

Open Software Solutions for the Government

*Prepared for the Department of Homeland Security
Homeland Open Security Technology program
BAA Number: N66001-08-X-2080 C41SR*

*Mark Lucas
Open Source Software Institute
29 September 2010*

Introduction

The government acquisition process was established to develop and deploy systems that fulfill unique requirements when a commercial marketplace has not fully evolved. Examples include military hardware and the space program. Unique government requirements are addressed with rules and practices that mitigate risk while imposing processes and requirements for governance of these projects.

As technology and the marketplace evolves, risk diminishes and the emphasis shifts towards commercial solutions. Today, computer software practices are being led by the commercial world where far more resource is available. Adoption of open standards and open source software practices has accelerated the process even more. New business models have evolved that take advantage of these advances.

The government is best served by applying these commercial and open solutions to efficiently meet its needs. Understanding the advantages of these practices and how to work within the government infrastructure will lead to more efficient solutions.

There is also a wide variance within government agencies on policies and procedures. The operational requirements of the military and intelligence communities tend to be more stringent and are used as examples. Establishing processes and procedures that satisfy those requirements will typically satisfy the policies established by other government agencies.

The Department of Homeland Security (DHS) was established to better coordinate the activities of many agencies, provide new efficiencies, and better protect the country. Open source software and approaches holds the promise of more efficient technology acquisition, interoperability, and standardization – which will directly result in a more secure nation. To that end, the DHS is leading a number of policy and technology efforts under the HOST program.

Open source software solutions potentially address many challenges that face DHS managers – interoperability across diverse agencies and the ability to scale and distribute software solutions without incurring licensing fees are definite advantages. Approximately 70% of the jurisdictions in the United States have populations of less than 5000 - typically they don't have the resources to invest in expensive information technology systems. Open source solutions could easily be extended to these jurisdictions for reporting and interoperability.

Homeland Open Security Technology (HOST)¹

The Homeland Open Security Technology (HOST) program is designed to help federal, state and local government agencies realize technical, security and economic benefits made available through open source and open technology solutions. To facilitate the adoption of Open Technology Solutions, the HOST program addresses issues of IT governance, acquisition and deployment policy, Information Assurance (IA) evaluation and security, collaborative development and availability of OTS resources.

¹ <http://www.cyber.st.dhs.gov/host.html>

Goals

This document provides guidance for software vendors and government program managers desiring to evaluate open technologies.

Software developers and solutions providers that follow an open source software development model and wish to market their products and services to government agencies. These providers fall into two broad categories:

- commercial solutions providers
- existing government contractors

Commercial solutions providers

This paper will provide a brief overview of how the government acquires software and solutions from commercial vendors. An overview of the government roles and responsibilities along with initial guidance on how to market and provide open source based solutions within government agencies is covered.

Existing government contractors

For existing government contractors, the paper will focus on recent initiatives and policy shifts that prefer an open source approach. The government is beginning to realize the benefits of leveraging open technology solutions. Educating government managers and administrators on the advantages of this approach will ease the transition providing enhanced value to government customers.

Navigating government infrastructure, policies, and bureaucracy can seem daunting to the uninitiated. Understanding the needs, processes, and concerns of government program managers, contracting officers, and security personnel will simplify the process of introducing new technologies and practices within the government.

For government managers this paper discusses how to manage open source software technologies and objectively evaluate open source projects.

The government supports a wide spectrum of programs, budgets, and processes from advanced concepts development to tightly configured trusted operations. Over the years, government policies have structured processes and reviews to manage risk based on the ultimate use of the solution. The requirements for research and development differ from critical operational environments.

Assessing the maturity of the new technology will help to identify the appropriate development phase to target.

As with any business endeavor, it eventually depends on building a productive working relationship with key personnel and providing the proper support while addressing customer needs. This paper will provide basic guidance and information that will assist the software provider and help provide government agencies with highly leveraged open solutions.

Outline for the paper

This document is divided into the following main sections:

- Understanding the customer
- Marketing and Business Development
- Development and Support
- Government policies and validations
- COTS versus GOTS
- Open Technology Evaluation Process
- Success stories
- Recommendations and Conclusions
- References

Understanding the customer

Establishing a positive relationship with the customer is key. Understanding the processes, policies, goals, requirements, and constraints that influence key government personnel is a necessary first step. Providing solutions for an agency requires additional process and criteria. Often, the government customer has to define a formal requirement, obtain a budget, and find an appropriate contract vehicle to acquire services or a solution. More often than not, this cycle is tied to the government fiscal year.

Fiscal Year is a term that is used to differentiate a budget or financial year from the calendar year. It is commonly abbreviated as FY.

The Federal Fiscal Year runs from October 1 of the prior year through September 30 of the next year. For example:

FY 2010 runs from October 2009 through September 2010.

FY 2011 is from October 2010 through September 2011.

FY 2012 is from October 2011 through September 2012.

This allows the new Congress (and anyone else newly elected) to participate in January, when they take office, to prepare the budget for the next year.

The government categorizes and budgets based on technological maturity and operational requirements. Research and development funds are maintained separately from operations and support. Rules and regulations are governed by

the Federal Acquisitions Regulation² (FAR). The FAR provides guidance how systems, products, and services are acquired by the federal government. A detailed review of the FAR is beyond the scope of this document.

² <https://www.acquisition.gov/far/>

Roles and responsibilities

Understanding the roles and responsibilities of the key government positions will increase the probability of success when introducing new approaches and solutions. The ability to efficiently address the concerns and challenges of key government personnel should be the focus of ongoing communication. To accomplish this, a quick overview of key positions is provided below:

- Program Manager
- Contracting Officer
- Contracting Officer's Technical Representative
- Security Officers

Program Manager (PM)

The Program Manager is in charge of the overall effort - which may involve many individual projects. Projects deliver outputs; programs create outcomes. On this view, a project might deliver a new functional capability, system or process. By combining these projects with other deliverables and changes, their programs might deliver functionality for a new mission. Typically, a software provider will not interface directly with the PM on a routine basis. A project manager, typically a COTR (see below), will be established for that purpose.

Government program managers need to evaluate the impact that locking into one set of proprietary standards or products may have, not just on the future ability to react and respond to adversaries, but to their own programs' resilience, robustness, and level of technical and management risk. For instance, what if a closed, proprietary component licensed by the contractor is phased out or no longer supported by a third party company? Using a component with open interfaces allows a program manager to mitigate this risk. There is no sure way to "future proof" a program, but open technologies (both open source and proprietary) increase the likelihood that a capability can sustain its value in the field. --OTD Field Manual

Contracting Officer – (CO)³

A **CO** is the only person who can obligate the government in a contract. Contracting officers have a complex set of rules and regulations they must comply with in the execution of their responsibilities. The CO typically focuses on contracting issues and relies on the Program Manager and the COTR (see below) for technical and functional guidance.

³ http://en.wikipedia.org/wiki/Contracting_Officer

Contracting Officer's Technical Representative - (COTR)⁴

A **COTR** is a liaison between the **United States government** and a **private contractor**. He or she ensures that their goals are mutually beneficial. The COTR is normally a federal or state employee who is responsible for recommending, authorizing (or denying) actions and expenditures for both standard delivery orders and task orders, and those that fall outside of the normal business practices of its supporting contractors and sub-contractors. Most COTRs have experience in the technical area (i.e., electronics, chemistry, public health, etc.) that is critical to the success of translating government requirements into technical requirements that can be included in government acquisition documents for potential contractor to bid and execute that work. A COTR is designated by a Contracting Officer. The CO has the actual authority to enter into, administer, and/or terminate contracts and make related determinations and findings. Other terms for COTR include Contracting Officer's Representative (COR) and Project Officer (PO). The terminology may be agency specific. Typically, your primary government interface will be the COTR.

Responsibilities

The Contracting Officer's Technical Representative is responsible for monitoring the contractor's progress in fulfilling the technical requirements specified in the contract. The COTR ensures that all required documentation and data are submitted in accordance with the procurement deliverable schedule. Should the contractor fail to fulfill the contractual requirements, the COTR must inform the contractor of such failure. The COTR informs the contracting officer of any technical or contractual problems or delays. The COTR maintains administration records, approves invoices and performs final inspection and acceptance of work performed under the contract.

There are limits to the authority delegated to the COTR. The COTR is not authorized to make any commitments or obligations on behalf of the **government**. Therefore, the COTR can be thought of as a Project Manager. The advantages of an open approach for COTRs include:

- cost effective introduction of new technologies
- increased collaboration between team members
- more effective problem solving across contractor interfaces

⁴ <http://en.wikipedia.org/wiki/COTR>

Security Officers

Most government agencies will have formal positions established for physical and computer security. Typically, there will be a need to interface with security organization to introduce new software solutions into government facilities and programs.

Open technology development methodologies are important to the National Security and National Interest of the U.S. for the following reasons:

- *Enhanced agility of the defense industry to more rapidly adapt and change to user needed capabilities.*
- *Strengthen the industrial base by increasing competition. Makes industry more likely to compete on ideas and execution versus product lock-in.*
- *Enables DoD to secure the infrastructure and increase security by understanding what is actually in the source code of software installed in DoD networks.*
- *Rapidly response to adversary actions as well as the rapid and continual change in technology .*

The national security implications of open technology development (OTD) are clear: increased technological agility for warfighters, more robust and competitive options for program managers, and higher levels of accountability in the defense industrial base. Open technology design and development methodologies increase the speed at which military capabilities are delivered to the warfighter, and accelerate the development of new, adaptive capabilities that leverage DoD's massive investments in software infrastructure. OTD Field Manual, August 2010

Understanding the responsibilities and concerns of these positions is key for providers of open solutions. It is often necessary to educate and inform these officers on the configuration management, testing, and processes that provide reliability and security for open solutions.

Facility Security Officer (FSO)

The FSO is in charge of overall security associated with a physical location. This includes access control, handling of classified material, and storage. Typically, material will need to be coordinated and introduced through a pre-defined process - which often includes virus scanning in the case of computer software.

Information System Security Officer⁵ (ISSO)

The ISSO is responsible for ensuring that users comply with the INFOSEC program requirements and procedures. The ISSO will manage the process for approving new software functionality into a government facility. Depending on

⁵ <http://www.marcorsyscom.usmc.mil/sites/ia/references/don/NAVSO%20P5239-07%20ISSO%20Guide.pdf>

the background of the ISSO this may require discussion and education on open source software development practices and testing. If the ISSO is not already aware, it is helpful to provide pointers to existing open source policy and implementations within the government.

Marketing and Business Development

Overall Approach

The government can benefit by following an open technology approach. Over the last several years there have been a number of studies, papers and policy changes within the government that promote an open technology approach:

Mitre Whitepaper⁶

The Mitre white paper was considered a milestone for the introduction of open source techniques and software within the military.

This paper analyzes the business case of open source software. It helps Program Managers evaluate whether open source software and development methodologies are applicable to their technology programs. In the Executive Summary, the paper explains open source, describes its significance, compares open source to traditional commercial off-the-shelf (COTS) products, presents the military business case, shows the applicability of Linux to the military business case, analyzes the use of Linux, discusses anomalies, and provides considerations for military Program Managers. The paper also provides a history of Unix and Linux, presents a business case model, and analyzes the commercial business case of Linux.

Open Technology Development Roadmap⁷

Open Technology Development (OTD) is an ongoing effort to transition technology development practices towards Open Standards Interfaces, wider inclusion of Open Source Software, and Open Collaborative development techniques. OTD additionally refers to technology and business process mechanisms that will allow DoD - funded software code to be shared and collaboratively developed across DoD activities. This methodology would allow DoD organizations and industry to rapidly adapt and extend existing software capabilities in response to shifting threats and requirements without being locked in to a specific vendor or held hostage to closed proprietary standard.

As the concept advances, OTD could help ease the looming shortage of available software programmers, needed to meet DoD's ever increasing demands for the rapid generation of accurate, reliable software in a cost effective manner. OTD will determine the policy, procedural, and cultural changes needed

6 www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/kenwood_software.pdf

7 [Herz2006] <http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf>

within DoD to employ these industry proven strategies. Building on previous Open Source software studies, experiments, projects, and initiatives, this project will recommend changes in the process of technology acquisition to shift procurement from inefficient, opaque closed systems to open, market-based methodologies based on open standards, service-based architectures, open source collaboration and software reference implementations.

The OTD Roadmap⁸ provides an overview of open technologies with recommendations on how to apply these principles within the government. The Open Technology Development working group is currently completing an OTD field manual - sections of which are referenced in this document.

DoD CIO⁹ OSS Policy

This document clarifies policy regarding use and consideration of open source software within the Department of Defense. For contracting purposes, it should be considered equivalent to COTS. There are numerous advantages for the use of open source software and it should be the preferred approach in any acquisition. All software needs to be properly evaluated and maintained over its life cycle.

For commercial vendors

Commercial vendors that do not follow government accounting practices can best approach the government market place through two vehicles:

- Subcontracting through a government contractor
- Adding services and products to the GSA Schedule

Sub contracting through government integrating contractor

Government contractors are typically certified through the Defense Contract Audit Agency (DCAA). Commercial vendors that are not DCAA certified can provide commercial products and services through a DCAA certified prime contractor. This provides an effective vehicle for providing solutions to government agencies. For most commercial companies, the business model, accounting process, and DCAA requirements will be prohibitive for initial expansion into government markets – sub contracting through a DCAA certified integrating contractor will be a practical way to enter the government marketplace.

Government Services Administration (GSA) Schedule¹⁰

For software providers that are not familiar with working with the government, one of the best avenues is to use the GSA schedule. The GSA establishes long-

8 [Herz2006] <http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf>

9 <http://cio-nii.defense.gov/docs/OpenSourceInDoD.pdf>

10 <http://www.gsa.gov>

term government wide contracts that allow customers to acquire a vast array of supplies (products) and services directly from commercial suppliers.

To become a GSA Schedule contractor, a vendor must first submit an offer in response to the applicable GSA Schedule solicitation.

GSA awards contracts to responsible companies offering commercial items, at fair and reasonable prices, that fall within the generic descriptions in the GSA Schedule Solicitations. Contracting Officers determine whether prices are fair and reasonable by comparing the prices/discounts that a company offers the government with the prices/discounts that the company offers to commercial customers. This negotiation objective is commonly known as “most favored customer” pricing. In order to make this comparison, GSA requires candidates to furnish commercial price lists and disclose information regarding their pricing/discounting practices.

Consulting services for establishing services and products on the GSA schedule are widely available.

For existing government contractors

There are several existing government contractors that have realized that open source and open technology approaches can provide a competitive advantage within the government solutions world. At its core, open technology provides an effective means for technical collaboration and integration of highly leveraged solutions.

Professional Services approach

Professional services development and support of open source based solutions is a sustainable business model for existing government contractors. Contractors that manage open source repositories and development efforts can market their support to solve government requirements. OSSIM, Opticks, and Falconview are examples that are reviewed in this document.

Integrating contractor

For government contractors that act as integrators, the open technology approach provides an open framework for evolving technological solutions, providing more efficient life cycle evolution, and avoiding costly Operations and Maintenance licensing fees. Open approaches allow problems to be chased across contractor interfaces. Open technology development provides a mechanism to reduce switching costs as better solutions evolve.

Development and Support

Government officials need to mitigate risk when providing systems and solutions. Demonstrating robust technologies and processes for configuration management, testing, quality assurance, and security vulnerabilities will go a long way to alleviating concern as the open technology approach is explored.

Modern day software development practices focus on building security and risk management into the software development life cycle.¹¹ In the final analysis, the government contractor wants to mitigate risk for deployed solutions.

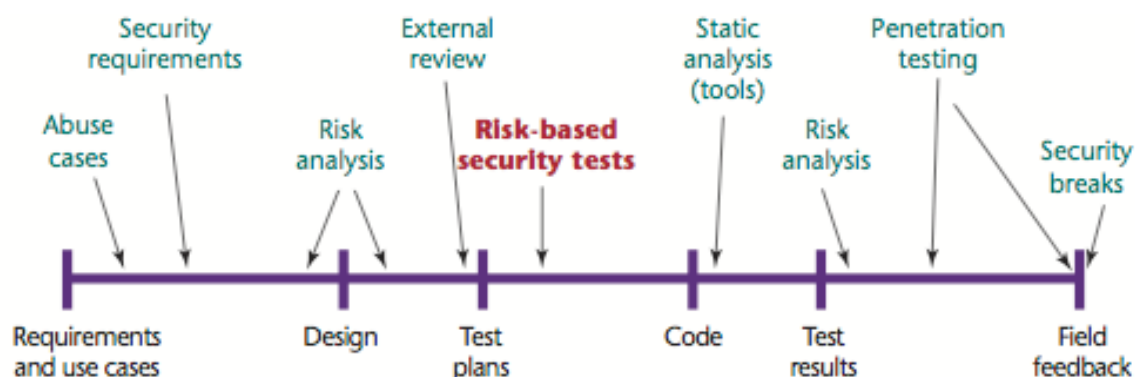


Figure 1 The software development life cycle.

Software security practitioners perform many different tasks to manage software security risks, including

- Creating security abuse/misuse cases;
- Listing normative security requirements;
- Performing architectural risk analysis;
- Building risk-based security test plans;
- Wielding static analysis test tools;
- Performing penetration testing to the final environment; and
- Cleaning up after security breaches

Open source software and open solutions do not guarantee a secure implementation, but it does have several advantages that are worth consideration:

- The inner workings of the program can be inspected and audited
- When problems are discovered, they can be readily fixed
- Project communities encourage coding standards through peer pressure
- Most open source software solutions rely on best of breed open libraries

11 "Building Security In", Bruce Porter and Gary McGraw, IEEE Security and Privacy

While the effect on security of open source software is still being debated in the security community, an increasing majority of prominent experts believe that it has great potential to be more secure.¹²

Government policies and validations

Recent changes in policy have validated and promoted open technology approaches. Open approaches are now the preferred solution. Understanding and referencing these policies will help to overcome most objections to those not familiar with the recent changes.

We will focus on the following categories:

- Standards
- Open Interfaces
- Open source software

Standards

The government must use “voluntary consensus standards” in their procurements. The White House’s Office of Budget & Management (OMB) lays out the policy for the federal government around standards:

What Is The Policy For Federal Use Of Standards?

All federal agencies must use voluntary consensus standards in lieu of government-unique standards in their procurement and regulatory activities, except where inconsistent with law or otherwise impractical. In these circumstances, the agency must submit a report describing the reason(s) for its use of government-unique standards in lieu of voluntary consensus standards to the Office of Management and Budget (OMB) through the National Institute of Standards and Technology (NIST).

OMB also lays out the case for why the government should use open standards:

What Are The Goals Of The Government In Using Voluntary Consensus Standards

Many voluntary consensus standards are appropriate or adaptable for the Government’s purposes. The use of such standards, whenever practicable and appropriate, is intended to achieve the following goals:

- Eliminate the cost to the Government of developing its own standards and decrease the cost of goods procured and the burden of complying with agency regulation.

12 [Wheeler2003] How to secure Linux and Unix systems

- Provide incentives and opportunities to establish standards that serve national needs.
- Encourage long-term growth for U.S. enterprises and promote efficiency and economic competition through harmonization of standards.
- Further the policy of reliance upon the private sector to supply Government needs for goods and services.

Federal regulation is very clear: the government must use voluntary community based standards (which this report equates to ‘open standards’) when acquiring systems. To recap, an open standards is one:

- That is freely available to use and;
- That requires no payment to anyone to use the standard

Standards in DoD

Within DoD, the office of OASD-NII sets policy for information systems and C4 (command, control, computers and communications) systems. There are a few key documents that provide guidance DoD for how to deal with IT standards. Among them is DODD 5101, which makes DISA the executive agent for IT standards:

Further policy directives (i.e., DOD Directive 5101.7¹²) have directed DoD to adopt non-Government standards as DoD standards. DISA has been identified as the executive agent in charge of adopting and pushing standards for DoD, which runs an index of acceptable DoD standards at <https://disonline.disa.mil>. The DoD IT Standards Registry (DISR) is only available to DoD and government personnel. One interesting feature of the policy that led rise to the DISR is a requirement for “contractors to identify instances where cost, schedule, or performance impacts may preclude the use of IT standards mandated in DISR,” the implication being that proprietary and closed standards are less that desirable.

To insure interoperability among systems, DODD 4630.05 requires contractors to use only approved standards from the DISR for their systems.

DODD 4630.05: Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS) May 5, 2004

5.8.5. Implement procedures to ensure the use and implementation of approved standards contained in the DoD Information Technology Standards Registry for programs under the DoD Components’ cognizance.

Open Interfaces¹³

In this context, *interfaces* refers to data formats, communication protocols and other interoperability interfaces of computer programs - with other computer programs or with hardware components.

An interface may be referred to as an *Open Interface* if it fulfills all of the following criteria:

Documentation:

The interface shall be documented carefully and completely.

Public specification:

The specification shall be published. Royalty-free copying and redistribution shall be allowed.

Strictness:

The specification shall be formulated precisely and sufficiently strictly, so that it ensures interoperability of conforming implementations within the scope of its stated objectives.

Technical quality:

The specification shall be technically correct, mature and stable.

Complete implementation:

There shall be at least two interoperable complete implementations, or one complete implementation of which the source code has been published.

Freedom of implementation:

There shall be no restrictions that prevent implementing the standard, under any model of software licensing, partially, fully, or with extensions. In particular, there must not be any requirement of royalty payments for any use of the standard. Restrictions against other models of software licensing are likewise unacceptable.

Openness of further development:

It shall be ensured that if further development of the standard is necessary, this can be done by means of a process in which all market participants are free to participate without restriction. It shall be ensured that proposed changes are accepted only if they do not unfairly disadvantage any group of market participants.

Loss-less migration:

If it is intended with the specification of a data format to replace a previous version, or another older standard, then the new specification shall define data formats that support representing all information that can be represented by the old data format. (The implementation of such features may be declared optional.)

¹³ Creative Commons attribution license: <http://siug.ch/ointerfaces/def-en>

Open source software

Managing and applying open source software projects for use by the government can present unique challenges. Successful open source software projects focus on building and maintaining an active community of interest. Government agencies sometimes have a tendency to take a snapshot of the baseline, take it inside, and modify internally. Unique government requirements, security considerations, and government contractors that only have daily access to government networks are often given as rationale for this approach. In the long run, this approach negates the advantages of an open source approach. A detailed summary of open source software and how to manage open source projects is beyond the scope of this document. Instead, this section will focus on the unique aspects and challenges of supporting open source software solutions within the government.

The following recommendations are for companies that provide service and support for open source solutions:

Strictly adhere to best open source development practices

Successful open source projects quickly migrate to best of breed tools and practices. As a result, there is a remarkable amount of uniformity between open source projects. Technical agility and evolution is one of the primary benefits of an open source approach.

Maintain a single repository

The primary resource to be managed in any open source project is the online community. Projects should always be located where the most resource is available. For most open source projects, this equates to the unclassified internet. Trying to manage multiple repositories and keep them synchronized is extremely difficult. When supporting multiple repositories, there needs to be an efficient mechanism for capturing internal enhancements and fixes so they can be contributed to the main repository. Unique government requirements should be addressed through plugin architectures as discussed below.

In the open source world, “forking” a project is considered a sign of failure. It unnecessarily divides the available resources and will quickly degrade to a divergence of software baselines. Commercial open source providers should treat the software solution as any other COTs delivery. A supported and tested baseline should be delivered as a fully configured solution. Proposed changes and enhancements to that baseline should be vectored back to the main repository.

Manage proprietary and classified interfaces through plugins

For those situations where classified, unique, or proprietary functionality needs to be accommodated, plugins should be used. Plugins are shared libraries that

integrate with the software solution at runtime. This mechanism allows segregated development from the main baseline.

Evaluate dependencies for maturity

Most solutions are built on external open source libraries. A documented evaluation process of how those libraries are selected and updated should be an integral part of the final documentation to the government. The evaluation criteria described in this paper is a good starting point. Many government agencies will require documentation on delivered dependencies to assess the overall maturity and security of the system.

Provide support, training, and management of the solution

In the final analysis, proper configuration management, testing, documentation, training and quality assurance needs to be provided through out the life cycle of the government solution. Most agencies will have a process for evaluating the support and maturity of acquired solutions. Proper documentation and processes combined with recent policy guidance should ease the transition to government operations.

Relevant Processes and Certifications

A brief summary of processes and certifications that are commonly encountered when validating software within the US government:

FISMA – Federal Information Security Management Act

The Federal Information Security Management Act of 2002 is a United States federal law enacted in 2002 as Title III of the E-Government Act of 2002 (Pub.L. 107-347, 116 Stat. 2899). The act recognized the importance of information security to the economic and national security interests of the United States. The act requires each federal agency to develop, document, and implement an agency-wide program to provide information security for the information and information systems that support the operations and assets of the agency, including those provided or managed by another agency, contractor, or other source.

FISMA has brought attention within the federal government to cybersecurity and explicitly emphasized a “risk-based policy for cost-effective security.” FISMA requires agency program officials, chief information officers, and inspectors general (IGs) to conduct annual reviews of the agency’s information security program and report the results to Office of Management and Budget (OMB). OMB uses this data to assist in its oversight responsibilities and to prepare this annual report to Congress on agency compliance with the act.

FISMA assigns specific responsibilities to federal agencies, the National Institute of Standards and Technology (NIST) and the Office of Management and Budget (OMB) in order to strengthen information system security. In particular, FISMA

requires the head of each agency to implement policies and procedures to cost-effectively reduce information technology security risks to an acceptable level.

According to FISMA, the term information security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide integrity, confidentiality and availability.

NIAP – National Information Assurance Partnership (NIAP)

The NIAP is a United States government initiative to meet the security testing needs of both information technology consumers and producers that is operated by the National Security Agency (NSA), and was originally a joint effort between NSA and the National Institute of Standards and Technology (NIST).

The long-term goal of NIAP is to help increase the level of trust consumers have in their information systems and networks through the use of cost-effective security testing, evaluation, and validation programs. In meeting this goal, NIAP seeks to:

- Promote the development and use of evaluated IT products and systems
- Champion the development and use of national and international standards for IT security

CC – Common Criteria

Common Criteria (CC) is an international set of guidelines and specifications developed for evaluating information security products, specifically to ensure they meet an agreed-upon security standard for government deployments. Common Criteria is more formally called “Common Criteria for Information Technology Security Evaluation.”

DIACAP – DoD Information Technology Security Certification and Accreditation Process

The Department of Defense Information Assurance Certification and Accreditation Process (DIACAP) is the standard DoD approach for ensuring that information systems operate at an acceptable level of risk. DIACAP standardizes and consolidates activities leading to the security certification and accreditation of Information Technology (IT) including automated information systems, networks, and sites in the DoD. The primary purpose of the DIACAP is to protect and secure information systems that make up the Defense Information Infrastructure (DII). The process is the same in all branches of the armed forces and all DoD agencies and applies to all unclassified and classified DoD IT systems that collect, store, transmit, or process information. The DIACAP consists of four phases - system definition, verification, validation, and post accreditation.

COTs versus GOTs

For government contracts, software is categorized in two broad categories - COTs and GOTs¹⁴, with different criteria for each:

- **COTS** (commercial off-the-shelf) product is one that is used “as-is.” COTS products are designed to be easily installed and to interoperate with existing system components. Almost all software bought by the average computer user fits into the COTS category: operating systems, office product suites, word processing, and e-mail programs are among the myriad examples. One of the major advantages of COTS software, which is mass-produced, is its relatively low cost.
- **GOTS** (government off-the-shelf) product is typically internally developed by the technical staff of a government agency. It is sometimes developed by an external entity, but with funding and specification from the agency. Because agencies can directly control all aspects of GOTS products, these have been historically preferred for government purposes.

Within the Department of Defense recent policy has established the open source software is to be considered as COTs for purposes of government acquisition¹⁵. Therefore, whether the software was built with proprietary or open source methods is irrelevant to the government acquisition process.

Open Technology Development Evaluation process

Note: The following are excerpts from “open source for the enterprise” by Dan Woods and Gautam Guliani, available from O-Reilly. The excerpts have been slightly modified to address the specific needs of the Open Technology Development effort. This evaluation process has been successfully used in the past to introduce open source software into operational environments.¹⁶

The government has requirements for life cycle support of its acquisition systems. The Open Technology Development process is chartered to assist in the transition to Open Technologies. This evaluation matrix will assist the assessment of the maturity and supportability of open source software technologies.

Measuring the maturity of any software, commercial or open source, is far more an art than a science. This evaluation will focus on the specific elements of an open source project, and those elements will serve as a guide to determining its maturity. Maturity is an indicator not only of age, but also of various dimensions of quality.

¹⁴ <http://fcw.com/articles/2009/10/27/dod-open-source-guidance.aspx>

¹⁵ DoD CIO policy clarification on Open Source Software [DoD CIO 2009]

¹⁶ [WoodsGuliani] “Open Source for the Enterprise”

The Open Source Maturity Model

The Open source Maturity model attempts to quantify the maturity of an open source product. This should help a project decide whether to adopt the product for long term use.

This model assumes the following:

- A functional specification of requirements exists
- A functional specification has been matched to a product functionality list, and a short list of products that match has been created
- The project uses the results of these two steps to determine which product from the short list is suitable for adoption. It should be that this evaluation process might be too resource intensive for a short term product but is a very wise investment for a long term one.

The plan assesses the maturity in three major criteria areas:

- Product
- Use
- Integration

Product criteria

Product criteria are specifics about the product itself. Since open source software (OSS) products are often under rapid development, with major advances made in a few weeks to a few months, we list momentum as a criterion to offset the age criterion. This helps us spot products that aren't mature enough today but are worthy of keeping an eye on.

Use criteria

Use criteria are specifics about what it takes to use the product from day to day, from the effort of initial installation and configuration to the work required for daily upkeep and support mechanisms available to help in tailoring the product the project's needs and fixing defects encountered.

Integration criteria

Integration criteria are specifics about what it takes to make the product work in the project environment. This is often overlooked during evaluations and its a critical factor to consider in order to succeed with open source software in the project. For each criteria we assign a score of 1, 2, or 3:

1 Immature

The product is lacking in several critical areas. It would be dangerous for a project to use it in a mission critical function. Some projects remain in this state until the first major release.

2 Reasonable mature

The product has a sufficiently long history of stable deployment, a loyal user base, and a bright future.

3 Very Mature

The product has a long and stable history, a broad and vibrant user community,

The elements of open source maturity

- Leadership and Culture
- Vitality of community
- Quality of end-user support
- Extent and scope of documentation
- Quality of packaging
- Momentum
- Quality of code and design
- Testing practices
- Integration with other products
- Support for standards
- Quality of the project site
- License type
- Potential for commercial conflicts
- Corporate support

Measuring the Maturity of Open Source

The following maturity matrix is derived from “open source for the enterprise” by Dan Woods and Gautam Guliani , O-reilly Press. The book is an excellent resource for integrating open technologies into the infrastructure.

Maturity Criteria	Immature	Reasonably Mature	Very Mature	Criteria Description
Age	< 6 months	6 months - 2 years	> 2 years	OSS Efforts that are just getting underway are risky for enterprises.
Multiple Supported Platforms	One Platform	Many related platforms	Multiple heterogeneous platforms	Products that work on Windows, Unix, and Mac OSX are most desirable

Maturity Criteria	Immature	Reasonably Mature	Very Mature	Criteria Description
Momentum	No release in last 6 months	< two releases in the past year	Regular releases	This is key to helping separate vital products from ones that are withering
Popularity	Unknown Product	Viable Alternative	Category Leader	Popular OSS products are well tested and therefore more mature. They are also likely to be interoperable with a large number of other products.
Design Quality	Monolithic Application	Multiple components	Well-defined API	This criterion is key in determining the effort required to extend and adapt the product.
Setup cost	Poorly documented install process; poor documentation; help available from developers	Well documented install process, reasonable documentation; help available from developers; help available in support forum	Well documented install process; install wizards/scripts available; reasonable documentation; help available from developers; help available in support forums; third party install services	Most products should require a setup effort of hours or days, not weeks or months.
Usage cost	Poor or non-existent documentation; help available only through direct contact with developers	User manuals available; help available in support forums	Third party training services available	This criterion is often overlooked when evaluating a product
End-user support	No forums or mailing lists	Some forums or mailing lists	Well-run forums and mailing lists with archives and search; third-party support options	User community (forums, mailing lists) and third party support are vital to a product's success
Modularity	Monolithic Structure; possible but hard to extend	Multiple modules; possible to extend	Multiple modules, well defined API. possible and easy to extend	
Collaboration with other products	Unknown	Known cases of integration	Lots of integration documented	

Maturity Criteria	Immature	Reasonably Mature	Very Mature	Criteria Description
Standards compliance	Unknown or proprietary	Outdated	Current industry standards	
Developer Support	No forums or mailing lists	Some forums or mailing lists	Well-run forums and mailing lists with archives and search; third-party support options	

Leadership and Culture

One of the most important factors in evaluating the maturity of an open source project is the quality of its leadership. Are they serious developers with a strong understanding of technology and the kinds of problems that the project needs to solve? What previous successes do they have to show for their work?

The first question to ask about a potential project is simply this: is the leadership identified? Unfortunately, a number of open source projects get started by individuals who hack together a piece of code that works, more or less, and then leave it there, unfinished, unpolished, and unworkable for any serious applications. Likewise, many government projects offer up “abandoned” code hoping that a community of interest will magically form around it and sustain it. Clearly, no enterprise wants to start depending on what is essentially an abandoned project, no matter how good its founders’ original intentions were or how hard they once might have worked to realize an initial vision. Like all software, open source evolves; it’s more a process than a static, deliverable object. It is practically a living thing that needs ongoing nourishment, encouragement, and care. And without leadership of some kind, an open source program will wither and eventually die.

One good indicator of serious, devoted, and informed leadership specific to the open source realm is the degree to which a project leader or other team member participates in the many forums that have sprung up to facilitate the open source movement. All of this shows a level of personal, emotional, and perhaps even financial investment in not only leading their specific projects, but also furthering the progress of the open source concept as a whole.

Vitality of community

There is a huge correlation between leadership and the vitality of the community. Leadership breeds a healthy culture that will spawn an active community. In such communities, everyone finds something useful to do, and one of the key indicators of this is a division of labor between the project’s developers and users. Evidence of this can include separate lists for users and developers, and

subprojects dedicated to productization activities such as creating easy to use installation packages or user documentation.

Healthy open source communities also tread a fine line between disciplining lazy behavior, such as using curt replies to direct people who have simple questions to locations where the answers can be easily found, and welcoming new people into the project.

The size of the project is a good indicator of a project's viability. How active are forums? How many downloads happen, and how often? How frequently is the project referred to on Google? What is the project's rating on SourceForge (www.sourceforge.net)

Quality of end user support

End user support is a key element of a project's maturity, and it's one of the most crucial elements in terms of saving time when using an open source project. Active forums, well maintained Frequently Asked Questions (FAQs), and documentation that are available through a search engine are generally the biggest time-saving feature of an open source project. Such mechanisms identify not only answers, but also members of the community who are fellow travelers, who are using the software in the same way you are. Contacting such community members directly can often be a huge timesaver if both sides of the conversation have something to offer each other.

One of the most compelling aspects of open source projects are the very public and free-ranging discussions that take place among the lead developers, far-flung contributors, and end users. These discussions serve a variety of purposes, not the least of which is providing general advice, tips, and specific answers to end users' problems.

Not surprisingly, the most popular open source applications and programs tend to have the most active forums-and the most seasoned and helpful experts participating in those forums. Under the best of circumstances, answers to a technical question might show up within minutes or hours of the original post. Overnight is more typical. Usually, the answer is entirely satisfactory. But on some list and boards, questions simply go unanswered.

Extent and scope of documentation

Another good clue about a project's work process and code quality is the quality of its documentation. Sloppy documentation can well reflect sloppy coding practices or, at the very least, an arrogance or even disrespect for the final user of the code. Perhaps it's too much to expect that every line of source code should have a comment attached to it, as some textbooks would subscribe. But it's not out of line to expect that the instructions for installing, running, and fine-tuning a piece of enterprise software be written in reasonably clear English. Moreover, the historical milestones in the code's development ought to be available for all to see. Transparency is a must.

Something else to look for in a project's documentation is a comprehensive user manual or reference guide. This manual should provide complete instructions for the installation of the software in question, and not just a simplistic README file. Particularly important, and too often completely missing, is information about how to configure the software. Configuring a new program can be enormously time consuming, even with good documentation at hand. But working without a manual clearly outlining common configuration scenarios can add significantly to the integration costs.

Quality of packaging

Some programs are available only as source code, thus requiring users to undertake the somewhat difficult task of compiling that code into binary, executable form. Other projects provide both source and binaries, and still others come with installers designed for specific operating systems.

Perhaps the highest state of evolution for an open source program is when someone has had time to write an installation package that can install the software easily on many different platforms and configurations. This usually is not fun work, and for a project to have attracted a community large enough for this to be accomplished is a sign of the project's maturity and strength. Sometimes, one of the senior developers will actually take the time to do this sort of work himself, and much earlier than it would have happened otherwise. This is not uncommon when the open source effort is fueled by a core team that is also interested in selling consulting services related to the project.

Momentum (or Frequency of Releases)

Like all software, open source tends to undergo constant updating. New features, new extensions, bug fixes, and other changes get worked into each new release. How often new releases of a program become available might help or hinder its effective use. Clearly, if updates are not released often enough, users will be forced to wait too long to use the latest code and won't be able to use the program as effectively as possible. And if an important bug is discovered, the fix for it ought to be made available as quickly as possible.

Veterans of the open source world know how frustrating it can be to discover in the archives of a project's mailing list that an important problem has been addressed but that the code is not yet available in the official release of the program. Until the official release is available, the only option users have is to undertake a new build of the program on their own, using the prerelease version of the software - a difficult undertaking that many developers wish to avoid.

On the other hand, there is such a thing as a too-frequent release schedule. Ideally, new releases should be put forth only when substantial additions and changes have been made to program, and not simply with every sprinkling of not-so-important changes.

The right release schedule depends largely on how stable and mature a project is. Many programs reach a stage where they are not receiving many updates and the rate of new releases slows down to one or two a year, if that. On the other hand, a lack of updates can be an indication of abandonment or stalled development activity, so look carefully at other clues on the project's web site. In the past, stalled open source projects have been resurrected as business conditions and related technologies changed. A less mature product that is in high demand and wide use can be updated as often as once a month.

Check out the release history, though, to see if the new releases are mainly significant or trivial. A well-managed release cycle indicates the presence of experienced technologists at work. Well managed can mean different things depending on the culture of the open source project.

Quality of code and design

Exceeding the importance of almost every other factor, of course, is the quality of the actual code that is produced for others to use. In contrast to commercial software, open source code is just that: open to direct inspection at the source-code level by anyone and everyone, including potential users, developers, and competing development teams. Few people have the knowledge or time, however, to evaluate the software's quality fully, by inspecting its source code line by line, module by module. But it is possible to glean some meaningful insight into the thinking that has shaped the code- into the mind or minds that have produced what you see.

The clues are primarily visual. Take the code layout, for example:

- Have the authors organized the code in a way that invites understanding, and that reveals at least some of the organization?
- Is the code modularized?
- How are the modules grouped together?
- Has a naming convention been rigorously adhered to?

Anyone with a modicum of experience in writing and working with enterprise-level software should be able to "read" this high-level structure and the labels being used and get at least a partial feel for the software's functions and the quality of its coding.

The deeper the level of inspection of a program, of course, the more fully its quality will be revealed. Many open source programs are the results of team efforts, with some modules, functions, and Java classes, for instance, of higher quality than others.

Quality of Architecture

The quality of an open source program's architecture is difficult to assess, but for obvious reasons, it tends to be an important measure of the code's maturity. By

architecture, we mean system components (such as classes in J2EE, C++, etc.), use of design patterns, Object oriented techniques, and naming conventions.

Testing practices

Some open source code comes with automated, built-in testing facilities as standard features. In particular, the developers of many mature Java projects have gotten the testing religion and have begun to make test routines a significant portion of the code they develop and make available for public use. As much as 70% of some code is devoted to this kind of testing, which can be especially important when dealing with mission critical requirements. The presence of unit tests is a key indicator of good design.

Integration with other products

All software in a project operates as part of what developers often refer to as an ecology, meaning that a set of interdependencies cause programs to call on each other, vie for shared resources, and exchange transactions and information. As users evolve their computing setups, they might swap out one program for another or install a new system, and such changes can disrupt the ecological balance, as it were, by altering or ignoring certain dependencies between previous sets of applications and subsystems.

Frequently, open source projects are aware of such dependencies and new releases are proactively tested with other open source software. Pay attention to and take advantage of such nests of compatibility.

Support for standards

Closely related to the question of dependencies is the need for programs to use standards-based APIs. For various reasons, open source projects tend to adopt open standards where possible to increase interoperability. Be aware of solutions that advertise themselves as standards “aware” vs standards “compliant”. Many dominant proprietary solutions will support open standards interfaces for ingest but aren’t as cooperative on output.

Quality of the project site

Open source websites are usually masters of brevity and clarity when it comes to site design. That said, some are more concise and well organized than others, and this can matter a lot when a large team with different skills and relationships to the open source project are involved in a deployment. A great site can make it easy for everyone to educate themselves and find what they need.

License type

Open source projects employ a variety of different licenses. In some cases, there may be no restriction on the use of the software for developing that is based on a certain open source program. But when it comes time to distribute those applications and the underlying program, or make the applications

available through a public website, the license will need to be understood and followed.

One of the great attractions of open source is that users get direct access to the program's source code. This can be of great comfort when problems arise. One of the worst disasters is to hit a mission critical requirement only to find that the code's supplier has gone out of business. When this happens, the mission's entire investment is put at risk. With an open source program, users can get at the source code and change it if it need be. This is a vital form of insurance, with advantages that many corporations and agencies are now waking up to. Not every organization will bother to modify open source code, but just knowing that it is an option can be a major comfort.

Assessing Maturity and Support Infrastructure

Metrics for Software Solutions

Evaluating the maturity level of software components in an objective fashion will provide supporting documentation for the approval process. As a matter of course, an objective evaluation process should be used in the design and integration of any open source software solution. There are literally hundreds of thousands of OSS projects available. In most functional areas there will be dozens of potential candidates to address functional requirements. The maturity of the development community around these projects is a key indicator of the functional performance, maturity, and level of support available.

Technical Readiness Levels – (TRL)

The government has established its own rating system for assessing the maturity of software solutions. The TRL grading system helps government program and technical managers grade software for potential use. Combining this rating system with OSS based metrics is highly recommended.

<i>TRL</i>	<i>Criteria</i>	<i>Detail</i>
1	<i>Basic principles observed and reported</i>	Transition from scientific research to applied research. Essential characteristics and behaviors of systems and architectures. Descriptive tools are mathematical formulations or algorithms.

TRL	Criteria	Detail
2	<i>Technology concept and/or application formulated</i>	Applied research. Theory and scientific principles are focused on specific application area to define the concept. Characteristics of the application are described. Analytical tools are developed for simulation or analysis of the application.
3	<i>Analytical and experimental critical function and/or characteristic proof-of-concept</i>	Proof of concept validation. Active Research and Development (R&D) is initiated with analytical and laboratory studies. Demonstration of technical feasibility using breadboard or brassboard implementations that are exercised with representative data.
4	<i>Component, subsystem validation in laboratory environment</i>	Standalone prototyping implementation and test. Integration of technology elements. Experiments with full-scale problems or data sets.
5	<i>System, subsystem, component validation in relevant environment</i>	Thorough testing of prototyping in representative environment. Basic technology elements integrated with reasonably realistic supporting elements. Prototyping implementations conform to target environment and interfaces.
6	<i>System/subsystem model or prototyping demonstration in a relevant end-to-end environment (ground or space)</i>	Prototyping implementations on full-scale realistic problems. Partially integrated with existing systems. Limited documentation available. Engineering feasibility fully demonstrated in actual system application.

TRL	Criteria	Detail
7	System prototyping demonstration in an operational environment (ground or space)	System prototyping demonstration in operational environment. System is at or near scale of the operational system, with most functions available for demonstration and test. Well integrated with collateral and ancillary systems. Limited documentation available.
8	Actual system completed and “mission qualified” through test and demonstration in an operational environment (ground or space)	End of system development. Fully integrated with operational hardware and software systems. Most user documentation, training documentation, and maintenance documentation completed. All functionality tested in simulated and operational scenarios. Verification and Validation (V&V) completed.
9	Actual system “mission proven” through successful mission operations (ground or space)	Fully integrated with operational hardware/software systems. Actual system has been thoroughly demonstrated and tested in its operational environment. All documentation completed. Successful operational experience. Sustaining engineering support in place.

Foundations and Incubation Guidelines

There are a number of foundations that promote 'best practices' in the development of open source capability. The Apache foundation and the OSGeo Foundation¹⁷ are just two examples. These foundations run formal incubation evaluations for projects that wish to join.

The guidelines for OSGeo incubation are shown below¹⁸:

Purpose

The purpose of the OSGeo incubation process is to ensure that projects :

- have a successfully operating open and collaborative development community
- have clear IP oversight of the code base of the project
- adopt the OSGeo principles and operating principles
- are mentored through the incubation process

Principles of OSGeo Projects

- Projects should manage themselves, striving for consensus and encouraging participation from all contributors - from beginning users to advanced developers.
- Contributors are the scarce resource and successful projects court and encourage them.
- Projects are encouraged to adopt open standards and collaborate with other OSGeo projects.
- Projects are responsible for reviewing and controlling their code bases to insure the integrity of the open source baselines.

Operating Principles

- Projects should document how they manage themselves.
- Projects should maintain developer and user documentation.
- Projects should maintain a source code management system.
- Projects should maintain an issue tracking system.
- Projects should maintain project mailing lists.
- Projects should actively promote their participation in OSGeo.
- Projects are encouraged to adopt OSGeo look and feel, branding, logos on their project sites.
- Projects are encouraged to participate in OSGeo standardization efforts to present a common interface for OSGeo visitors and members.
- Projects should have automated build and smoke test systems.

Success Stories

The adoption of open source software solutions is commonplace in the commercial world. The commercial world is leading the information technology and software development world. Government programs typically lag the commercial world in the adoption of these practices. Within the government there is increasing pressure and management support for reducing and eliminating this lag. In many instances, government acquisition regulations are a barrier to rapid adoption of these new practices and procedures.

*OSSIM*¹⁹

OSSIM - a contrived acronym for Open Source Software Image Map, pronounced “awesome”.

OSSIM is a high performance software geospatial system for remote sensing, image processing, geographical information manipulation and photogrammetry. The project has been successfully supported and deployed by US government intelligence and defense agencies, since 1996.

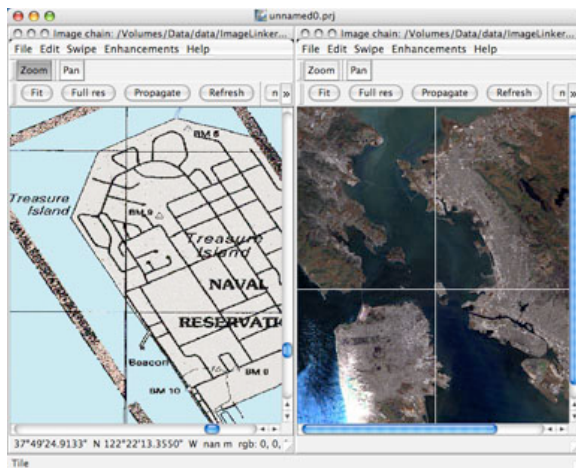


Figure 1 OSSIM Viewer - ImageLinker

The OSSIM project provides an interesting example of how an open source software project is influenced by US Government funding. Current developers are funded in part by the Government to develop needed capabilities, which in turn influences the direction of the software baseline.

Even though the core contributors are funded by government programs – the project is controlled and maintained on the unclassified internet. Classified or unique requirements are segmented through a plugin architecture. This approach allows OSSIM to maintain a worldwide developer community and

19 <http://www.ossim.org>

develop at internet speeds. The project has a very tightly controlled software baseline with a project steering committee governing the contents of each release. Contributors come from multiple companies. OSSIM is one of the founding projects in the Open Source Geospatial Foundation.

Large Data JCTD

The Large Data Joint Capabilities Technology Development (JCTD) project demonstrated a large scale project that integrated state of the art technologies with an open technology approach. Applying the latest technologies in advanced storage, networking, image processing and visualization - the project demonstrated global real time access to national data archives, imagery, and full motion video. Open source software and technologies were used at all layers of the network, storage, processing, and visualization stack. The project demonstrated the advantages of this approach and has gone on to operational transition. A number of open source software technologies were employed at all levels of the infrastructure:

- Lustre Global File System
- Suse operating system
- Open source tools and databases
- Open source infiniband drivers
- OSSIM, OMAR, and ossimPlanet data management and visualization

The Large Data JCTD is particularly significant in that it addresses the need for global geospatial collaboration. ossimPlanet provides an open geospatial visualization framework that other applications and systems can interface with. The scientific accuracy of the system enables newly collected data from military and commercial sensors to be rapidly placed in three dimensions on the virtual globe. Live interfaces alert decision makers to new developments and the status of global or regional events. Ongoing development will continue to focus on collaboration mechanisms and interfaces to live data - allowing decision makers to intuitively understand the latest information in a distributed virtual environment.

Opticks

Ball Aerospace launched Opticks in 2007 as its first open source software project designed to enable detailed analysis of remote sensing data and complement strategy promoted by the Department of Defense's Open Technology Development Roadmap. Opticks is used by scientists and analysts within the DoD community to analyze remote sensing data and produce actionable intelligence.

Opticks is an example of government software that was open sourced and released to the outside world.

Opticks supports Imagery, Motion Imagery, Synthetic Aperture Radar (SAR), and multi-spectral and hyper-spectral remote sensing data. Ball Aerospace expects Opticks to increase the demand for remote sensing data and broaden the features available in existing remote sensing software.

FalconView

FalconView²⁰ is a mapping system created by the Georgia Tech Research Institute for the Windows family of operating systems. It displays various types of

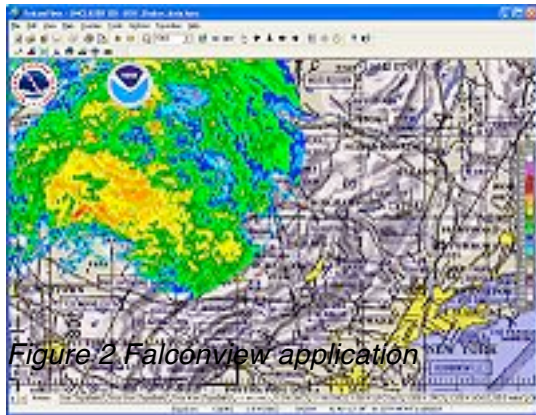


Figure 2 Falconview application

maps and geographically referenced overlays. Many types of maps are supported, but the primary ones of interest to most users are aeronautical charts, satellite images, and elevation maps. FalconView also supports a large number of overlay types that can be displayed over any map background. The current overlay set is targeted toward military mission planning users and is oriented towards aviators and aviation support personnel.

FalconView is an integral part of the Portable Flight Planning Software(PFPS). This software suite includes FalconView, Combat Flight Planning Software (CFPS), Combat Weapon Delivery Software (CWDS), Combat Air Drop Planning Software (CAPS) and several other software packages built by various software contractors.

In late 2008, Georgia Tech was funded to develop and deploy an open source software (OSS) version of FalconView. The OSS version of FalconView includes most of the functionality of the GOTS system, excluding only a few overlays considered to be exclusively related to military mission planning. Georgia Tech posted the first alpha version of FalconView as open source in June 2009.

OSCMIS

Open Source Corporate Management Information System (OSCMIS)

The Open Source Corporate Management Information System (OSCMIS) is a collection of web-based software tools and modules developed and deployed internally by the Defense Information Systems Agency (DISA) to manage the agency's human resources, security, training and personnel management activities. The application suite currently consists of approximately 110 different modules and is used by all of DISA's more than 16,000 users worldwide.

²⁰ <http://en.wikipedia.org/wiki/FalconView>

As the CMIS program was original work developed by DISA employees, the software code was available as public domain. In an effort to make the application suite and future code releases broadly available to all government and non-government entities, DISA collaborated with the non-profit Open Source Software Institute to release an open source version of the CMIS software. As part of the continuing collaborative agreement, DISA provides technical support through regular code updates and program releases to the new open source code base. In addition, DISA's IP legal council was actively involved in selecting an open source license which best addressed DISA's concerns and mission objectives for the release of the suite as an open source solution.

The OSCMIS suite is now available to government, academic, industry and private users under a DISA-approved open source software license (Open Software License v 3.0). There are no initial or recurring license fees or other mandatory fees or restrictive obligations associated with the open source program. As with any enterprise software application suite, adopters are responsible for common expenses associated with deployment, customization, support, maintenance, hosting and training. Several lessons have been learned from this effort. These include:

- 1) the government has identified a unique way to protect publicly-funded intellectual property by means of open source software licensing model
- 2) government IP legal counsel has reviewed the body of open source software licenses and identified existing licenses which meet and satisfy government concerns with regards to internal and external distribution of software code
- 3) and the government has identified a successful method of releasing code and contributing modifications and updates back to an open source software program community.

For additional information and updates on the OSCMIS program refer to:
<http://www.oss-institute.org>.

Recommendations and Conclusions

Leverage open development practices and software

The government no longer leads state of the art in software development. The resources and practices that are being deployed in the commercial and open source world are far greater and more advanced than within the government. Applying the latest practices, tools, and methodologies from the outside will dramatically improve government implementations and solutions.

Insist on open standards and interfaces

A rigorous implementation of open standards and interfaces will provide a level playing field for development and evolution of the solution over its life cycle. Competing solutions and new functionality can be integrated with low switching costs.

Take advantage of plugin frameworks

New functionality, proprietary or classified modules should be integrated through the use of software plugins. Modern programming environments - such as GRaILs - provide a framework for the rapid addition, integration, and removal of plugins.

Collaborate with existing software development communities

Resist the tendency to take a snapshot of an external capability and begin a separate development path within the government. Carefully question the validity of unique government requirements or any rationale that would 'fork' the software development. Divergence from the original development community is considered a failure in the open source community as it divides the available resource for moving the technology forward.

Security concerns and specialized requirements can be addressed through plugin architectures and routinely upgrading to the latest formal release of the software. Changes, fixes, and enhancements to the software baseline should be contributed back to the original project. This will avoid the need to repetitively patch these changes with every software upgrade. Effective evolution of the software baseline requires collaboration with the maintainers of the central repository.

References

[WoodsGuliani] “open source for the enterprise” by Dan Woods and Gautam Guliani , O-reilly Press

[Fogel2009] “Producing Open Source Software: How to Run a Successful Free Software Project” by Karl Fogel. 2009. <http://producingoss.com/>

[Wheeler2003] Secure Programming for Linux and Unix HOWTO by David A Wheeler <http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO.html>

[Kegel2004] “Contributing to Open Source Projects HOWTO” by Dan Kegel. 22 Nov 2004. <http://www.kegel.com/academy/opensource.html>

[Raymond2002] “Software Release Practice HOWTO” by Eric Raymond. 2002-01-04. <http://www.tldp.org/HOWTO/Software-Release-Practice-HOWTO/index.html>

[Wheeler2010] “Releasing Free/Libre/Open Source Software (FLOSS) for Source Installation” by David A. Wheeler. 2010-03-03.

<http://www.dwheeler.com/essays/releasing-floss-software.html>

[Berkus2010] “LCA: How to destroy your community” by Josh Berkus and Jonathan Corbet. January 18, 2010. <http://lwn.net/Articles/370157/>

[Jones2008] “How to get your code into an open source project” by Richard Jones. 2008-10-05. <http://people.redhat.com/~rjones/how-to-supply-code-to-open-source-projects/>

[DoD2009a] “Clarifying Guidance Regarding Open Source Software (OSS)” <http://cio-nii.defense.gov/sites/oss/2009OSS.pdf>

[DoD2009b] DoD Open Source Software (OSS) Frequently Asked Questions (FAQ). 2009. [http://cio-nii.defense.gov/sites/oss/Open_Source_Software_\(OSS\)_FAQ.htm](http://cio-nii.defense.gov/sites/oss/Open_Source_Software_(OSS)_FAQ.htm)

[http://cio-nii.defense.gov/sites/oss/Open_Source_Software_\(OSS\)_FAQ.htm](http://cio-nii.defense.gov/sites/oss/Open_Source_Software_(OSS)_FAQ.htm)

<http://www.oracle.com/us/industries/046045.pdf>

http://www.dwheeler.com/oss_fs_eval.html

<http://www.us-cert.gov/swa/>

<https://buildsecurityin.us-cert.gov/swa/>

<https://buildsecurityin.us-cert.gov/bsi/home.html>

http://en.wikipedia.org/wiki/Software_Assurance

Bibliography

[Herz2006] Open Technology Roadmap Plan by J.C. Herz, Mark Lucas, and John Scott. April 2006. <http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf>

OMB 2003, OMB M-04-16, Software Acquisition:
<http://www.whitehouse.gov/omb/memoranda/fy04/m04-16.html> The OMB policy recognizes that open source software is acceptable to use in software acquisitions.

DoD CIO 2003, DoD CIO Memo, May 28, 2003, Open Source Software (OSS) in the Department of Defense

DoN CIO 2005, June 5, 2007: Dept. of the Navy Open source Software Guidance: The DoD CIO office also released an initial 2003 memo accepting the use of open source software within DoD and further clarified its stance in a 2009 memo.

DoD CIO 2009, DoD CIO Memo, May 28, 2003, Open Source Software (OSS) in the Department of Defense

About the author

Mark Lucas has pioneered efforts in Open Source Software Development in remote sensing, image processing and geographical information systems. Mark established remotesensing.org and has led several government funded studies and development efforts since 1996. These efforts include the Open Source Software Image Map (OSSIM) projects for the National Reconnaissance Office (NRO), and the Open Source Prototype Research and Open Source extraordinary Program projects for National Geospatial-Intelligence Agency (NGA). OSSIM has been applied to disaster response activities including Katrina, Haiti, and the Gulf Oil spill. He is currently advising the Department of Defense on Open Technology Development policy. Mark has a BS in Electrical Engineering and Computer Science from the University of Arizona and a MS in Computer Science from West Coast University. He was a commissioned officer in the Air Force and assigned to the Secretary of the Air Force Special Projects organization. He has experience as both a government and contractor program manager through a number of classified programs. He is a founder and previous member of the Board of Directors of the Open Source Geo-spatial Foundation, He is a director for the Open Source Software Institute, and the National Center for Open Source Policy and Research. Mark is currently a principal scientist at RadiantBlue Technologies Inc.

He can be reached at: mlucas17@mac.com.