development, a cost benefit analysis, an assessment of the project risks, implementation and maintenance, user requirements, standards, configuration control, stakeholder roles and responsibilities, intellectual property management, and code obsolescence/retirement, as well as the long-term availability of hardware on which to run the software. This type of practice is commonplace in commercial industry. Investigation into industry practices for software lifecycle planning should therefore be conducted to provide guidance to future project teams.

A key concern regarding maintenance is the availability of software and SMEs to support the software during its lifecycle. The workshop participants recommended the development of standards for documentation and the formation of user groups as a means for ensuring the availability of human resources during the software lifecycle. Good documentation will enable a new programmer to understand the work of the original programmer and transition responsibility for a code. User groups (including representation of all software stakeholders) will increase the number of individuals and companies aware of safeguards software products, help to keep software alive by promoting interaction between current and future users, and provide a forum for discussion of software that has become difficult to maintain. Proper software archiving, such as in an escrow, can protect the users if the developer is no longer available.

MSSPs can assist the IAEA by considering cost effective options for the maintenance of software. Two options suggested by the participants are factory support contracts with vendors to provide quick response assistance and the placement of a cost free expert in the Department of Safeguards to take responsibility for one or more codes. Both the IAEA and the MSSPs should negotiate warranties in development contracts to cover the initial period of code implementation.

3.6 Knowledge Management/Transfer/Retention (AKA the "bus factor")

Challenges: IAEA safeguards instrumentation software is specialized, has a small user and developer community, and remains in service for many years. Often times, funding is not continuous throughout the lifecycle of the software to support ongoing maintenance. Availability of resources not only impacts sustainability from a maintenance standpoint, but also impacts the continuity of knowledge. Effective knowledge management can be difficult because funding gaps can cause a loss of personnel and institutional knowledge. If human resources are not continuously funded, the experts will be reassigned and may not be available when needed. If knowledge is not transferred to the next generation or the next responsible individual, development and maintenance can be disrupted or become impossible. As a result, it may be difficult to sustain the software product.

Solutions: As mentioned under *Maintenance*, Section 3.5, the participants endorsed the creation of user groups to increase and support the pool of knowledgeable users and developers, to promote the exchange of information between stakeholders, to share information about codes, to increase knowledge of codes, and to encourage cross training, introducing developers to software to which they haven't yet contributed, and succession planning. A user group can become an archive, using its members to store information as to how the code was developed, maintained, improved, and used and how problems were solved over the lifecycle of the software.

The software development can be kept active by ensuring continued funding. Otherwise a dedicated community of users must take ownership of the code to sustain it. Continuing funding over the long term is difficult and in many ways unrealistic due to competition for

funding. Extending the development time, using a phased approach at a lower annual funding level, may be attractive due to a lower annual investment, but could discourage completion due to sponsor fatigue. However, a phased approach has the benefit that some modules will be in service while development continues and required updates to the early phase products can be addressed in parallel with the development of the later phase modules, and lessons learned in the early phases can be applied to later activities. Code that is well-structured and documented can more easily be passed from one generation of users and developers to another. Software and the embedded algorithms should be documented in a clear and consistent manner. Documentation standards, such as those used by industry for software operation manuals, and the use of technical editors were also recommended during the workshop.

3.7 Funding

Challenges: Maintenance of commercial codes is funded in part by vendors, but the extent of their investment is constrained by the market. A vendor will not invest in software beyond its ability to sell it. Commercial entities will not maintain IAEA software versions that do not have commercial viability. In order to be commercially viable, the costs of sustaining a code must be exceeded by sales. Because of the small community of safeguards practitioners and users of associated software, safeguards software would have to be priced unreasonably high to cover lifecycle costs. As a result, software is priced at a level that is acceptable to users and the IAEA has to rely on internal or MSSP resources for maintenance, modifications and upgrades. National laboratories participating in software development depend on government sponsorship for their work.

Solutions: The workshop participants encouraged the development of sustainability plans for critical safeguards software. Understanding that funding is limited, the code audit can inform sponsors and vendors as to which codes have the highest priority and longevity. A software center, such as the Radiation Safety Information Computational Center (RSICC),[6] can manage licenses by leveraging contributions from multiple sponsors and by charging user fees, thus maintaining a funding base for code maintenance.

4. Software Sustainability Workshop Recommendations

Based on the summary of challenges and potential solutions in *Section 3*, above, and the compiled notes from the facilitated workshop discussions documented in *Appendix 3*, the workshop organizers identified the following points as the primary recommendations of the participants and important elements of a roadmap for software sustainability.

A. There was universal agreement from the attendees of the workshop that developing an inventory of codes is an important first step that the IAEA should complete. This inventory should include, but not be limited to, the following:

[6] For more information on RSICC, visit https://rsicc.ornl.gov.

a. The name of the code, the version used by the IAEA, what the code does, and an estimate of how long it will be used
b. A list of all other versions of the code that may exist (including their specific purpose)
c. The instrumentation that the code is used with
d. The time elapsed since the last update
e. System requirements for the code
f. Types of data generated by the code
g. Other software/systems that use data from the code
h. Indicate if code is used to look at archived data (do older versions of the code need to be preserved for this functionality?)
i. Sustainability needs (current and future updates) for the code
j. Relative priority level
k. Owner(s) of related intellectual property

B. There was consensus that the IAEA should develop lifecycle plans for all codes that must be sustained. These lifecycle plans must be maintained and updated annually for the entire life of each code. The plans should be detailed and inclusive of all desired and required features such that the plans become the basis for contracting with vendors and/or MSSPs. The IAEA should have a champion for each of these lifecycle plans. Some of the things that should be included in each plan are: the types of hardware that should be supported, file formats, security requirements, data structures, communication needs, data used by the code that comes from other sources, etc. The lifecycle plan should also contain a timeline that includes plans for full version re-writes and archiving old codes.

C. The workshop attendees agreed that sustaining pertinent codes requires investment. Making periodic investments over the lifecycle of the software was considered the most effective use of resources. Some options that could be considered for obtaining resource commitments for software sustainability are:

a. Incorporate software lifecycle plans into MSSP requests (SP-1s). The requests include a portion of the timeline detailed in the lifecycle plan. The SP-1 would require regular communication between the IAEA and the developers (similar to the practice that to date has worked very well with the OLEM project) and monitor progress.
    i. Program managers may want to seek ways to leverage code development and refurbishment costs with other sponsors (domestic safeguards, other non-proliferation, vendors, MSSPs, etc.)
b. IAEA establishes multi-year contracts with vendors and the IAEA assigns a SME to monitor each contract.

   i. Contract should specify lifecycle element(s) to be worked and should realize the risks associated with the future availability of the vendor.
   ii. Contracts must be actively managed by the IAEA

D. An important theme from the workshop was the use of software that contains IP. The IAEA and others are frustrated by the lack of access to this source code . The United States could assist the IAEA and itself by investigating the extent to which IP hinders the IAEA in applying and maintaining software, the added effort and cost caused by the private ownership of the IP, and how the IP might be managed differently to the IAEA's and international safeguards community's benefit.

E. User groups were identified as a solution to several of the challenges voiced by the participants. The organizers believe that sufficient enthusiasm exists in the community for user groups that no or minimal financial sponsorship would be necessary. The Institute for Nuclear Materials Management (INMM) and European Safeguards Research and Development Agency (ESARDA) have working groups that address technical safeguards issues. Since these particular user groups are involved with measurements and instrumentation, they represent an ideal forum for reporting bugs and developing wish lists of desired features that could be incorporated into future code requirements. Either of these organizations could form a working group on software sustainability or include the topic of software sustainability in one of their existing working groups, such as the Nondestructive Assay Working Group.

Appendix 1: Standard Software Anatomy

The following definitions of software components are taken from Wikipedia.

Algorithm

In mathematics and computer science, an algorithm is a step-by-step procedure for calculations. Algorithms are used for calculation, data processing, and automated reasoning. An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function. Starting from an initial state and initial input, the instruction describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing output and terminating at a final ending state. Some algorithms, known as randomized algorithms, incorporate random input.[7]

Data Acquisition

Data acquisition is the process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer. Data acquisition systems (abbreviated with the acronym DAS or DAQ) typically convert analog waveforms into digital values for processing. The components of data acquisition systems include:

- Sensors that convert physical parameters to electrical signals
- Signal conditioning circuitry to convert sensor signals into a form that can be converted to digital values
- Analog-to-digital converters, which convert conditioned sensor signals to digital values

Data acquisition applications are controlled by software programs developed using various general purpose programming languages such as BASIC, C, Fortran, Java, Lisp, and Pascal.

There are also open-source software packages providing all the necessary tools to acquire data from different hardware equipment. Those packages are usually custom fit, but more general DAQ packages like the Maximum Integrated Data Acquisition System can be tailored.[8]

Data Analysis

Analysis of data is a process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, in different business, science, and social science domains.

[7] http://en.wikipedia.org/wiki/Algorithm
[8] http://en.wikipedia.org/wiki/Data_acquisition

Data mining is a particular data analysis technique that focuses on modeling and knowledge discover for predictive rather than purely descriptive purposes. Business intelligence covers data analysis that relies heavily on aggregation, focusing on business information. In statistical applications, some experts divide data analysis into descriptive statistics, exploratory data analysis (EDA), and confirmatory data analysis (CDA). EDA focuses on discovering new features in the data and CDA on confirming or falsifying existing hypotheses. Predictive analytics focuses on application of statistical or structural models for predictive forecasting or classification, while text analytics applies statistical, linguistic, and structural techniques to extract and classify information from textual sources, a species of unstructured data.[9]

Escrow, Source Code Escrow

Source code escrow is the deposit of software source code with a third party escrow agent. Escrow is typically requested by a party licensing software (the licensee), to ensure maintenance of the software. The software source code is released to the licensee if the licensor files for bankruptcy or otherwise fails to maintain and update the software as promised in the software license agreement.

As the continued operation and maintenance of custom software is critical to many organizations, they usually desire to make sure that it continues to be sustained even if the licensor becomes unable to sustain it, such as because of bankruptcy. Obtaining a copy of the up-to-date source code allows a user to take responsibility for sustaining the software. The licensor, however, will often be unwilling to provide access to the source code, as the source code represents one of their most closely guarded trade secrets. Source code escrow can resolve this conflict by allowing access to the source code only when the maintenance of the software cannot otherwise be assured, as defined in contractually agree-upon conditions.[10]

Firmware

In electronic systems and computing, firmware is the combination of persistent memory and program code and data stored in it. Typical examples of devices containing firmware are embedded systems (such as traffic lights, consumer appliances, and digital watches), computers, computer peripherals, mobile phones, and digital cameras. The firmware contained in these devices provides the control program for the device. Firmware is held in non-volatile memory devices such as ROM, EPROM, or flash memory. Changing the firmware of a device may rarely or never be done during its economic lifetime; some firmware memory devices are permanently installed and cannot be changed after manufacture. Common reasons for updating firmware include fixing bugs or adding features to the device. This may require ROM integrated circuits to be physically replaced, or flash memory to be reprogrammed through a special procedure.[11]

[9] http://en.wikipedia.org/wiki/Data_analysis
[10] http://en.wikipedia.org/wiki/Source_code_escrow
[11] http://en.wikipedia.org/wiki/Firmware

Fork

In software engineering, a project fork is a separate and distinct piece of software that is created when developers use existing source code as a foundation upon which to begin independent development. Forks are software branches and usually represent a split in the developer and user communities. Free and open source software may be forked without permission from the original developer without violating copyright law.[12]

Graphical User Interface

A Graphical User Interface (GUI) is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels, or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces, which require commands to be typed on the keyboard. The actions in a GUI are usually performed through direct manipulation of the graphical elements.[13]

Hardware

Computer hardware is the collection of physical elements that constitutes a computer system. Computer hardware refers to the physical parts or components of a computer, such as the monitor, mouse, keyboard, computer data storage, hard drive disk, system unit (graphic cards, sound cards, memory, motherboard, and chips), all of which are physical objects that can be touched.[14]

Input/Output

In computing, input/output (I/O) is the communication between an information processing system and the outside world. Inputs are the signals or data received by the system and outputs are the signals or data sent from it. I/O devices are used to communicate with a computer. For instance, a keyboard or mouse is an input device for a computer, while monitors and printers are output devices. Devices for communication between computers, such as modems and network cards, typically perform both input and output operations.

Note that the designation of a device as either input or output depends on perspective. Mice and keyboards take physical movements that the user outputs and convert them into input signals that a computer can understand; the output from these devices is the computer's input. Similarly, printers and monitors take signals that a computer outputs as input, and they convert these signals into a representation that human users can understand. From the human perspective, the process of reading or seeing these representations is receiving input; this type

[12] http://en.wikipedia.org/wiki/Fork_(software_development)
[13] http://en.wikipedia.org/wiki/Graphical_user_interface
[14] http://en.wikipedia.org/wiki/Computer_hardware

of interaction between computers and humans is studied in the field of human-computer interaction.

In computer architecture, the combination of the CPU and main memory, to which the CPU can read or write directly using individual instructions, is considered the brain of a computer. Any transfer of information to or from the CPU/memory combo, for example by reading data from a disk drive, is considered I/O. The CPU and its supporting circuitry may provide memory-mapped I/O that is used in low-level computer programming, such as in the implementation of device drivers, or may provide access to I/O channels. An I/O algorithm is one designed to exploit locality and perform efficiently when exchanging data with a secondary storage device, such as a disk drive.[15]

Middleware

Middleware is computer software that provides services to software applications beyond those available from the operating system. It can be described as "software glue." Middleware makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application. Middleware is the software that connects software components or enterprise applications. Middleware is the software layer that lies between the operating system and the applications on each side of a distributed computer network. Typically, it supports complex, distributed business software applications.

Middleware is the infrastructure that facilitates creation of business applications, and provides core services like concurrency, transactions, threading, messaging, and the SCA framework for service-oriented architecture (SOA) applications. It also provides security and enables high availability functionality to an enterprise.[16]

Open Source Software

Open source software is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Open source software is often developed in a public, collaborative manner. [17]

Software

Computer software, also known as software, computer programs or code, is the non-tangible component of computers. It represents the set of programs that govern the operation of a computer system and provide desired functionality. Software contrasts with computer hardware, which is the physical component of computers. Computer hardware and software

[15] http://en.wikipedia.org/wiki/Input/output
[16] http://en.wikipedia.org/wiki/Middleware
[17] http://en.wikipedia.org/wiki/Open-source_software

require each other and neither can be realistically used without the other. Software includes all computer programs regardless of their architecture; for example, executable files, libraries and scripts are computer software. Software consists of clearly defined instructions that upon execution, instruct hardware to perform the tasks for which it is designed. Software is stored in computer memory.

At the lowest level, executable code consists of machine language instructions specific to an individual processor – typically a central processing unit. A machine language consists of groups of binary values signifying processor instructions that change the state of the computer from its preceding state. Instructions may change data in storage, which is not visible to the user, or change data on the screen, which would be visible to the user. The processor carries out the instructions in the order they are provided.

Software is usually written in high-level programming languages that are easier and more efficient for humans to use than machine language. High-level languages are compiled or interpreted into machine language object code. Software may also be written in a low-level assembly language, essentially, a vaguely mnemonic representation language using a natural language alphabet. Assembly language is converted into object code via an assembler.[18]

Software (or Hardware) Specification

A software (or hardware) specification is an explicit set of requirements to be satisfied by the software (or hardware). The specification differs from the user requirements document in that it may dictate how the requirements are met (e.g., what software language will be used, what data structure will be used, and what communications protocol will be used) and it should be based on the user requirements document.

User Requirements Document

The user requirements document (URD) is a document that specifies what the user expects the software to be able to do (e.g., the software will be used to calculate uranium enrichment and will be used remotely). The URD can be used as a guide to planning cost, timetables, milestones, and testing. The explicit nature of the URD allows stakeholders to make sure that all necessary features are included. Often a URD includes priority ranking for each requirement.[19]

Wrapper

A wrapper function is a subroutine in a software library or a computer program whose main purpose is to call a second subroutine or a system call with little or no additional computation.

[18] http://en.wikipedia.org/wiki/Software
[19] http://en.wikipedia.org/wiki/User_requirements_document

They can be used to make writing computer programs easier.[20] Wrapper libraries consist of a thin layer of code that translates a library's existing interface into a compatible interface to refine a poorly designed or complicated interface, to allow incompatible code to work together, and to enable cross language and/or runtime interoperability.[21]

[20] http://en.wikipedia.org/wiki/Wrapper_function
[21] http://en.wikipedia.org/wiki/Wrapper_library

Appendix 2: List of Software Acronyms and Abbreviations

AA	Authorization Archive
ABACC	Brazilian-Argentine Agency for Accounting and Control of Nuclear Materials
ACD	Auxiliary Communication Device
ACIV	Automatic Cobra Image Verifier
ACVD	Advanced Cerenkov Viewing Device
ADAM	Autonomous Data Acquisition Module
AISOCS	Advance In-Situ Object Counting System
ALIP	All-in-One Portable Surveillance System
ALIS	All-in-One Surveillance System
AMSR	Advanced Multiplicity Shift Register
ANL	Argonne National Laboratory
ANM	Alternate Nuclear Materials
AOI	Areas of Interest
AWCC	Active Well Coincidence Counter
BNL	Brookhaven National Laboratory
BWR	Boiling Water Reactor
CANDU	Canadian Deuterium Uranium Reactor
CCTV	Closed Circuit Television
CDM	Core Discharge Monitor
CZT	Cadmium Zinc Telluride Detector (CdZnTe)
CFE	Cost-Free Expert
CEMO	Continuous Enrichment Monitor
CHEM	Cascade Header Enrichment Monitor

CIOSP	Common Inspection Onsite Software Package
CIR	Computerized Inspection Report
Cobra	Fiber Optic General Purpose Seal
CoK	Continuity of Knowledge
COLLECT	Multi Instrument Collect – data gathering computer
COM	Component Object Modules
C/S	Containment/Surveillance
CRISP	Central RADAR Inspection Support Package (now called iRAP)
CSSP	Canadian Safeguards Support Program
CTBT	Comprehensive Nuclear Test Ban Treaty
CTBTO	Comprehensive Test Ban Treaty Organization
CVD	Cerenkov Viewing Device
DA	Destructive Analysis
DARC	Data Analysis and Review Component
DCC	Data Collection Computer
DCM 14	Digital Camera Module
DCVD	Digital Cerenkov Viewing Device
DDG-SG	Deputy Director General of Safeguards
DIS	Digital Imaging Surveillance
DG	Director General
DIPS	Data Input Processing System
DIQ	Design Information Questionnaire
DIV	Design Information Verification
DLL	Dynamic Link Library
DLM	Dynamic Linear Modeling
DMOS	Digital Multi-camera Optical Surveillance

DOE	U.S. Department of Energy
DOS	U.S. Department of State
DRS	Data Review Station
DSC	Data Storage Component
DSOS	Digital Single Channel Optical Surveillance System
DU	Depleted Uranium
DVD	Digital Video Display
DVR	Digital Video Recorder
DVT	Design Verification Test
EC	European Community
ECC	Equipment Coordination Committee
EMIS	Equipment Management Information System
EOSS	Electro-Optical Sealing System
EPROM	Electronically-programmable read only memory
EQUIS	EQuipment Utilization Information System
ESP	Electronic Sensor Platform
Euratom	European Union's nuclear regulatory and verification agency, akin to the IAEA for the European Union
FC	Fission Chamber
FDET	Fork Detector (Irradiated fuel measuring system)
FDMS	Fork detector measurement software
FORTRAN	Computer programming language – obsolete
FPGA	Field Programmable Gate Array
FRAM	Fixed-Energy, Response Function Analysis with Multiple Efficiency
FTIR	Fourier-Transform InfraRed
FY	Fiscal Year

GARS	General Advanced Review Software (for surveillance)
GBUV	Gamma Burn Up Verifier
GDP	Gaseous Diffusion Plant
GEMINI	Surveillance System developed by Aquila
GENIE 2000	Spectroscopy software developed by Canberra
GIS	Geographical Information System
GRAND	Gamma ray and neutron detector
GUI	Graphical user interface
HDIS	HAWK-SG based Digital Imaging Surveillance System
HEU	High Enriched Uranium
HKED	Hybrid K-Edge Densitometry
HLNCC	High-Level Neutron Coincidence Counter
HM-5	Hand Held Assay Probe
HMAC	Hashed Message Authentication Code
HMMS	Hulls Monitor and Measurement System
HPGe	High purity germanium detector
HPSOP	High Priority Safeguards and Other Projects
HRGS	High Resolution Gamma Spectroscopy
HSGM	High Sensitivity Gamma Monitor
I2SIP	Standard – IAEA Integrated Safeguards Instrumentation Programme
I3S	Integrated Inspector Information System
IAEA	International Atomic Energy Agency
ICAS	Introductory Course on Agency Safeguards
ICR	Inventory Change Report
ICT	Isotopic Correlation Techniques
ICVD	Improved Cerenkov Viewing Device

IFSM	International Spent Fuel Management Program
IFSS	Inspector Field Support System
IHVS	Integrated Head End Verification System
IIV	Interim Inventory Verification
ILON	Intelligent Local Operating Node
IMCA	Inspector Multi channel Analyzer
IMCF	Integrated Monitoring System for the Chernobyl Conditioning Facility
IMI	Instructor Manual for Instrumentation
IMS	Integrated Monitoring System
INCC	IAEA Neutron Coincidence Counting Software
INFCE	International Nuclear Fuel Cycle
INFCIRC	Information Circular IAEA Publication Nomenclature
INMM	Institute of Nuclear Material Management
INVS	Inventory Small Sample Counter
ION-1	ION-1 Detector for Spent Fuel NDA
IP	Intellectual Property; Internet Protocol
IPCAS	Improved Plutonium Canister Assay System
IPI	Lead Assessor
IPIV	Initial Physical Inventory Verification
IPSec	Internet Protocol Security
IRIS	Integrated Reprocessing Information System
iRAP	Integrated Review and Analysis Package
IRMP	International Remote Monitoring Project
IRP	IAEA Safeguards Information System Reengineering Project (now called MoSaIc)
IRS	Integrated Review Software

ISEM	Integrated Safeguards Evaluation Methodology
ISIS	IAEA Safeguards Information System (now being called MoSaIc)
ISO	International Organization for Standardization
ISOCS	In Situ Object Counting Systems
ISO 9000	Quality Management standards
ISPO	International Safeguards Project Office, Brookhaven National Laboratory
ISPSG	Information Security Policy Steering Group
ISVS	Input Storage Verification System
IT	Information Technology
ITV	International Target Values
JAEA	Japan Atomic Energy Agency
JNFL	Japan Nuclear Fuel Limited
JPO	Junior Professional Officer
JRC	European Community Joint Research Center
JRMS	Joyo Remote Monitoring System
JSGO	Japan Safeguards Office
JSR-12/14/15	Jomar family of Shift Registers
KAMS	K Area Material Storage
KEDG	K-Edge Densitometry
KG	Knowledge Generation
KM	Knowledge Management
KMP	Key Measurement Points
LALIF	Laser Ablation-Induced Fluorescence
LAN	Local Area Network
LANL	Los Alamos National Laboratory

LEU	Low Enriched Uranium
LIBS	Laser-induced Breakdown Spectroscopy
LMMM	List Mode Multiplicity Module
LNMC	Large Neutron Multiplicity Counter
LOF	Locations Outside Facility
LON	Local Operating Network
LWR	Light Water Reactor
MARS	Video review system designed/developed by Aquila
MCA	Multi Channel Analyzer
MCM	Management Coordination Meeting
MCNP	Monte Carlo Neutron Program?
MGA	Multiple Group Analysis (Plutonium)
MGAU	Multi-Group Analysis for Uranium
MIC	Multi-Instrument Collect Program
MiniGRAND	Miniature Gamma Ray and Neutron Detector
MINI-STAR	Mini Surveillance and Recording System
MIPS	MIVS Image Processing System
MIVS	Modular Integrated Video System (analog surveillance)
MMCT	Mobil Monitoring Container Transport System (Chernobyl)
MMS	Material Monitoring System
MoSaIc	Modernization of Safeguards Information Technology
MOU	Memorandum of Understanding
MOX	Mixed Oxide Fuel
MPC&A	Material protection control & Accounting
MS	Microsoft
MS	Mass Spectrometer (used for destructive analysis)

MSCS	MOX Storage Containment and Surveillance System
MS-DOS	Microsoft Disc Operating System
MSSP	Member State Support Program
MUF	Material Unaccounted For
MUF-D	Material Unaccounted For – Operator-Inspector Difference
MUX	Multi-camera multiplexed closed circuit television
NaI	NaI Detector
NaIGEM	NaI Gamma Enrichment Measurements
NCC	Neutron Coincidence Counter
NDA	Nondestructive Analysis or Assay; Nondisclosure Agreement
NDAMS	Nondestructive Assay Monitoring System
NDAR	NDA Review
NGAM	NDA electronics package developed by Bot Engineering
NGSS	Next Generation Surveillance System
NI	National Instruments
NIM	Nuclear Instrument Module
NMAS	Nuclear Material Accounting System
NNSA	U.S. National Nuclear Security Administration
NPP	Nuclear Power Plant
NPT	Nuclear Non-Proliferation Treaty
NPS	Neutron Pulse Simulator
NRC	U.S. Nuclear Regulatory Commission
NRTA	Near Real Time Accountancy
ODA	Operator Data Authenticator
OIOS	IAEA Office of Internal Oversight Services
OLEM	On-Line Enrichment Monitor

OPD	Operator Provided Declarations
PAC	IAEA Procurement Authorization Committee
PC	Personal Computer
PCAS	Plutonium Canister Assay System
PCSA	Protection, Containment, Surveillance, and Authentication
PDI	Person Days of Inspection
PIL	Physical Inventory Listing
PIMS	Plutonium Inventory Management System
PIT	Physical Inventory Taking
PIV	Physical Inventory Verification
PKI	Public Key Infrastructure
PMA	Portable Mini MCA
PMCA	Portable Mini MCA
PNCL	Passive Neutron Coincidence Collar
POTAS	Program of Technical Assistance to IAEA Safeguards (USSP)
PPAS	Program Performance Assessment System
PrNDA	Portable Nondestructive Analysis instrumentation
PRST	Portable Radiation Search Tool
PSMC	Plutonium Scrap Multiplicity Counter
PTH	Protection Technology Hanford
PTR-32	Pulse Train Recorder
PWCC	Passive Well Coincidence Counter
PWR	Pressurized Water Reactor
QA	Quality Assurance
QC	Quality Control
QCVS	Quality Control Verification Software

RADAR	Remote Acquisition of Data and Review (Euratom)
RadReview	Radiation Review Software – for review of data collected using unattended monitoring systems
RAINSTORM	SGTS standard software interface for remote monitoring.
RDBMS	Relational Database Management System
RDC	R&D Needs Committee
RECOVER	Remote Continuous Verification
REXX	Specialized command language developed by IBM
RFID	Radiofrequency Identification
RHMS	Rokkasho Hulls Measurement System
RMS	Remote Monitoring System
RMSA	Remotely Monitored Sealing Array
RMT	Remote Monitoring Team
RR	Research Reactor; Radiation Review
RRCA	Research Reactors and Critical Assemblies
RRF	Research Reactor Fork
RRP	Rokkasho Reprocessing Plant
RSICC	Radiation Safety Information Computational Center
SAGSI	Standing Advisory Group on Safeguards Implementation
SAL	Safeguards Analytical Laboratory (IAEA)
SANS	Computer Security Training Institute
SAR	Synthetic Aperture Radar
SARP	Safeguards Accounting and Reports Program
SBMF	Solution Blending Flow Monitoring System
SCU	System Control Units
SDIS	Server Based Digital Image Surveillance

SEI CMM	Software Engineering Institute's Capability Maturity Model
SEU	Single Event Upset
SF	Spent Fuel
SFAT	Spent Fuel Attribute Tester
SG	IAEA Department of Safeguards or safeguards
SGCP	IAEA Division of Safeguards Concept and Planning
SGIM	IAEA Division of Safeguards Information Management
SGIS	IAEA Office of Safeguards Information Systems
SGOx	IAEA Divisions of Operations (Inspectors)
SGOA	Operations A (Japan, South Korea, Australia, , DPRK)
SGOB	Operations B (North and South America, India, Iran)
SGOC	Operations C (Europe, Russia)
SGTS	IAEA Division of Scientific and Technical Services
SIAL	Satellite Imagery Analysis Laboratory
SIAU	Satellite Imagery Analysis Unit
SIDS	Safeguards Instrumentation Documentation System
SIMS	Secondary Ion Mass Spectrometry
SIR	Safeguards Implementation Report
SM	Safeguards Manual
SME	Subject Matter Expert
SMIS	Safeguards Management Information System
SMMS	Solution Monitoring Measurement System
SMS	Safeguards Manual for Support
SNF	Spent Nuclear Fuel
SNM	Special Nuclear Material
SNRI	Short Notice Random Inspections

SOH	State of Health
SP-1	Support Program Form 1 – used as the official mechanisms for requests to Member State Support Programs
SPCT	Support Program Coordination Team
SPI	Software Process Improvement
SPRICS	Support Program Information Communication System
SQ	Significant Quantity
SQL	Structured Query Language
SQP	Small Quantities Protocol
S/r	Shift Register
SRD	Shipper-Receiver Differences
SSTS	Subgroup on Safeguards Technical Support (responsible for U.S. Support Program activities)
SGTS	IAEA Division of Safeguards Scientific and Technical Support
SSAC	State Systems of Accounting and Control of Nuclear Material
SSEP	Safeguards Software Engineering Process
STR	Safeguards Technical Report
SURS	Surveillance Review Subsystem
TANCS	Tank level measurement software code
TARGA	Plutonium Isotopic Analysis Software
TCVS	Temporary Canister Verification System
TID	Tamper Indicating Devices
TLDS	Thermoluminescent Dosimeter
TRFS	Two-Way Radio-Frequency Seal
TRO	Toronto Regional Office
TSVS	Temporary Storage Verification System
UFBR	Universal FBR Assembly Counter

UDIS	Updated Digital Image Surveillance
UIMS	Ultrasonically Interrogated Metal Seal
ULTG	Ultrasonic Thickness Gauge
UMS	Unattended Monitoring System
UNAP	Universal NDA Data Acquisition Platform
UNARM	UNattended And Remote Monitoring
UNCL	Uranium Neutron Coincidence Counter
URM	Unattended Remote Monitoring
URMS	Unattended Remote Monitoring System
U/S	Ultrasonic - method of verification
USA	United States of America
USB	Universal Serial Bus
USSB	Ultrasonic Sealing Bolt
USSP	United States Support Program
USVC	United States Voluntary Contribution
UWCC	Underwater Coincidence Counter
VACOSS	Variably Coding Seal System – electronic seal that was modified for remote monitoring
VCAS	Vitrification Canister Assay System
VDIS	Digital Video Surveillance System
VHF	Very High Frequency
VHS	Video Home System
VI	Virtual Instrument
VIC	Vienna International Center
VIFM	VXI Irradiated Fuel Monitor
VIFM Collect	Data collection software for VIFM

VIFM Review	Data review software for VIFM
VLTM	Volume Measurement System for Calibration Measurements
VPN	Virtual Private Network
VWCC	Vitrified Waste Coincidence Counter
VXI	VMEbus eXtensions for Instrumentation
WCAS	Waste Crate Assay System (A or B)
WCSS	Wall Containment System

Appendix 3: Case Studies

Case Studies – Summaries of Discussions from Working Groups

Case Study 1: IP and Access to Source Code

Theme: Intellectual Property and Access to Source Codes

Problem Statement:

The IAEA has numerous software products that they have used for many years to collect and analyze data that were obtained from safeguards inspections. Many of the analysis codes have been in existence for more than 15 years and the code base is quite outdated. It is essential that the Agency be able to maintain and sustain these codes, so they can continue to collect and analyze new data and revisit data acquired from past inspections. These products are generally proprietary, meaning the Agency has no right to see or modify the source code. The code consists of hardware-specific calls to devices that are not always commercially available; there is no easy path to making the codes work with newer technologies, such as operating systems, hardware, etc., without access to the source code. Additionally, the process of maintaining these codes with "middle-ware" solutions is not cost efficient or sustainable. Middle-ware solutions include software and hardware options that are designed to maintain data formats and communications with older systems.

Summary of Breakout Group Discussions:

Most software currently in use by the IAEA was written by an outside entity that has control over the source code. This prevents the IAEA from making any changes in house. In addition, the codes and data file formats are not standardized. Much of the software and instrumentation used by the IAEA was originally developed for domestic and commercial applications in a timeframe when resources were available (via leveraging from domestic programs and other resources) to support IAEA needs. The participants endorsed lifecycle planning, developing an inventory of software, software escrow, regular maintenance, and the adoption of standard requirements. To provide the IAEA some access to the source code, the participants suggested separating the proprietary and nonproprietary parts of the code; there should be no concern about giving the IAEA access to the nonproprietary parts and the proprietary parts can be maintained by the IP holder.

Problems

- Output from older codes may not be compatible with the IAEA interface
    - No clearly defined interfaces
    - No modification of legacy interfaces

- It is inefficient to have numerous software packages with different file formats
- The IAEA safeguards instrumentation market is small
    - It is too small to make demands on the industry
        - Must be a business case for commercial providers to make and support software changes
    - Compatibility issues
- It is not always possible for the IAEA to make upgrades
    - Lack of access to source code
    - Programmers are no longer available
    - Some changes can only be implemented as patches and/or workarounds.
    - Commercial vendor not able to change software – code copyrighted by a national laboratory
- Open source and patched approaches typically result in too many competing versions of the same code ('forking')

Proposed Solutions:

- Use a file format converter to provide standard data streams
- Enable IAEA to develop software internally
    - This has financial and human resources implications and may require non-disclosure agreements with instrument suppliers
- Develop and maintain an inventory and status of all software; the listing should include: vendor, current hardware supported, users, pertinent algorithms utilized, and any variations or modified versions of the code.
- Use standard data formats for common data (e.g., dates)
- Standardize the input and output formats such that they are platform independent
- Conduct regular preventative maintenance of software
- Form user groups and/or topical sessions at conferences that discuss current and emerging needs for pertinent codes. This helps support sustainability planning.
- Use and develop platform independent software (expensive)
- Place software in escrow for archival purposes
- Lifecycle Planning
    - Plan for obsolescence of software
    - Maintenance may be required for 20+ years
    - Keep development active; keep the code alive
        - Plan for funding needed to maintain software

                - Leverage support from other stakeholders/users of codes (each pays a share)
            - Develop an appropriate set of test cases to validate the code when it is modified
            - Encourage continued testing against the environment and operating system
            - Add new features as needed
            - Plan and analyze new work
            - Maintain understanding of code
        - Minimize dependencies on developers and software
            - Modular coding methods
            - Improve documentation and distribute to larger community
        - Algorithms have a longer lifetime than software and can be "reused"
- Establish software requirements
    - Minimum requirements for algorithms and interfaces should be documented
    - Requirements should be distributed to Member State Support Programs and other stakeholders
        - SP-1s should include requirements
    - Requirements can be advertised on the Internet
    - Must keep up with hardware and the state-of-the-art
    - Set standards, such as RAINSTORM, U.S. Department of Homeland Security systems
    - Remain flexible
    - Establish acceptance criteria for externally developed software
- Analysis codes
    - Separate proprietary and non-proprietary parts and make the non-proprietary components available for vendors to use
    - Put proprietary components in a "black box" and enable use with defined inputs/outputs. If the system continues to support those I/O, the software will function correctly through all upgrades.
- Middle-ware
    - Minimize the need for middle-ware (on a case-by-case basis). This can be accomplished by the IAEA establishing and promulgating standards for data and data file formats.
- Case Studies
    - Study examples of successful instrumentation software for lessons learned and best practices
        - MCNP

- RAINSTORM
- CRISP

Case Study 2: Support Program Process

Theme: Building IAEA Self-Sufficiency & Sharing Source Code

Problem Statement:

Over the years the IAEA has utilized many software packages supplied to them from Member State Support Programs (MSSPs). These packages have proven to be useful and highly beneficial to the Agency, and it has become dependent upon them to accomplish its missions. However, the IAEA does not own the intellectual property (IP) or have access to the source code for these programs. This makes it difficult to implement needed updates or timely modifications (to support new commercially available operating systems, hardware, data formats, etc.). The Agency is often dependent on the MSSPs to make needed changes to software. The MSSP process can be slow and, therefore, inefficient for making minor software changes.

For this case study we will analyze the U.S. Support Program (USSP) process for modifying and updating software owned by non-IAEA entities (such as national laboratories or companies).

The USSP process is described below:

1. The IAEA generates a request describing the needed work
2. The request is transmitted to the USSP, which forwards it to a national laboratory or contractor for bid
3. The bid is provided to and reviewed by the USSP and IAEA representatives, and if accepted, is sent to the U.S. government for approval of funding
4. The U.S. government approves it or requests modification/discussion
5. If the funding is approved, a contract or other agreement is placed with the source code owner
6. The work starts
7. The code modification is completed and sent to the Agency for testing
8. If the modified code does not work properly, the source code owner and/or team may be sent to Vienna to work with the Agency to troubleshoot the problem
9. The code is eventually fixed and implemented

Additional time is required for IAEA in-house preparation, which includes review and approval of the request before it is submitted to the USSP for consideration. In the best case scenario for this process, funding may reach the source code owner within six months. Sometimes the process takes much longer. The availability of money and amount of time associated with this process significantly impacts the ability of the IAEA's Division of Safeguards Technical and Scientific Services to respond to its internal and external customers.

Summary of Breakout Group Discussions:

The MSSPs provide valuable support to the IAEA, but the process is sometimes too slow and bureaucratic to be fully effective. The IAEA can be left out of the communication between the MSSP and the contractor. The participants suggested establishing a separate process for small software projects, such as bug fixes. All software projects should include a lifecycle plan but MSSP projects do not usually include them. The participants suggested umbrella tasks and other options for expediting support to the IAEA. They also endorsed the formation of a working group on software sustainability. Effective and efficient software support would be a good topic for discussion at the 2016 MSSP Coordinators' Meeting or the 2015 INMM Annual Meeting in connection with establishing a working group.

Advantages of working with MSSPs:

- Provides access to national laboratory talent, expertise, and capabilities

Problems associated with the MSSP model:

- The MSSP process
    - Slower than working directly with a vendor
        - Inserts an unnecessary "middle man"
        - Sometimes requires iteration
    - Complicated
    - Costly
    - Bureaucratic
    - Barrier to small tasks - may be too formalized for minor software changes that can be done quickly
    - Inadequate communication
        - IAEA can be left out of the communication between MSSP and contractor
        - IAEA does not always clearly communicate requirements
    - Contracts are not set up to address the life cycle of the software
        - Maintenance is not built into the process
        - Continuity of knowledge for lifecycle support is not addressed
        - Do not address contributions from third-party
- MSSPs are more maintenance-oriented than development-oriented
- Funding budgeted for maintenance, testing, and documentation at the beginning of a project can get redirected for other activities when cost overruns are encountered during development

- Human resource may only be one deep and the right people may not be available when needed.
- Requirements
    - Requirements may change after the process has started
    - IAEA does not always clearly communicate requirements, especially in SP-1s

Proposed Solutions:

- Distinguish between small and big fixes
    - Establish appropriate mechanisms to address each
- Lifecycle Planning
    - Development partners should understand that the maintenance can cost more than development
    - Establish a plan for lifecycle support
        - Option: MSSPs offering to assist the IAEA accept all lifecycle costs
        - Option: IAEA supports maintenance through regular budget
        - Option: MSSP sponsors a CFE to support maintenance (could be an ongoing, long-term requirement)
        - Option: Look for ways to leverage support from other programs/partnerships that use these codes and could benefit from similar changes
    - Include lifecycle plan in the SP-1 request.
        - Example: Use of short-term consultancy on the order of once a quarter for 2 or 3 weeks
        - SP-1 request should support periodic direct communication with end users
        - Longer SP -1 contracts should be considered to better provide timely life-cycle support
        - Sustainable programming methodologies should be required for the development of new codes
    - Include CFE/consultant support as part of the lifecycle plan in the beginning of a project.
- Task models:
    - Umbrella tasks can be used to expedite the request process
    - Establish a task with pre-approved funding to respond to short notice, small effort software maintenance needs

- Use RAINSTORM as an example of a software standard. (IAEA should establish software standards and requirements for all requested instrumentation).
- Maintenance Options
    - IAEA contracts directly with the vendor
    - An MSSP contracts with the vendor on behalf of the IAEA
    - Have an umbrella task for software maintenance to expedite the request process
    - Industry sets aside funding (who funds is not important) and agrees in advance as to how the funds will be used
    - Identify contractors and place contracts for software maintenance in advance
    - Following delivery, IAEA assumes responsibility for all software maintenance and sustainability (would require access to source code)
- Form a standing working group/team to focus on particular sets of codes (i.e., codes used for NDA, codes used for surveillance, etc. (best practice)
    - Would help to resolve the issue of having qualified human resources by encouraging cross training and succession planning.
    - May include multiple support programs.
- Increase awareness of the importance of software maintenance. A few separate models for setting resources aside for maintaining SW. Use of several contractors to be responsible for maintaining specific software. This is discretionary budget for multiple SW packages and contingency. One model is to have a specific contract with the vendor to maintain their SW.
- Improve project management within the MSSPs
- The IAEA and MSSPs should discuss the request process to see if there are changes that can be made to increase efficiency (agenda item for the 2015 USSP Biennial Review Meeting or the 2016 MSSP Coordinators' Meeting or interim discussion at 2015 INMM Annual Meeting)

Case Study 3: Joint Development Partnerships (CRISP)

Theme: Building IAEA Self-Sufficiency & Sharing Source Code

Situation Analysis:

This case study explores the Central RADAR[22] Inspection Support Package (CRISP) as a joint-development partnership between the IAEA and Euratom.

The IAEA currently uses a wide variety of containment, surveillance and nondestructive assay (NDA) equipment to monitor facilities under safeguards. These instruments were developed over a long period of time, and each was developed largely independently of the others. The data streams from each instrument are very similar in content, but they are stored in different file formats, imported and displayed by various software tools, and analyzed using methods developed by independent developers. The development, training, installation, and maintenance costs associated with such a broad range of independent software products are high and continue to grow. Developing and maintaining training for the inspectors is complicated and costly. Inspecting data from a given facility may require training and utilization of three or more software products. Given these considerations, the IAEA initiated a project with the following goals:

- Provide a single, common interface to inspectors for data review and analysis
- Give the IAEA ownership of and access to the software source code
- Create a simple and generic interface so that future development can be specified more clearly and easily added to the software as semi-independent software modules

Many technical alternatives were considered in coming to a conclusion about the path forward for such review software. Existing products were analysed with respect to proprietary disposition, maintainability, and inclusion of needed features and overall cost of development. The conclusion of this technical comparison was that the Euratom CRISP product offered the most viable path forward in developing an all-in-one software solution. CRISP was, therefore, selected for a joint-development effort between the IAEA and Euratom for the following reasons:

- Euratom began CRISP independently and has invested significant time and resources developing the product that the IAEA can now leverage
- Euratom is amenable to sharing this code and entering into an agreement by which the IAEA and Euratom can develop common functionality, thus reducing risk to either party to undertake future development tasks.
- Significant cost savings can be realized through this option because the product is non-commercial
- Intellectual property rights would be granted to the IAEA, allowing small changes in the code to be done quickly and at little to no cost

[22] Remote Acquisition of Data & Review

- The current product already contains much of the required functionality required by IAEA inspectors
- Newly developed equipment that is used by both agencies will need analysis software, and by entering in a partnership to develop this code, both agencies can save time and money during software development and then have a common product used by inspectorates of both agencies.

Summary of Breakout Group Discussions:

The IAEA's joint development of CRISP with Euratom was seen to be a good initiative. The two organizations have a common goal and can leverage each other's contributions to obtain the product. The IAEA will have access to the CRISP source code and, therefore, will be able to maintain it in house. Some participants questioned what would happen if one party abandons the projects and whether it is reasonable that this model could be repeated in other projects. The participants suggested that the partners use a change control board to assess all changes to the project and its requirements and noted some best practices for successful partnerships.

Benefits of Joint Development Partnerships:

- IAEA benefits from basis developed by Euratom
- Cooperation towards a common goal though the organizations are different and have different goals
    - Mutual benefits
    - Introduces new ideas
    - Leverages resources
    - Versatility
    - Increased number of programmers familiar with the code
- Design
    - Flexible with interfaces; modular approach; expandable
    - Extendable functionality (& works on different platforms)
    - Process based software
    - Transparency with interfaces; communication is defined
- Can be considered a "win-win" for partners
- Source code is shared
- Allows for quicker bug fixes than would be supported through MSSPs or IAEA-only development because there are two organizations using the product, more individuals interested in the fix, and more resources as a team that can be directed to fix the bugs

Problems associate with Joint Development Partnerships:

- If one party walks away, the remaining party would be responsible for the entire project
    - How do you manage value transfer? (who gets what and what does it cost?)
    - Both are owners of "enhanced version"
    - Conversely, there is pressure for the IAEA not to leave the project even if it is not fully meeting the their needs
- Partnership
    - Bureaucracy increases
    - Development is slower due to negotiated process
- Legal considerations, e.g., it may be unclear who, IAEA or Euratom owns the code, and this can lead to future disputes over ownership
- Transitions/implementation with commercial software
- Limited modularity that results in elements of the code not being compatible with other codes or applicable for other uses in the future
- Regression analysis cannot extend to modules provided by other vendors, e.g., third party algorithms
- Security concerns, e.g., who, Euratom or IAEA, has final control over code security
- Because the IAEA joined an ongoing Euratom project, they did not investigate the commercial market and take advantage of potential economies that could have been realized
- Not a viable model for commercial software or for multiple MSSPs
- Not a suitable process for small tasks
- Partners may not be available for future support
- This case was the only example to date of a successful partnership
- Slower development timeline than a commercial development
- So far, this is a data set of one; the community should see if this model can be repeated.
- The community needs more experience with partnerships.
- This is not a viable option for commercial software.

Recommendations:

- Use a change control board to evaluate all changes to the project plan (best practice)
    - Avoids conflicting goals
    - Addresses cost impact
    - Allows customization through modularization, e.g., encryption was a higher priority for the IAEA, so it was added as a module.
- Partnership

  - Keep frameworks high enough level to not conflict with different goals of organizations
  - Partners should apply configuration management in the same way
  - Partners participate in the evaluation of vendor bids for contractor selection
  - Resources and risk are shared
- Should continue to consider partnership relationships for mutually beneficial activities
- Identify success criteria, evaluate success, and document lessons learned.
- Conduct a lessons learned review of the CRISP project

Case Study 4: Vendor Supplied Codes

Theme: Best Practices for Sustaining Software & Ways to Implement Sustainable Practices for New Software Developments

Problem Statement:

Over the years the IAEA has obtained and become dependent on software developed and supplied by vendors. The IAEA has had many positive experiences with software development vendors. In general, vendors are very responsive to the problems and needs of the IAEA. In these cases, vendors provide the requested software changes that address the needs of the Agency. However, in other cases, the Agency has experienced problems with these commercial developments that are similar to those experienced with national laboratories (see *Case Study 2* on the support program process). Problems can arise with vendor-supplied software when the vendor maintains the intellectual property and working knowledge for the firmware and/or software and the IAEA requires modifications. Sometimes the IAEA cannot get the service it needs because the vendor (often a small company) is unavailable or concentrating its effort on new developments/products or software solutions for larger customers than the IAEA. In other cases, the IAEA's version of the code that the vendor supplied and the IAEA previously certified is no longer available due to a discontinuation or a new development implemented for other customers.

Summary of Breakout Group Discussions:

One breakout group addressed this case study. The group discussed the need for and practicality of warranties and maintenance contracts. During the lifetime of software, the embedded algorithms may need to be updated. Sustainability required good software practices, including documentation, and a plan for maintenance. Software developers in both the public and private sectors get reassigned when development ends; maintenance and periodic modifications are not sufficient for them to remain dedicated to a software project. Placing software in escrow will protect the client from default of the contractor and can ensure access to the code if the company goes out of business. The MSSPs' role and responsibilities in the software lifecycle should be better defined.

Advantages of vendor supplied codes:

- The IAEA benefits from vendor testing
- Large companies can host many skilled developers and attract the best developers

Problems associated with vendor supplied codes:

- Need for a warranty/maintenance contract needs negotiation
    - National Laboratories do not provide warranties

  - Vendors do provide warranties
  - Unknowns are not covered by warranties
  - Warranties have expiration dates but support requirements do not
  - Maintenance contracts can be expensive
  - Requirements creep can significantly delay the completion of a project
  - Vendor support/development is based on business cases which may differ from the needs of some users
- Control of Software
  - Data
    - The IAEA would have to accept the data structure and the size of data sets as designed by the vendor and may find that it is not convenient to their business.
    - The size of data streams may result in problems (e.g., incompatibility, warehousing issues) that the IAEA cannot correct internally
  - Algorithms may require modification due to advances in science and engineering
    - The IAEA may be dependent on the vendor to change to the code
  - The IAEA version of code may no longer be available from or maintained by the vendor
- Poorly documented software cannot be maintained or modified effectively
- IAEA needs to be more proactive with upgrades
  - There should be a long term plan for software maintenance
- Human Resource (HR) issues with vendors
  - When Bus Factor = 1, maintenance can be delayed or impossible
  - Staff members leave to do other work
  - Staff members can be assigned to other work
  - Many of the companies supporting the agency are relatively small and may merge or go out of business resulting in lost capabilities or staff.
  - Member State Support Programs
    - When project ends, experts are reassigned to other projects
    - There is no product evolution without an active task

<u>Proposed Solutions:</u>

- Development contracts should include a warranty or provide for a follow on factory support contract to assist the IAEA when problems arise
  - Vendors may need incentives to agree to such provisions
- Place source code in escrow
  - Protects against default

    - Could ensure access to code if company ceases to exist or cannot provide support
- If the desired code cannot be shared, consider an alternative
    - Cascade Header Enrichment Monitor – success story[23]
- IAEA and MSSPs should better define the support program role in terms of sustaining technologies

[23] According to Peter Santi, Los Alamos National Laboratory, the Cascade Header Enrichment Monitor software source code was given to the IAEA in 2008 or 2009. The IAEA has successfully managed the software and modified it for use in safeguards implementation in Japan.

Case Study 5: IMCA Software – Portable Nondestructive Analysis

Theme: Legacy Codes

Problem Statement:

The IAEA also has a need to sustain so-called "legacy codes." This term refers to older codes that may have been written in the best choice of programming language or the most up to date coding structure at the time of development, but are not necessarily consistent with modern software approaches and modern methods. An example of a legacy code is presented in this case study.

The *InSpector* Multiple Channel Analyzer-2000 (IMCA) software represents a specialized code for inspectors to acquire, analyze, and report measurement results obtained with the help of the portable IMCA-2000. The software was developed by Canberra Industries (circa 1995-1998). In the course of its extensive use by the IAEA, it has undergone numerous upgrades, which resulted in the current version, V.2.0C+. For several reasons, the software was maintained and upgraded cooperatively by Canberra with contributions from Agency staff. Nevertheless, over the years, the ability to use a substantial part of its original functionality has eroded due to, in particular, the emergence of more modern applications not available in the IMCA tool box (e.g., evaluation of low resolution gamma spectrometry spectra with NaIGEM), and the deployment of LaBr detectors and associated data reduction tools developed internally (LABRod, LabPel applications). IMCA has become increasingly difficult to maintain. At present, the IMCA software needs major redevelopment, mainly for the following reasons:

1. The software was programmed using the specialized command language, REXX, which was developed by IBM. The REXX execution engine is not fully compatible with modern operating systems, and therefore, it requires major modernization. In this situation, a complete redevelopment of the IMCA software based on a modern programming language, preferably with a built-in multi-platform compatibility, may be required.

2. Some of the methods originally implemented in the IMCA, such as the two-region method for the U-235 enrichment determination, have become obsolete and are no longer used by the Agency. These were replaced by more accurate methodologies, such as NaIGEM, LabRod, LabPel, LabGEM, developed both internally and externally, and implemented as stand-alone software programs. Lack of built-in capability in the IMCA for easy incorporation of new analysis procedures and algorithms does not allow implementation of these programs with IMCA. Thus, a re-design of the IMCA software is required to make it more open for the implementation of new measurement approaches and customizable and adaptable to specific measurement needs.

Summary of Breakout Group Discussions:

IMCA was presented to the breakout groups as an example of a legacy code that is written in a nonstandard programming language and whose application has diminished over the years. IAEA knowledge of the code has deteriorated. The participants suggested rewriting the code since the maintenance of the existing IMCA is becoming increasingly costly and difficult. One option is to remove the engine and put it in a wrapper; in this way the user interface can be modernized, additional functionality can be build upon the existing code, and the code can be integrated with other, incompatible codes. The IAEA must contribute to MSSP software development by clearly defining their organizational needs and priorities, providing requirements, setting standards, and actively participating. Lifecycle planning will result in proactive decision making. The participants endorsed the use of cost benefit analysis as a means to determine which codes should be maintained, updated, replaced, or abandoned.

Advantages of IMCA software:

- IMCA is freely available and compatible with many other products
- The REXX code has been adaptable to Windows upgrades
- The REXX source code is available to users

Problems associated with IMCA software and other legacy software:

- IMCA
    - Written in an obsolete, specialized command language REXX
        - REXX is a rarely used language with a very small market
    - It is not maintained adequately
    - There is no lifecycle plan
    - It no longer addresses the original application
    - Many of its functionalities are no longer needed
- There is inadequate support to rewrite all legacy codes
- Scripting for data passage is tedious
    - Only one person in IAEA knows how
    - A better platform is needed (integration is problem, not function)
- REXX software is not supported by IBM
    - IAEA never requested an upgrade to modern software language
    - There is no ongoing support contract
- The IAEA has too many programs that do similar things and compete for maintenance resources
- Replacing (rewriting) IMCA will result in loss of trust in a well-vetted safeguards measurement process

- Libraries of routines called by the software evolve over time and can conflict with the original intention and subsequent usage
- Maintaining legacy codes becomes more expensive with time
    - Some codes should be allowed die. Some may not be needed and the cost of re-development may be cheaper long-term than maintaining some codes.

Proposed Solutions:

- Remove the IMCA engine and put it in a wrapper
    - The function of IMCA is sound, but its algorithms should be integrated with newer tools
- Best practices for software development
    - Use a modular, plug & play structure (I/O, data acquisition, analysis)
        - Modularity would allow for separate modules that could be owned by different vendors and could be modified/maintained independently from each other
        - This would involve rewriting the code, not revising it
    - The requirements should be well defined and articulated.
        - IAEA and vendor(s) should conduct a joint requirement analysis; MSSP(s) could provide funding
        - Better identify the problem – problem analysis, what is needed, what are gaps?
    - Use GENIE 2000 programmable libraries
    - Do not use a specialized proprietary code/scripting language for software development
        - Choose a language that will be around forever, like C or derivative thereof, rather than the latest modern code
        - Choose a compiler that will be around forever
        - Set standards to prolong life, or set a lifetime for code (e.g., 10 years)
    - Conduct joint design reviews as the project progresses
- MSSPs and IAEA should conduct Cost Benefit Analyses:
    - The IAEA should provide official recognition of need, such as documentation in the Development and Implementation Support Programme or an SP-1 or a letter request to MSSPs, not just a recommendation from individuals
    - Need should be presented in the context of a larger plan for software maintenance
    - An upgrade project should only be undertaken if it still serves a purpose for the Agency

    - A cost benefit analysis should be performed to determine which legacy codes should be rewritten and which should not
- Approach to Windows applications is different than approach to Unix programs. The End User will drive approach.[24]
- Consider alternatives
    - The IAEA should work to minimize the use of equipment and instruments that rely on commercial operating systems. Instruments and systems can be built today that are not dependent on specific OS versions.
    - The IAEA could change its procedure such that they can reduce the software and instruments in their inventory
- Compile an NDA inventory and roadmap for the software needed by the IAEA
- Outsourced vs. internal resources
    - Obtain quotes from both (and consider lifecycle costs)
    - Need a dedicated in-house team composed of 50% permanent and 50% temporary staff
- Project Management
    - IAEA needs to sit in the 'driver seat;' problem ownership
    - Lifecycle planning
    - Maintenance contracts are essential

[24] Windows applications tend to be self-contained and execute an entire analysis process whereas Unix applications tend to perform a single step in a sequence of data processing steps. Regardless, developers should avoid using features that may not be supported in the future, should use a standard language, and should know how the data will be used next and format it in a compatible format.

Case Study 6: Development, Support, and Maintenance of INCC – Portable Nondestructive Analysis

Theme: Sustaining Legacy Software and Knowledge Management

Problem Statement:

INCC (IAEA Neutron Coincidence Counting) software is a general purpose neutron coincidence counting program that runs on Microsoft Windows-based computers. It is used for nondestructive passive and active neutron verification applications in unattended and attended modes. It can interpret the pulse counts from shift register electronics connected to neutron detectors such as the High Level Neutron Coincidence Counter (HLNC) and the Active Well Coincidence Counter (AWCC). INCC was developed by Los Alamos National Laboratory (LANL). The Agency has used INCC as the standard platform for neutron measurement applications for several decades.

To support changes in hardware (i.e., counting system and electronics upgrades) and new applications (e.g., higher counting rate applications), LANL has provided the Agency with technical support for upgrading and maintaining the INCC program through funding from the U.S. Support Program. Recently, a new update for INCC is underway by LANL to allow it to work with the IAEA's CRISP system (see *Case Study 3*). This effort required adding support for operation in unattended mode and for neutron data acquisition electronics such as the JSR-15 (manufactured by Canberra), and List Mode modules [including the LANL-developed List Mode Multiplicity Module (LMMM) and the Hungarian Institute of Isotopes' Pulse Train Recorder (PTR-32)] for attended applications. These new user demands, plus the elimination of bugs in INCC, will be fulfilled through its new update.

The knowledge base for this code exists at LANL. The IAEA is very dependent upon this code; hence the USSP is funding LANL to complete required upgrades and/or bug fixes. This current approach for INCC maintenance, wherein the United States provides funding for piecemeal development for multiple users, is expensive and inefficient and, in the end, is not sustainable by the USSP.[25,26]

[25] At the time of this report, the USSP is funding a cost-free expert and two other projects to assist with an update of the INCC code. The first project involves the upgrade of INCC to make it compatible with the Pulse Train Recorder and to provide enhanced functionality with the List Mode Multiplicity Module (LMMM). There are other plans to update INCC in support of the Uranium Neutron Coincidence Counter (UNCC). The second project involves modernizing INCC interfaces and databases to work with Microsoft SQL Server technology and the CRISP interface specifications.

[26] INCC is supported by laboratory program funding; use and distribution is limited by export control requirements.

Summary of Breakout Group Discussions:

INCC was presented to the breakout groups as an example of a legacy code that is difficult and expensive to maintain. INCC was developed by a U.S. national laboratory; this is advantageous because the U.S. government will keep software active. But national laboratory staff can only work on those projects for which they are funded and are reassigned when funding ends. INCC does not have and never had a lifecycle plan. There are multiple versions of INCC that were developed for different users with funding from multiple organizations. The IP rights are convoluted and prevent efficient code modification. The lack of collaboration with the private sector is detrimental. The participants suggested an audit of codes used by the IAEA for safeguards instrumentation as a baseline for addressing software management. The participants listed best practices for software management and advocated the creation of a user group to encourage collaboration and knowledge sharing. Strong knowledge management is particularly important when working with legacy codes.

Benefits of the INCC model:

- Developers feel sense of pride/ownership
- National labs sometime keep codes alive for a national interest, not for a profit since they are not working for profit, but even at national laboratory they have to justify to Government (e.g. US DOE)

Problems associated with the INCC model and sustaining legacy software:

- INCC
  - Needs a comprehensive review and redesign
  - Has no lifecycle plan
  - Does not produce data in the correct format to interface with data reporting and analysis codes
- USSP Issues
  - It is expensive to pay a national laboratory to maintain software
- There are multiple versions of INCC ("Forks")
  - It can be difficult to understand the differences between the versions
  - Add-ons by other researchers as LANL improves it
  - Decreases efficiency but shares burden
  - A reduction in the number of codes and versions of codes (forks) could save money that could be directed towards the reimplementation of legacy codes that are needed.
- There is no collaboration between private vendors and national laboratories on INCC
  - There is no incentive to share bug fixes
  - Stakeholders work independently

  - Version control is difficult with one developer; with multiple developers it requires a coordinator
  - The intellectual property is distributed among stakeholders
  - Software needs a champion as there is no economy of scale (The IAEA did not agree with this point)
    - Champions are needed both for research and development. The champion for the research and development could be the national lab and company, respectively (related to the modularity issue). A technical champion is one who has technical authority. A user champion sets priorities. Funding and leadership champions may also be useful.

Lessons Learned/Best Practices:

- Management of IAEA codes
  - Perform an audit of codes used by the IAEA
  - IAEA, as user/customer, should set priorities
    - Reevaluate priorities every year in consultation with stakeholders (similar to Environmental Sampling working groups)
  - Make sustainability part of the culture
    - Develop roadmap
    - Develop lifecycle plans
  - Develop sustainability plans for critical codes
- Options and best practices for software development:
  - Build upon the experience of CRISP
  - Use open source software
  - Establish software escrow
  - Share codes under license agreements
  - Backward compatibility is important
  - Fully document codes and algorithms and adhere to recognized programming standards to enable universal use of the code
  - Segregate the physics package (algorithms) from acquisition other portions of the code and assign responsibilities accordingly
  - Consider transferring the development and maintenance to a vendor
    - Vendors do not have access to nuclear facilities and material that are necessary to test codes during development and maintenance
  - Replace INCC with a customized version of Canberra's NDA2000
    - Must compare NDA2000 and INCC to determine what functionality is missing in NDA2000

- Knowledge management
  - Document and share information about codes with others to increase knowledge of code and to support new users/maintainers
  - Assign responsibility for knowledge management
  - Make use of knowledge retained in by commercial entities, whose KM is more mature
- Increase collaboration between the users and developers
  - Create user groups
    - Benchmark what others do to maintain code (learn from the others' experience)
  - Combine the Canberra and LANL libraries
    - Coordinate development with Canberra (and other relevant vendors)
  - Use a repository such as GitHub[27]
    - User community is involved
    - Elect lead and steering committee
- Determine the ownership of codes used for safeguards instruments, and investigate the legitimacy of copyright claims and intellectual property ownership (should be part of software audit).
- Options for increased efficiency
  - Separate expert knowledge from other knowledge
  - Understand what maintenance is really required and what role the MSSP must play.
  - Likewise for upgrades. Establish who will use the produce and at what cost. Perhaps distribute though RSICC (or other way) to collect fee? Establish maintenance contracts for codes associated with MSSP to improve efficiency

[27] GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as wikis, task management, and bug tracking and feature requests for every project. (http://en.wikipedia.org/wiki/GitHub)

Case Study 7: Universal NDA Data Acquisition Platform and DCView Software – Portable Nondestructive Analysis

Theme: Knowledge Management with a Limited Developer Community

Here, we presented two case studies that capture issues pertaining to software development projects in the unique environment of safeguards. Issues include ensuring sustained support for software provided by small companies, sustaining and capturing knowledge of legacy codes, and ensuring ongoing support for legacy codes themselves. The first case study is based on the UNAP, a recently completed development project, and the second case study addresses the DCView software, which has been developed over a long period of time and has been in use by the Agency.

Situation Analysis 1:

The UNAP development was completed in February 2014, following a lengthy development and extensive hardware and software testing by the IAEA. The original software developer was selected based on years of experience and knowledge developing software for international safeguards. However, the developer retired before the project was completed, and responsibility for the software was transferred to a one-employee, affiliated company. The affiliate completed the software, responded to test failures, and agreed to provide a one-year warranty on the software following the completion of testing.

The affiliate has acted with professionalism and commitment and assumed all responsibilities from the original developer. However, following the completion of the software development, the affiliate's employee found other work at a large software house and his availability to support the product is limited.

Many of the companies and consultants who engage in software development for the IAEA are small and have limited resources for supporting IAEA needs. In addition, the IAEA typically has very specific and unique requirements for their products that others in the commercial nuclear industry do not need; as a result the IAEA must rely on a very limited and many times small developer and vendor community.

Situation Analysis 2:

The development of the Digital Cherenkov Viewing Device (DCVD) has been supported by both the Swedish and Canadian support programs. The DCVD is commercially available through Channel System Inc., which also provides the associated DCView software to the IAEA. DCView was developed over a ten year period under the auspices of the Swedish Support Program (SWE SP). DCView is now maintained by a small company while SWE SP owns the intellectual property.

The DCView software requires reengineering for the following reasons:

1. Features have been progressively added over a long period of time and the software structure is not maintainable and upgradable to ensure long term sustainability.
2. The addition of new features has become increasingly difficult
3. DCView should be further developed to support off-line data evaluation features and other advanced visualization features that are not worth it to be developed under the current architecture.

Summary of Breakout Group Discussions:

The software for the UNAP and DCVD instruments discussed in the case study were developed by small companies. The participants acknowledged that there are benefits and disadvantages associated with software development by small companies. The IAEA cannot avoid working with small companies because small companies are not discouraged by the size of the international safeguards market, are more agile to respond to the IAEA's needs, and enable the IAEA to work directly with the technical staff. The unique risks associated with small companies are that they are more likely to be unavailable to maintain and upgrade software and they may be less likely to use standard software practices. These risks can be avoided through adherence to recognized software best practices,[28] good project planning and management, due diligence during contractor selection, standardization, establishing a clear understanding of stakeholders' expectations, and strong contracts.

Similarities between the two case studies:

- Both the UNAP and DCVD software were developed by a small company with one employee
- The intellectual property for the components of the UNAP and DCVD systems are owned by multiple parties
- The UNAP and DCVD software were each developed by one individual
- The use of a small company to develop the software for the UNAP and DCVD was a conscious decision and was considered the only logical choice

Difference between the two case studies

- The company that developed UNAP was set up for that purpose while the company that developed the software for the DCVD was an existing company identified by the Swedish Support Program and it has other clients and projects

[28] The UK Software Sustainability Institute is a good source of information regarding best practices for software development. See http://www.software.ac.uk/software-evaluation-guide.

Advantages of the UNAP and DCVD models:

- Small developers can offer advantages such as greater enthusiasm, lower costs and overhead, greater IAEA leverage in the relationship, better customer service, prompt response time, and greater willingness to provide source code.
- Small companies encourage more personal interaction between the client and the technical staff (with larger companies the client works more closely with sales and marketing staff)
- Small companies are not discouraged by the small market associated with international safeguards

Problems associated with the UNAP and DCVD models:

- Problems encountered in the UNAP and/or the DCVD projects
    - The contract for the development was complex and complicated
    - The MSSP decided to retain the intellectual property
    - There were separate contractors for the hardware and software development
    - The user requirements were incomplete or nonexistent
    - The development champion designed the instrument to replace many other instruments and to perform many functions. It was too ambitious and the software had to be too complicated
    - There were too many development partners and the partnerships were poorly structured. It was difficult to determine who was responsible.
    - There was no lifecycle planning
- Risks associated with small companies
    - Small developers can offer advantages (see above)
    - The primary risk is potential loss of developer due to change in work status, illness, or death.
    - In addition, a small developer may make nonstandard architectural choices that lead to a source code product that is difficult for another developer to understand.
- Development projects that are split up and separated between multiple contractors face higher risks of failure and higher costs for project coordination
- For legacy codes that were supplied by small developers, incremental improvements may no longer be sustainable or appropriate, and reengineering may be necessary.

Additional Factors/Outstanding Questions:

- The Agency cannot avoid using software developed by small companies
- A good relationship with a commercial company matters a great deal
- Who should assume lifecycle risks? Can (or should) these be passed to the MSSP?
- Who assumes the lifecycle costs of software and why?
- Fractional outsourcing can be resource intensive and place the burden for integration on the IAEA
    - Project Management can be contracted to another entity

Lessons Learned /Recommendations:

- There are unique risks associated with doing business with small companies (see above), but they can be overcome with proper project management techniques, such as using software escrows (e.g., Iron Mountain)
- Development approaches
        - Modular, phased approach
        - Reduces risk and avoids rework
    - Subdivide the work between multiple companies
    - Have one lead developer responsible for coordinating and integrating all work
    - Use short development cycles
    - Ensure strong project management
    - Leverage COTS software and hardware whenever possible
    - Avoid customization where possible
- Intellectual Property (IP)
    - Consider all IP options
    - Allow the IAEA to hold the IP of the project results
    - Include IP provisions/plans in contracts
    - If the IAEA is contracting with a small company, obtain the source code/IP
        - This is only useful if provisions are made for another entity to assume responsibility
- Project planning
    - Identify risks and mitigation strategies at the beginning of the project
    - Define the deliverables that should be in the contract with the developer (see legal remedies, below)
    - Practice due diligence with the contractor with respect to long term availability for maintenance

    - Reduce risk by asking the right questions, performing a risk analysis, and rolling out the software in phases
    - Plan for the lifecycle of the software
- Project Management
    - With small companies, that may not have expertise in all subjects associated with the project or an overall depth of knowledge, the IAEA has to be more involved to mentor the technical staff. There must be dedicated IAEA project support at the technical level
    - Dedicated IAEA involvement is important to all projects
- Projects Sponsored by Member State Support Programs (MSSPs)
    - Ensure a mutual understanding between the IAEA and MSSP(s) of the desired project outcome and development process
    - Research and learn from other organizations that are the beneficiaries of contributions and are not direct parties to contracts that provide the contributions
    - Ensure a mutual understanding of the project by the IAEA and MSSP(s) (or other benefactor)
- Establish a set of standards for software development
    - e.g., RAINSTORM may mitigate many of these risks (via standardization of requirements)
    - This will help to avoid "quirky architectural choices" by the developer
    - Requirements and/or specifications should be routinely included in SP-1s
    - Avoid over-specification
- Contracting issues
    - IAEA should specify the scope of the contract and the expected deliverables the IAEA will get at the end of the project
    - MSSP could transfer money to the IAEA with conditions attached (e.g. the IAEA may only contract with a select set of companies)
    - Write contracts that include long-term support
        - This can be difficult - either impractical or cost prohibitive
    - Determine what proper terms and conditions should apply to the contract and ensure they are included
- Lifecycle Planning
    - The total lifecycle costs need to be considered and understood from the beginning of the project
    - Could consider maintenance, life span, knowledge management and transfer
    - Should include risk considerations and any associated cost implications (these need to be taken into account somehow)

- Identify core capabilities in the code and determine whether those can be pulled out, preserved, and/or maintained separately
- Contractor Selection
    - Establish requirements for selecting vendors for IAEA instrumentation software projects
    - Consider company staffing profile and capabilities
    - The risks associated with small companies (e.g., bus factor, imminent retirement, competing projects) must be recognized and mitigated
    - Use a larger software house that is less likely to have such problems
    - Request that a small, one-person company contract through a larger company that could alleviate some of the risk (this would increase cost)

Case Study 8: Instrumentation Software Development – LabVIEW as a platform for maintaining systems

Theme: Best Practices for Sustaining Software & Ways to Implement Sustainable Practices for New Software Developments

Problem Statement:

Historically, the software development approach for instrumentation software/firmware has been almost entirely left to the discretion of the developer. This includes the selection of the programming language, the design of the hardware supported by the code, the implementation of embedded algorithms, and the display and format of acquired data. The IAEA Division of Safeguards Technical and Scientific Services (SGTS) is increasingly assuming responsibility for the maintenance of these codes and is interested in exploring this part of the software development process.

The development of dedicated hardware for data acquisition has seen dramatic increases in the cost of development and difficulties in long-term maintenance. In many cases, instrumentation is nearly commercially obsolete by the time it is deployed and maintaining the requisite skills and knowledge base to service these instruments is a significant drain on Agency resources. More significantly, the need to maintain unique, single-use instrumentation reduces flexibility and is an impediment to enhancing the efficiency and effectiveness of safeguards instrumentation. For this case study we will explore the advantages and disadvantages of using LabVIEW (virtual instruments) as software replacement for data acquisition instrumentation.

Summary of Breakout Group Discussions:

One breakout group was in universal agreement that there were no advantages of LabVIEW for safeguards instrument applications, but the other two groups were able to identify advantages. LabVIEW is intended for use in laboratories for experimental software. Use by the IAEA for safeguards instrumentation would require the development and support of new modules, and a LabVIEW capability within the Department of Safeguards would have to be established and maintained. LabVIEW is not be the answer to every programming need and will present some licensing challenges. LabVIEW provides no sustainability benefits. Conventional codes can do everything that LabVIEW can do, but with more effort.

Advantages of LabVIEW:

- LabVIEW is a potential option for standardization
    - Follows standards
    - LabVIEW is not application specific

    - LabVIEW is hardware independent
- LabVIEW is already used by inspectors for in-field review
    - A strategy is needed for continued, systematic use
- LabVIEW has a large, established community of users
    - Knowledge already exists and could be leveraged
    - The use of LabVIEW in processing industries suggests a level of reliability/utility
- LabVIEW can do everything conventional codes can do
- Ease of Use
    - LabVIEW field programmable gate arrays (FPGAs) can simplify the development process for certain instrumentation
        - Another group said LabVIEW should not be used to create firmware
    - LabVIEW simplifies the creation of graphical user interfaces
        - LabVIEW promotes standardization of user interfaces for a variety of different instruments
    - LabVIEW simplifies coding because it comes with many built in drivers, standardization interfaces, etc.
    - Allows for codes to be written in C
- Maintainability
    - Virtual instruments are sustained; bugs addressed by National Instruments[29]
- LabVIEW is flexible (also a problem – could compromise data)
- Reduced cost for potential applications
- National Instruments
    - has a suite of supported instruments in multiple of categories
    - provides instrument simulations
- LabVIEW obviates the need for software architecture
- LabVIEW can serve as a prototype and testing environment
- LabVIEW could be used to develop simple instruments

<u>Problems associated with LabVIEW:</u>

- Name "LabVIEW" may be a nonstarter for some
    - Negative reputation
    - Name implies different use
- General
    - The LabVIEW compiler function is more complex than other compilers
    - LabVIEW provides too many options

[29] Can we rely on National Instruments to address bugs?

  - LabVIEW is probably too complicated
  - LabVIEW results are not reproducible
  - LabVIEW is not the answer to all software development needs
- IAEA's one experience with LabVIEW was not very successful
- LabVIEW is not a particularly reliable way to put systems in the field
- Budget considerations
  - LabVIEW requires a user license
  - Reliance on LabVIEW will require periodic training to keep current with the software, the retention of trained staff, and/or excellent KM
- Updates
  - National Instruments cannot be expected to update software for each individual instrument
  - Frequent updates create problems for maintainability in remote monitoring scenarios
- It is inefficient to add new modules
- For the IAEA to use LabVIEW for instrumentation software, new applications would have to be developed and supported
- Compatibility with Safeguards mission
  - Use of LabVIEW by the IAEA would be challenging because the IAEA's needs are highly specialized
    - The Virtual Instruments provided through LabVIEW are not specific enough for Safeguards applications
  - LabVIEW was built for laboratory experiments
  - LabVIEW was written for non-programmers; the actual code writing is poor
  - LabVIEW is an exploratory tool
  - LabVIEW does not effectively support data analysis applications
  - To access a module (e.g., a small driver), National Instruments' protocol necessitates the download of an entire library
  - The IAEA would need to obtain documentation of the LabVIEW components that are needed
- Sustainability
  - There is no sustainability benefit from LabVIEW
  - LabVIEW code is not maintainable in the long term
    - System hardware is not maintained
- LabVIEW itself may have a limited lifetime
- LabVIEW development times are too long

Solutions/Recommendations:

- Conduct a study to identify those projects for which LabVIEW would be useful
  - LabVIEW may be useful in specific cases
    - In-field attended gamma measurements where the integration actually helps
    - As a test platform to compare different approaches
    - LabVIEW plug ins could be used for research scenarios
  - Consider the costs and benefits of a software development project using LabVIEW
  - Use Build VI (Build Virtual Instrument) driver for existing systems as an example to compare software developed through a traditional approach to software that can be developed using LabVIEW
    - Compare project schedule, project complexity, savings
  - Conduct a study to determine where LabVIEW is used, where it is not used, the experience users have, etc.
  - Use results of study as benchmark for IAEA software strategy, guidelines and policy, and develop a Department-wide policy for the use of LabVIEW for safeguards.
- Alternatives to LabVIEW
  - CRISP is a model for alternatives to LabVIEW in that it demonstrates another software development approach
  - Software architecture should support future changes
- Do not consider LabVIEW to be a cure-all solution; do not require the use of LabVIEW for all IAEA instrumentation software as proposed in the case study

Case Study 9: Open Source Software

Theme: Best Practices for Sustaining Software & Ways to Implement Sustainable Practices for New Software Developments

Situation Analysis:

One of the proposed solutions for IAEA software development and implementation is the use of open source software. The obvious benefit is that the IAEA has access to the source code with all the many benefits this entails (e.g., a potentially cost-free developer community,[30] the ability to fix bugs and/or add features without being locked to a certain vendor, the ability to make modifications in-house, and easier vulnerability assessment (VA) via access to the source code and the ability to compile the source code).

Despite the benefits, open source may not be a panacea. The "guardian(s)" of open source code may not accept new features developed by/for the IAEA or, in some cases, may not even accept bug fixes. This can happen for many reasons. For example, the "guardian(s)" may not want the new features, may not recognize the IAEA as a legitimate user/developer, may not like the coding style, may not agree that something is a bug, and/or may just not care that changes were made. In this case the IAEA can either pay for an entirely new product to be developed or create a "fork" of the open-source software.

> A **project fork** happens when developers take a copy of source code from one software package and start independent development on it, creating a distinct new piece of software.

Forking has the advantage that the IAEA can be the "guardian" of the fork. In this way all of the aforementioned disadvantages go away. Additionally, it may be easier to VA a fork because the IAEA can control how often a new version of the fork is released. However, forking requires the IAEA or its stakeholders to maintain the fork, which includes carefully attempting to integrate pertinent changes from the source of the fork.

Summary of Breakout Group Discussions:

The use of open source software will solve some problems (e.g., it provides greater accessibility to the code and increases the number of cognizant developers) but will introduce other problems (e.g., it will require careful oversight to ensure quality standards are met). Open source software is not necessarily cost free and can have some licensing requirements. Algorithms used in safeguards applications are the intellectual property of their inventors and may not be available for inclusion in open source software. The development and/or use of open source software could include contributions from individuals and national laboratories but

[30] As a result of the workshop and ensuing discussions, the authors learned that some assumptions about open source software in this case study are not realistic.

would likely discourage the private sector from participating due to the noncommercial nature of the products. Configuration control of open source software can be problematic and a guardian would be needed. Many of the advantages and disadvantages discussed in the break out groups were not specific to open source software.

Discussion Points:

- Open Source is one mechanism to allow the IAEA access to source code
    - may be particularly palatable for US national laboratories, since they have experience with it
- The use of open source requires a cost-benefit analysis
- Open source vs. IAEA possession of source code
    - Who should have/needs access?
- National Laboratory issues
    - National labs can do open source developments; IAEA would just have to ask or require that approach
    - A U.S. national laboratory developer may not have control over whether a development ends up as open source software
        - Lab may seek to make it proprietary
        - Sponsor (e.g., DOE) can determine with it will be open source or proprietary
    - National laboratories have experience with open source developments for various end-users (e.g., security applications)
        - Used often to facilitate collaboration among researchers
- Need for trust in the product remains
- Turn-over within the IAEA increases the need to identify a sustainable maintenance plan
- Examples of proprietary software that could become open source:
    - IMCA: if everyone had access, changes could be made and the functionality of the code would improve. Right now, this is not happening. Requires a gate-keeper, however.
    - INCC: if INCC became open source, the IAEA would be able to modify the code internally without assistance from the USSP.
    - Provide crypto-middleware for token: had to cope with evolving requirements and minimal communication was a serious problem; influenced decision-making
- Member States trust the IAEA to handle confidential information in certain circumstances; why would the member states not trust the IAEA to handle software obtained through a license agreement responsibly?
- MSSPs can contribute to sustainability of open source software by ensuring that there continue to be subject matter experts (human capital development)

- Could open-source software be compatible with MSSP developments? A study could be performed to compare the requirements of open-source software development with the practices and constraints of MSSP software development to support.

Advantages of open source software:

- Allows greater accessibility to the code
    - The IAEA can view the software and compile it themselves
    - Having this access closes a security gap because the IAEA can read the code to see if any malware is embedded
    - The quality of the code can be examined
- Facilitates sustainability if funding is not required for maintenance
- Allows for collaborative environments (e.g., GitHub)
- Facilitates modifications because the IAEA can either do the modifications in house or contract out for support
- Less likely than proprietary code to disappear
- Often more successful for development
- Bug fixes can be done more easily and quickly
- Bug reports can be provided with more detail and quality
- Development and maintenance is not a burden on any one entity
- Gets more scrutiny from a wider pool of developers and users
- Documentation may be completed in a more timely manner
- Could be immune to staff turnover in the IAEA if it is supported by an external community of users
- Eliminates or reduces bureaucratic barriers
- IP issues can be resolved through licensing
- Open source contributors can supplement IAEA human resources
    - Talent can be identified through open source collaboration
- Independent development towards objective is possible; can pursue work in parallel paths
- There is a great deal of available open-source software related to image detection/surveillance
- Open Source Culture
    - abandoning less useful branches of programs often results in better quality tools
    - Stimulates creation and consideration of new ideas from different sources
        - Cost-free nature of the software facilitates this behavior

Problems associated with open source software:

- Open source does not always mean accessible to everyone; it sometimes is implemented where only a designated community of individuals have the ability to view and/or modify the code
- Open source options would require the IAEA to have a robust software QA program
- Resource impact
    - Open Source does not mean cost free
        - licenses are still needed in most cases
            - some requirements may be difficult for the IAEA to adhere to due to national laws, etc.
        - What does "free" mean in this case? May be cost free but not free of restrictions
    - Maintaining source code at the IAEA can be expensive
        - The IAEA would create and have to maintain forks
    - Open source development does not reduce resource requirements or administrative burden – it just shifts them elsewhere
        - From lab/support program to IAEA staff or a developer
        - From developer to 'gate-keeper'
    - Oversight/guardianship requires time and financial resources (guardian needed in both closed and open source environments?)
- There is no standard quality control process for open source codes
- Software guardians
    - Open source developments require an engaged, reliable gate-keeper (can contract this out, change gate-keepers, etc.)
    - A single guardian is similar to a bus factor of 1 with respect to developers
    - Guardian must look after quality control
        - the larger the community, the less quality control
- Open-source products can be difficult to combine or integrate (either logistically or due to IP issues)
- Open source codes are continually changing and version control for open source software can be problematic unless the IAEA can maintain its own fork and ensure that all changes within that fork are tested and evaluated for vulnerabilities and performance.
- Open source development may cause reliance on skills of IAEA staff subject to rotation
- Algorithms do not exist as open source code and are not expected to become open source soon

- The limited community for safeguards specific applications reduces the incentives to open source developers who would be willing to fix bugs or otherwise contribute
    - It would be difficult to maintain documentation with smaller, less dedicated communities
- While having access to the source code can give the IAEA confidence that there is no malware embedded, security issues arise due to the number of people who have access to the code and the difficulty of reviewing software that may have tens of thousands of lines of code
- Legal complexity may increase, e.g., if the base code is owned by one entity and the IAEA develops a fork based on the base code, there may be disagreements over who owns the fork.
- Open source developments may require:
    - particular attention to knowledge management and version control.
    - an active user base (not necessarily large) to make this work; often challenging in the field of international safeguards.

Proposed Solutions/Recommendations

- Conduct a proof of principle open source software development to demonstrate effectiveness
- Develop standards for IAEA open source development
    - Include them in SP-1s or requirements for software development projects
    - Advertise them on the IAEA website
    - Include documentation requirements
        - Documentation should be completed by programmer
- Analysis
    - Open source options should be evaluated by the IAEA on a case-by-case basis to determine if the cost of ownership can be supported and if it represents a sustainable approach.
    - Perform a cost benefit analysis to determine whether open source development would save resources for particular cases or categories of software.
        - In some cases, it may just shift the resource burden elsewhere
    - IAEA could categorize software according to certain criteria (e.g., security requirements, mission requirements, usage, and user base) and decide whether certain categories of software might benefit from an open-source approach
- Identify well-managed and well run open source projects to use as models
- Plan for the lifecycle of software
- Investigate licensing options/requirements for open source software

    - Determine what is possible
    - Write an 'ideal' license and see if developers will accept it (IAEA)
- Establish an open source community to stimulate collaboration for safeguards instrumentation software
    - Establish guardians/change control board (CCB)
        - The guardian is usually is the organization that develops the software
        - Establish Quality Assurance/Quality Control requirements (reviewed by CCB)
        - Establish responsibilities (e.g., development, documentation, maintenance)

Appendix 4: Workshop Working Paper

U.S./IAEA Workshop on Software Sustainability for Safeguards Instrumentation

May 6-8, 2014

Working Paper[31]

Susan Pepper, Brookhaven National Laboratory
Louise Worrall, Oak Ridge National Laboratory

Workshop Objective:

The United States and the IAEA are convening a workshop on "Software Sustainability for Safeguards Instrumentation" to identify strategies for improved software development and maintenance practices for IAEA safeguards instrumentation software.

For the purpose of this workshop, "software sustainability" is defined as ensuring that safeguards instrument software and algorithm functionality can be maintained efficiently throughout the instrument lifecycle, without interruption, and providing the ability to continue to improve that software as needs arise.

The Challenge of Software Sustainability:

Today's international safeguards instruments are sophisticated tools that require numerous, complex software applications for instrument control and for acquisition, storage, transfer, and analysis of data. The IAEA's vast and diverse inventory of instruments and the associated software provides a means for the IAEA to apply scientific principles to inspection measurements and, at the same time, meet international safeguards requirements. These instrument software applications run on a variety of platforms, are built from unique source code by developers at multiple public and private organizations, and require development practices and training of users and stewards that are unique to specific subject matter expertise.

Sustaining IAEA instrumentation software is complicated by the many stakeholders and the unique environment of international safeguards; software development for international safeguards is achieved using codes owned by the IAEA, its member states and their national

[31] This working paper is adapted from talking points compiled by the International Safeguards Project Office, Brookhaven National Laboratory (2012) and a follow-on white paper entitled, "Developing a Technical Needs Analysis for Software Sustainability to Mitigate Long-Term Software Maintenance Issues & Costs," by Louise Evans Worrall, Stephen Croft, James R. Younkin, Nathan C. Rowe, and Chris A. Pickett of the Safeguards & Security Technology Group, Oak Ridge National Laboratory (2012).

laboratories, commercial vendors, and individuals. Before new software can be adopted by the IAEA, it must be certified through testing. New hardware for safeguards instruments is often accompanied by new embedded software. Embedded software must be compatible with software used by the IAEA for data analysis. If compatibility was not preplanned, the IAEA will depend on the equipment developer to modify the embedded software for its use or middleware must be developed. Due to the nature of safeguards, the IAEA often has unique software requirements that are not needed by other users. The situation is compounded by the relatively small size of the international safeguards software community of users and developers. Technical knowledge on software operation is often limited to the developers or small communities of expert users. Further, the relatively small user community does not command the attention of large software houses and does not have the luxury of widespread or extensive beta testing for pre-service bug identification. IAEA staff members often find themselves serving as beta testers of new software products, indirectly increasing the overall resource cost of development.

One example of how IAEA instrument software is developed is through United States Support Program (USSP) funding for national laboratories, private companies, or consultants to develop particular instrument hardware and software for IAEA safeguards implementation. In this example, the USSP has typically assumed the cost of modifying and maintaining software throughout its lifecycle including extending the software beyond the end of its planned lifecycle, expanding the capabilities of the software in response to IAEA requests, updating the software to a new platform when the original software platform becomes obsolete or unsupported, and fixing bugs as they are identified or become unmanageable. However, given constrained budget environments and competing demands for limited resources, the USSP may not be able to continue to assume these costs in the future. This workshop is intended to identify effective, efficient, and creative solutions to sustaining IAEA instrument software.

Workshop Format:

In light of challenges described above, the U.S. Next Generation Safeguards Initiative (NGSI) and the IAEA Department of Safeguards are convening a workshop on "Software Sustainability for Safeguards Instrumentation" to be held May 6-8, 2014, at the Vienna International Centre in Vienna, Austria. This workshop will assemble international safeguards instrumentation software stakeholders for informative and constructive discussion of the issues related to software development and maintenance from a sustainability perspective. The objective of the meeting is to obtain feedback from software and instrumentation experts and users to guide the U.S. and other Member State Support Programs to a more effective and efficient process for developing, modifying, maintaining, and sustaining instrumentation software for the IAEA Department of Safeguards. Invited workshop participants include representatives from the IAEA, member state governments and national laboratories, companies, and think tanks. The workshop is designed with presentations in the mornings to provide background information on the issues that face software development for the international safeguards community, and with breakout sessions in the afternoons where case studies of specific situations will be discussed and analyzed for the identification of improved pathways for technical support. Themes have been defined for each day based on information obtained through interviews with IAEA staff and company

representatives. During the concluding session, time will be reserved to allow the participants to talk about the overall results of the workshop and to provide input as to the most pertinent ideas that were discussed and how they might be implemented.

The theme for the first day of the workshop is "Building IAEA Self-Sufficiency and Sharing Source Code and Intellectual Property." This theme arises from the IAEA's desire to increase their self-sufficiency and their desire to have control of the source code related to their instrumentation. There is a corresponding need to make the entire process (software delivery through field implementation) more efficient for all stakeholders. The workshop participants will be asked to consider the obstacles to the IAEA's self-sufficiency, the advantages and disadvantages of giving the IAEA access to source code, and the challenges presented by intellectual property.

The theme for the second day is "Knowledge Management and Sustaining Legacy Software." This theme arises from the reality that the safeguards software user community is relatively small, that the supporting companies tend to be small, and that the aging of the industry is causing a shortage of subject matter experts and software developers with this specialization. Participants will be asked to provide input as to how the community can better support the IAEA by ensuring the transfer and avoiding the loss of institutional knowledge. We will also discuss how to ensure that legacy software will be available to support IAEA safeguards until replacement software is available.

The final theme for the workshop is "Best Practices for Sustaining Software and Ways to Implement Sustainable Practices for New Software Developments." Based on the first- and second-day reviews of the current state of software management and the management of legacy codes, the participants will explore options and identify sustainable practices for new software developments and short, medium, and long-term sustainability planning.

At the conclusion of the workshop the facilitators and participants will review the findings of the breakout sessions and prioritize them. Following the workshop, the organizers will review the findings, along with the prioritization, in more detail to create an action plan. The action plan will be presented to the Next Generation Safeguards Initiative and the IAEA as a recommended roadmap for future work.

Appendix 1: *Technical Considerations for Software Sustainability:*

IAEA end-user requirements:

- Application-specific requirements
- Requirements for unattended vs. attended vs. remote monitoring equipment
- Usability requirements
- Quality assurance requirements and guidelines

Technical Challenges:

- Software and data authentication
- Verification
- The user is part of the checking process for attended use
- Control inputs/outputs for unattended and remote use
- Quality assurance
- Common language, e.g., C/C++/C#, FORTRAN
- Defining a standard, e.g., documentation, readability, coding conventions
- Performing vulnerability assessments (less expensive at the design phase)
- Qualification
- Data interface, i.e., inputs and outputs

Administrative Challenges:

- Documentation
- Intellectual Property
- Version control
- Control of software modifications
- Access control
- User privileges
- Open source modifications
- Software quality assurance
- Training

Appendix 5: Report to the Workshop Participants

Report No. BNL-105966-2014

The U.S./IAEA Workshop on Software Sustainability for Safeguards Instrumentation

Louise G. Worrall, Chris A. Pickett, Oak Ridge National Laboratory
Susan E. Pepper, Katherine M. Bachner, Al Queirolo, Brookhaven National Laboratory

August 2014

DISCLAIMER

The U.S./IAEA Workshop on Software Sustainability for Safeguards Instrumentation

Louise G. Worrall, Chris A. Pickett, *Oak Ridge National Laboratory*
Susan E. Pepper, Katherine M. Bachner, Al Queirolo, *Brookhaven National Laboratory*

Workshop Objectives

The United States and the International Atomic Energy Agency (IAEA) convened a workshop on *Software Sustainability for Safeguards Instrumentation* in Vienna, Austria, May 6-8, 2014. The primary objective of the workshop was to assemble a cross-section of all safeguards instrumentation software stakeholders (i.e., users, developers, vendors, and sponsors) to identify strategies for ensuring that critical safeguards instrumentation software products continue to be available for use by the IAEA as required and that software functionality does not degrade over time. Safeguards instrumentation software must be sustained in a changing environment with increasing requirements and limited resources. The approaches taken in the past may not be the best model for the future and, therefore, the organizers wanted to evaluate these past approaches.

Workshop Highlights

Neil Chue Hong, Founding Director of the United Kingdom Software Sustainability Institute, presented the keynote talk on *Scientific Software: Sustainability, Skills and Sociology*. His presentation highlighted the fact that scientific software has a lifetime that is considerably longer than the lifetime of the associated computing hardware. Therefore, lifecycle planning models for software must anticipate changes in hardware approximately every 2-5 years. Software requires a significant overhaul approximately every 10 years. In his words, software "rots" over time, and therefore, simply doing nothing is not a viable approach for sustainability. For example, one common misconception is that the correct way to preserve source code is to keep it in a repository, but Mr. Chue Hong noted that even in a repository the software has to be maintained. Further, whether called the "bus factor" (Chue Hong) or "lottery factor" (Alexey Anichenko, IAEA), the number of software developers devoted to the sustainment of a key software product should always be greater than one. The points made in this keynote talk were revisited throughout the breakout sessions, and Mr. Chue Hong was quoted throughout the workshop.

The workshop provided the opportunity for external software developers to meet with IAEA staff developers and other external developers. For some external developers, this was the first time that they had met other external developers working on software for safeguards instrumentation. The workshop also provided the IAEA with the opportunity to promote their RAINSTORM project[32] and its benefits. One of the stated goals of the RAINSTORM project is to standardize remote data retrieval and data security for all future IAEA Safeguards Technical and Scientific Services (SGTS) equipment. This is an important goal that will lead to more uniform and shareable analysis software. Discussion of the On-

[32] RAINSTORM is the IAEA's user requirements for implementing a remote monitoring interface in new safeguards instrumentation designs.

Line Enrichment Monitor (OLEM) and associated software development highlighted the importance of early and iterative collaboration among stakeholders. The development of the Central RADAR[33] Inspection Support Package (CRISP) jointly by Euratom/DG-ENER and the IAEA was also highlighted by the IAEA as an exemplary model for sharing development effort and resources, and the resulting source code. The CRISP software package offers the promise of providing a way to integrate divergent data sources into a common format, which will enhance the ability of the IAEA to develop data analysis software that is more readily shareable.

Figure 1 is a graph prepared and presented at the workshop by Alain Lebrun, IAEA, to illustrate the status of the IAEA safeguards software that is used for portable non-destructive assay (NDA) instrumentation. The graph characterizes software according to whether the software is safeguards-specific (indicates there may be other user communities) and whether the software is owned by the IAEA or another party (indicates the level of access and/or responsibility the IAEA may have to the code for use and maintenance). The codes that are owned by the IAEA and are safeguards-specific are the codes for which the IAEA can take responsibility. The codes that are safeguards specific but are proprietary are of concern to the IAEA because the IAEA does not have the required access to the source code to perform reviews to ensure the software operates as intended or to make necessary modifications. This graph gave workshop participants a very useful framework for identifying critical safeguards software and could also be an important aid for future software sustainability planning.

Summary of Recommendations from the Workshop Breakout Sessions

The workshop was formatted with the delivery of informative presentations each morning and breakout sessions each afternoon. The workshop breakout sessions were structured around multiple relevant scenarios and case studies prepared with input from the IAEA, and time for expert discussions was provided. The resulting discussions among the participants led to numerous recommendations from the participants for improving the management of safeguards instrumentation software. A summary of the significant recommendations from the workshop is provided below.

It is important to the IAEA to have the in-house capability to address software sustainability issues. In particular, the IAEA wants the independence to be able to make minor modifications to software that do not warrant the time and expense associated with a typical member state support program (MSSP) task. In addition, the IAEA would like the flexibility to apply resources, including those available through the MSSPs, as appropriate. For example, in some cases hiring a cost free expert or a junior professional officer is more appropriate than contracting with a vendor, but not in others.

[33] Remote Acquisition of Data and Review

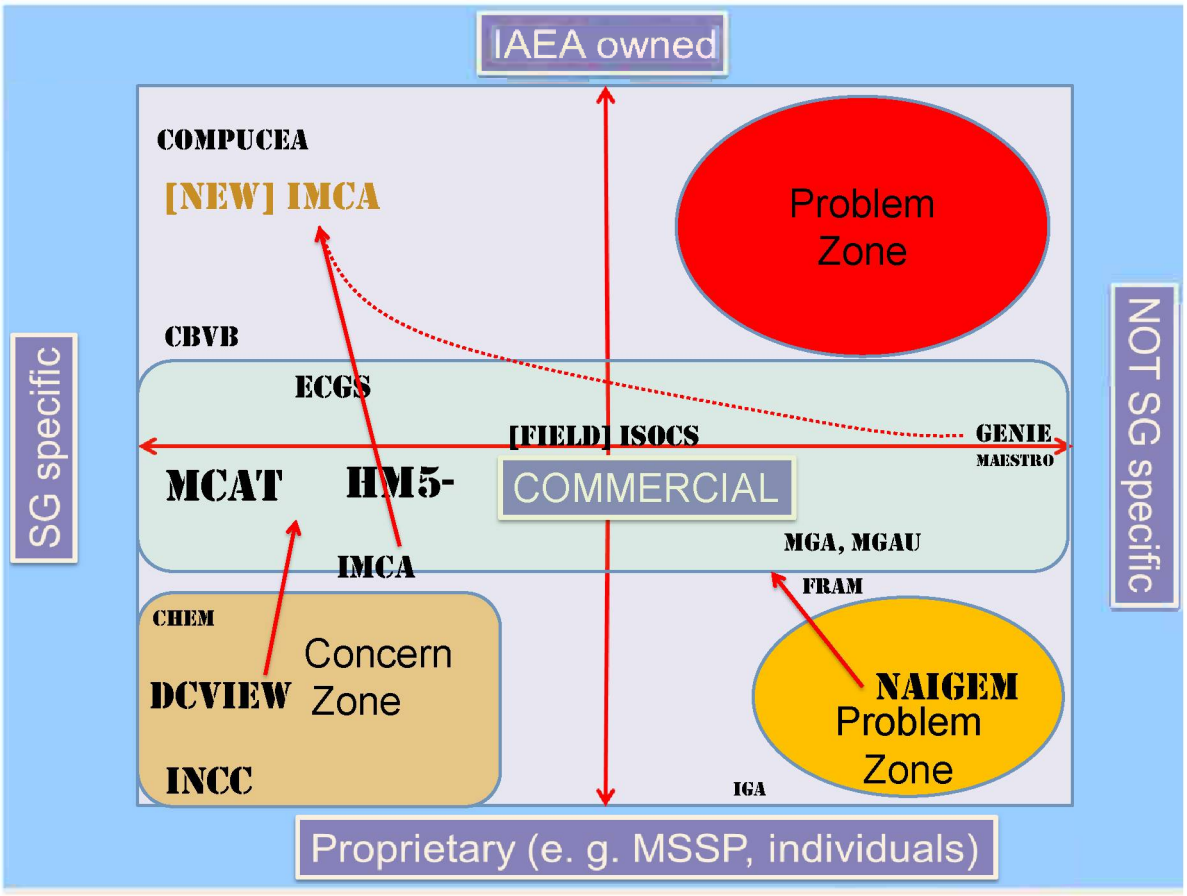

Figure 1: Characterization of the IAEA portable NDA software based on application and ownership

It is widely recognized that the mission to sustain software is a broad and ongoing challenge that encompasses legacy codes and codes that are not yet written, multiple uses and applications, and multiple stakeholders; therefore, there is no single "one size fits all" solution. A key finding of the workshop was the need to develop lifecycle plans for critical safeguards software. For lifecycle planning, the IAEA must create an inventory of current safeguards instrumentation software. Workshop participants recognized that sustainability does not just mean keeping software in use, but it also means knowing when to take certain software out of service or when it is best to re-write or replace the code (e.g., in the case of legacy software). This recognition led to the recommendation that a code audit be conducted to identify the software packages required by the IAEA to support safeguards instrumentation, their relative prioritization, the users and level of use of these codes (including the user community external to safeguards), the maintenance requirements and who is responsible for maintenance, the current cost of maintenance (i.e., capture the cost data), the availability of developers to work on these codes, who owns these codes, and what needs to be done to sustain them. This code audit should also take into account and capture dependencies between MSSPs. This inventory will promote efficient investment in safeguards software by identifying critical software packages and maintenance needs. It will facilitate a gap analysis and will become the basis for software management and lifecycle planning. It is widely recognized that sustainability will require funding, but allocations should be targeted to those codes that are both in demand and of high priority to the IAEA. A

consolidation and prioritized assessment of the portfolio of codes requiring ongoing support and maintenance resulting from the code audit and periodic assessment of new options could also increase the ability of all stakeholders to sustain them. Proper software archiving methods were also discussed by the keynote speaker and should be considered during the audit.

Human resources are a key consideration of software sustainability and sustainability planning. Stakeholders need to be committed and involved in order to successfully tackle the software sustainability challenge. In simple terms, people must be motivated to sustain safeguards instrumentation software and have good reasons or incentives to do so. Software sustainability and maintenance culture must be an integral part of institutional culture and become a routine way of doing business. It was recommended that a "user champion" initiate, lead, and become the proponent for the code audit and sustaining critical software. Code-focused user groups or working groups were also recommended to "socialize" the code, share best practices, and improve knowledge management. Establishment of these groups is a best practice because the groups increase knowledge and understanding of codes, engage next generation professionals, and thereby enlarge the user community. A user or working group need not be expensive or require government or extra budgetary funding. The workshop demonstrated that significant interest and motivation exist among the stakeholders and that a user or working group(s) for safeguards instrumentation software could be formed with minimal encouragement by the USSP or other sponsor.

When codes are in use and there is no immediate plan for upgrades, the subject matter experts (SMEs) and programmers may be reassigned to other tasks and may not be available to address even minor unplanned modifications. It is necessary for stakeholders to devise a plan for ensuring that these experts are available when needed. Applying software development best practices reduces the risks associated with a "bus factor" of one, i.e., a sole developer, and protects users against the unavailability of the SMEs and programmers. A well-structured and documented computer code with a sole developer could, if necessary, be assumed by a competent programmer immediately. There are a number of widely-used, open-source codes that are good examples of this principle.

The participants encouraged investment in sustaining critical safeguards software and supporting associated training. Funding could come from a single "resource champion" or a number of "resource champions." Options for software sustainability will vary depending on the owner of the codes, but may include planning and providing for a maintenance budget over the lifetime of the software, using umbrella tasks[34] for maintenance, and negotiating technical support contract arrangements with vendors. Another model is the Radiation Safety Information Computational Center (RSICC) system (https://rsicc.ornl.gov), which provides and manages licenses and leverages multiple programmatic support vehicles along with limited user fees to cover the costs associated with software sustainability. Each of these options would ensure that funds are available to support maintenance activities in the timely manner desired by the IAEA. Improved lifecycle planning and a proactive approach to project management would help to ensure maintenance support over the entire software lifecycle. Lifecycle

[34] Umbrella tasks are MSSP activities that consolidate a number of small, related activities.

planning should, therefore, take into account the "total cost of ownership" for each software product akin to how vendors support key software products.

Timeliness of support from MSSPs was identified as an area for improvement. Recognizing that the IAEA and MSSP processes ensure efficient and effective use of limited financial resources, the approval processes within both the IAEA and the MSSPs can result in delayed access to technical support from the MSSPs.

Discussions regarding intellectual property (IP) led to a recommendation to assess licensing possibilities. The stakeholders need to understand who owns the IP for each of the safeguards software packages and whether the packages can be shared. While some software codes may not be made available to the IAEA, there may be ways to creatively license the software to meet the needs of the IAEA while addressing the concerns of all stakeholders, including those who own the various pieces of IP. Some IP issues could also be mitigated by determining at the start of development who will hold the software IP at the end of development. Again, this dialogue should happen early in the development process and should become a routine part of any development project.

The IAEA believes the noncommercial nature and the small market impact of IAEA activities obviate or lessen the need for IP protection, and the need for IP protection on safeguards-specific software is not justified (see Figure 1). IAEA representatives proposed the concepts of non-exclusive licenses for noncommercial use and partial IP sharing, which would protect proprietary algorithms while open-sourcing architecture and interfaces.

A software escrow can simplify IP issues when agreed to in the planning phase of a software development project. A software escrow is a legal contract which gives the client access to the software developer's source code and other proprietary materials if the developer becomes incapable of supporting the software. A neutral third party serves as the escrow agent and provides such services as checking that deposited assets are readable and virus free, confirming that decryption keys for encrypted files are on deposit, providing a complete audit and inventory of your deposit, validating that the development environment can be recreated, testing the functionality of the compiled deposit materials, and confirming functionality of released software.

A phased approach to software development could mitigate some of the challenges, such as lengthy development times (interim software products could be implemented earlier) or a product that does not match the IAEA needs (there would be chances to review the project and make corrections at midpoints in the development). Active participation by the IAEA in software development projects should also be part of the phased approach. Software requirements and applicable standards should be defined at the beginning of the project to avoid changes in scope.

The IAEA, as the end-user, must be an active participant in the software development process. It is not acceptable for the IAEA to contract with a developer and remain uninvolved during development. The IAEA must also be actively involved in developments where the contract is between an MSSP and the

developer. Similarly, there should be IAEA champions to promote sustainability of the different instrumentation software programs. This is a challenge due to the IAEA's "rotation policy," which results in many professional staff members leaving the IAEA after seven years. Thus, there should be an institutional commitment to software to ensure that software sustainability can span the rotation of the sustainability champions.

Other recommendations encouraged better software documentation and more complete documentation of software algorithms, which would address a variety of problems, including knowledge management and the ability for software to survive unavailability of the software developer. It was recommended that teams of SMEs and software developers consult with technical writers to produce high quality documentation. In particular, the IAEA could prepare software requirements to document the required functionality for vendors to use in preparing software. The requirements can be updated as new measurement approaches emerge. This approach addresses both the "rot" problem and the IAEA's desire to have source code and allows the IAEA to define the requirements for the software without having to own it. A system that does not meet the requirements would not be saleable to the IAEA.

Innovative and promising approaches, such as the CRISP joint development and the OLEM instrumentation project, should be benchmarked. It was recommended that more experience should be gleaned from development partnerships or the use of RAINSTORM. Furthermore, success indicators or metrics of future software development projects should be clearly defined for future projects. The safeguards community should learn from other scientific communities that have previously faced and addressed the software sustainability issue.

Specific technical recommendations include the use or improved implementation of modular programming methods, which was regarded by the participants as an essential component of programming. Modules of safeguards software would include data acquisition, data management, and data analysis. This would keep the functional elements, which may be proprietary, separate from the interfaces, which may be customized for the IAEA's use, and facilitate desired access to the code for the IAEA. Standardization of software features, such as basic modules and input/output formats, was also recommended for the future.

Workshop participants agreed that the IAEA should seek feedback on RAINSTORM. The IAEA has implemented RAINSTORM in several systems including the Universal NDA Data Acquisition Platform (UNAP), the Laser Mapping System for Containment Verification (LMCV), the Next Generation Autonomous Data Acquisition Module (NGAM), OLEM, and other instruments and sees it as a standard for the future. However, because workshop participants were not widely familiar with RAINSTORM prior to the workshop and were only familiar with it through its application in OLEM, they recommended more review. This recommendation supports the establishment of a user group that includes people who are knowledgeable in all aspects of developing and sustaining software. While RAINSTORM is not yet a standard, the workshop participants applauded the IAEA's initiative in developing this product

which will one day serve that role and encouraged the community to develop other such standards and associated requirements.

Acknowledgements

This project was funded by the National Nuclear Security Administration's Next Generation Safeguards Initiative's Safeguards Technology subelement. The Department of State's High Priority Safeguards Program provided funding to cover the expenses of many of the private sector attendees. The workshop team would like to give special acknowledgement and thanks to Jim Regula (IAEA) who worked closely with the workshop team to plan for, develop, and make arrangements for the workshop, Emil Farkas (IAEA) and Chris Orton (NNSA) for facilitating workshop breakout sessions, David Peranteau (IAEA) and Hilary Lane (NNSA) for taking notes in the breakout sessions, Inna Cherkasskaya (IAEA) for taking care of many of the logistical aspects in Vienna, Barbara Hoffheins and Ben Deering (U.S. Mission to International Organizations in Vienna) for providing on-site assistance in Vienna, and Laura MacArthur and Michele Rabatin (BNL) for providing administrative assistance to the workshop team. Finally, the workshop team appreciates the participation and contributions of the many software, hardware, and international safeguards experts who attended the workshop.