OPEN TECHNOLOGY DEVELOPMENT (OTD)

LESSONS LEARNED AND BEST PRACTICES FOR MILITARY SOFTWARE

Sponsored by the Assistant Secretary of Defense (Networks & Information Integration)
(NII) / DoD Chief Information Officer (CIO)
and the Under Secretary of Defense for Acquisition, Technology, and Logistics (AT&L)

Open Technology Development (OTD):

Lessons Learned & Best Practices for Military Software

2011-05-16

Sponsored by the Assistant Secretary of Defense (Networks & Information Integration) (NII) / DoD Chief Information Officer (CIO) and the Under Secretary of Defense for Acquisition, Technology, and Logistics (AT&L)

Portions of this document were originally published in the Software Tech News, Vol.14, No.1, January 2011. See https://softwaretechnews.thedacs.com/ for more information.

Version - 1.0

Acknowledgements

Development of this document was sponsored by Dan Risacher of the Assistant Secretary of Defense (Networks & Information Integration) (NII) / DoD Chief Information Officer (CIO), Fritz Schulz of the Under Secretary of Defense for Acquisition, Technology and Logistics (AT&L) Rapid Fielding Directorate, the AT&L Cross-cutting Studies Group and Heather Burke of SPAWAR Systems Center Atlantic (SSC LANT).

This document was authored by John Scott, David A. Wheeler, Mark Lucas, and J.C. Herz.

The authors would like to thank the following groups and individuals for their edits, inputs, help and guidance: John Weathersby, Kane McLean, Gunnar Hellekson, Josh Davis, Deb Bryant, Scott Goodwin, the MIL-OSS working group (http://mil-oss.org & http://groups.google.com/group/mil-oss) and many others.

Submitted By: _____
Name: John Scott, Contractor Date
Title: Sr. Systems Engineer & Open Technologies Lead, RadiantBlue Technologies, Inc.

Approved by: _____
Name: Mr. Fritz Schultz, U.S. Government Date
Title: Oversight Executive, OASD Research & Engineering, Rapid Field Directorate

Approved by: _____
Name: Mr. Dan Risacher, U.S. Government Date
Title: Associate Director, Office of the DoD CIO, Enterprise Services & Integration

Table of Contents

Chapter 1. Introduction

The purpose of this document is to help U.S. government personnel and contractors implement open technology development (OTD) for software within government projects, particularly in defense.

OTD is an approach to software/system development in which developers in different military, federal, commercial and possibly public organizations can collaboratively develop and maintain software or a system in a decentralized fashion. OTD depends on open standards and interfaces, open source software and designs, collaborative and distributed online tools, and technological agility. [OTD2006]

In April 2006, the Deputy Under Secretary of Defense for Advanced Systems and Concepts (AS&C) published the Open Technology Development Roadmap [OTD2006]. The OTD Roadmap Plan focused on steps and recommendations to implement these practices within the Department of Defense.

This document is divided into four chapters. The first briefly explains the context for OTD and why it is important to the U.S. military. Chapter 2 lays out the concrete steps for establishing, managing and distributing OTD projects within the government. Chapter 3 identifies programmatic procedures for OTD, including analyses of alternatives, the Request for Information/Request for Proposal (RFI/RFP) process, evaluating proposals, source selection, contracting language, and acceptance/approval criteria for deliverables. Chapter 4 deals with life-cycle management: transition, operations and maintenance, and leveraging a developer community for ongoing development.

1.1 Software is a Renewable Military Resource

> *"The United States cannot retreat behind a Maginot Line of firewalls or it will risk being overrun. Cyberwarfare is like maneuver warfare, in that speed and agility matter most."*
>
> *— William J. Lynn III. [Lynn2010]*

Software has become central to how the warfighter conducts missions. For reliance on software to be a strength, DoD must pursue an active strategy to manage its software portfolio and foster an internal culture of open interfaces, modularity and reuse [Scott2010]. In particular, DoD must have software that is easily adaptable to changing mission needs and can be evolved rapidly and delivered quickly at lower costs to meet mission requirements in a timely manner. This technological evolution entails a parallel evolution in acquisitions methodologies and corporate attitude to facilitate discovery, re-use, and modification of software across the DoD and U.S. Government.

Software is the fabric that enables modern planning, weapons and logistics systems to function. It might be the only infinitely renewable military resource. Capabilities evolve as new software is created anew or builds on existing software. From ground sensors to satellites, software is pervasive; it is the final expression of military knowledge transformed into source code and deployed on the battlefield.

We need an undated way of developing, deploying and updating software-intensive systems that will match the tempo and ever-changing mission demands of military operations.

Imagine if only the manufacturer of a rifle were allowed to clean, fix, modify or upgrade that rifle. The military often finds itself in this position with taxpayer funded, contractor developed software: one contractor with a monopoly on the knowledge of a military software system and control of the software source code. This is optimal only for the monopoly contractor, but creates inefficiencies and ineffectiveness for the government, reduction of opportunities for the industrial base, severely limits competition for new software upgrades, depletes resources that can be used to better effect and wastes taxpayer-provided funds.

To resolve these issues, a modern software intellectual property regime to broaden the defense industrial base needs to be defined that enables industry-wide access to defense knowledge, including software source code, documentation, hardware designs and interfaces. The result will be increased competition and a net lowering of the cost of innovation. Over time, the military would evolve common software architectures and industry-wide baselines to increase the adaptability, agility and, most importantly, capacity to meet new, dynamic threats quickly.

Key benefits of OTD are:

- Increased Agility/Flexibility: Because the government has unrestricted access and rights to the source code developed with taxpayer funds, that source code can be made discoverable and accessible to program managers, civil servants and contractors alike, increasing the potential of matching a need or requirement to an existing source code base that provides a large proportion of the solution that can be improved or enhanced to meet a new mission. Likewise, pre-existing government-funded components from different programs can be assembled without unnecessary costs and delays untangling intellectual property rights to determine what is and is not allowed. Instead of having to start from scratch to develop or enhance a capability, the government can re-use what it has already paid for and that works and draw from a broad base of developers and contractors who are familiar with the source code and component and can rapidly assemble, merge and modify existing systems and components with other existing source code.
- Faster delivery: Because developers only need to focus on changes to, and integration of, existing software capabilities instead of having to redevelop entire systems, they can significantly reduce the time to delivery for new capabilities. Even when a module or component is developed from scratch to replace an outdated one, such re-development benefits from open interfaces and standards that have a proven track record in the systems with which it interacts. Enabling cross-pollination of source code that is owned and paid for by taxpayer funds, development and deployment time can be significantly reduced.
- Increased Innovation: With access to source code for existing capabilities, developers and contractors can focus on innovation and the new requirements that are not yet met by the existing source code capabilities. This agility is particularly important because of a projected shortfall in the number of U.S. citizens with engineering and computer science degrees who will be clearable to work on military projects in the coming decades [National Academies 2008]. As a greater proportion of software engineering degrees are held by foreign nationals, and U.S. programmers are lured by innovative and lucrative work in the private sector, the military will face a long-term shortage of software engineers to work on military-specific systems. The Defense Department must therefore focus on the long-term challenge of generating higher levels of innovation out of a more limited pool of human talent and skill. It will be important to leverage that human capital by having engineers focus on the 10% of source code that actively improves a system without also being required to re-create the 90% of capability that already exists.
- Reduced Risk: creating new capabilities from scratch is riskier than re-using existing capabilities that are already proven and well understood. By re-using existing capabilities in the form of government-owned source code, interfaces and systems, developers can spend more time and resources on the riskiest parts of the implementation.
- Information Assurance & Security: One of the biggest values of open source development is enabling wider community access to software source. In this manner bugs become shallow and thus more easily found. Wider access to software source code also is key for forming and

maintaining a software security posture from being able to review software source code to seeing what is actually present within that software.

- Lower cost: The first cost to fall by the wayside with OTD is the monopoly rent the government pays to contractors who have built a wall of exclusivity around capabilities they've been paid by the government to develop. They may have internally developed source code (IRAD – internal research and development) that's valuable, but in an OTD system that code has been modularized so the government can make a rational decision about whether they want to re-license it for a new project or pay to develop a replacement. The entire value of the government's investment hasn't been voided by the mingling of IRAD into a government-funded system as a means of ensuring lock-in to a particular vendor. With unlimited rights and access to government-funded source code, the government can draw on a broader pool of competitive proposals for software updates and new capabilities that leverage current systems. The elimination of monopoly rent, combined with greater competition, will drive down costs and improve the quality of resulting deliverables, because any contractor who works on a system knows that they can be replaced by a competitor who has full access to the source code and documentation.

As Defense Secretary Robert Gates has said "The gusher [of money] has been turned off and will stay off for a good period of time." DoD needs a more efficient software development ecosystem – more innovation at lower cost. OTD squeezes financial waste out of the equation by reducing lock-in and increasing competition.

By implementing OTD, DoD could help make software code an infinitely renewable military resource.

1.2 What is Open Technology Development (OTD)

> *"In a real world of limited resources and skills, individuals and groups form, dissolve and reform their cooperative or competitive postures in a continuous struggle to remove or overcome physical and social environmental obstacles. Technological agility should be a metric."*
>
> *— Col John Boyd (USAF) [Boyd1976]*

As noted above, OTD is an approach to software/system development in which developers (outside government and military) collaboratively develop and maintain software or a system in a decentralized fashion. OTD depends on open standards and interfaces, open source software and designs, collaborative and distributed online tools, and technological agility. [OTD2006]

These practices are proven and in use in the commercial world. Open standards and interfaces allow systems and services to evolve in a shifting marketplace. Using, improving, and developing open source software minimizes redundant software engineering and enables agile development of systems. Collaborative and distributed online tools are now widely used for software development. The private sector also often strives to avoid being locked into a single vendor or technology and instead tries to keep its technological options open (e.g., by adhering to open standards). Previous studies have documented that open source software is currently used in many of DoD's critical applications and is now an inseparable part of military infrastructure [MITRE2003] [OTD2006].

OTD methodologies rely on the ability of a software community of interest to access software code or application interfaces across the enterprise. This access to source code, design documents and to other developers and end-users enables decentralized development of capabilities that leverage existing software assets. OTD methodologies have been used for open source development, open standards architectures, and the most recent generation of web-based collaborative technologies. The most

successful implementations come from direct interaction with the end-user community. The open source software development model is successful because communities of interest involve both developers and users.

OTD includes open source initiatives but is not limited to open source software (OSS) development and licensing regimes, which enforce redistribution of code. It is important, in the context of this report and resulting policy discussions, to distinguish between OSS and OTD, since the latter may include code whose distribution may be limited to DoD, and indeed may only be accessible on classified networks. Nor does the promotion of OTD within DoD impinge on the legal status of software developed with private sector money by commercial vendors.

In general, it is important to simplify use, modification, and distribution. If it takes a team of lawyers to determine if there are adequate rights to modify a program, it will not happen.

1.3 Off-the-shelf (OTS) Software Development Approaches, including Open Government OTS (OGOTS) and Open Source Software (OSS)

An OTD strategy allows organizations to develop and maintain software in a collaborative way. To maximize collaboration, software should be developed to use off-the-shelf (OTS) components and to be itself OTS to the maximum practical extent.

Off-the-shelf (OTS) software is simply software that is ready-made and available for use. The rationale for developing OTS software is to create software that can be used for multiple purposes, instead of using custom-built software for a single purpose and use. OTS software has the potential to save time, save money, increase quality, and increase innovation through resource pooling. Even when a custom system is needed, building it from many OTS components has many advantages.

There are many different ways that off-the-shelf (OTS) software can be maintained. Some OTS may be retained and maintained inside the U.S. government (e.g., because it is classified or export controlled); such software is termed government OTS (GOTS). Off-the-shelf items that are commercial items (e.g., by being sold, licensed, or leased to the public for non-governmental use) are commercial OTS (COTS). Note that by law and regulation, software licensed to the public and used for at least one non-government purpose is commercial software, even if it is maintained by the government. Figure 1 illustrates these different kinds of OTS maintenance approaches.

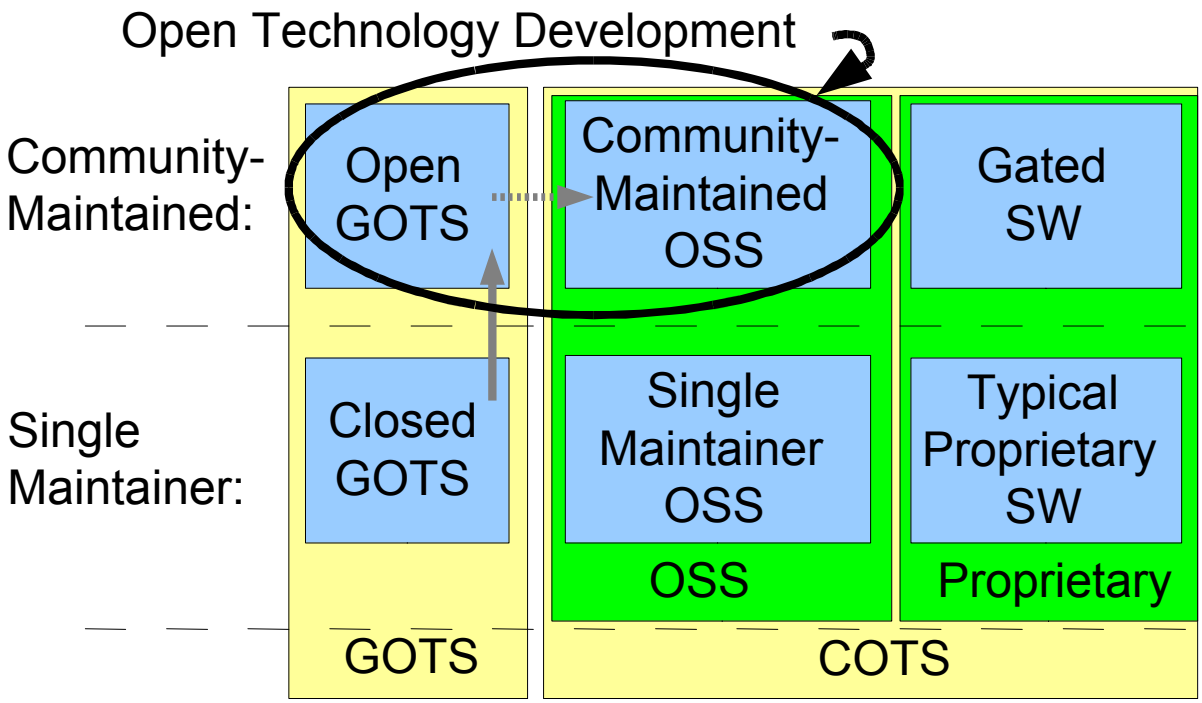

Figure 1. Off-the-Shelf (OTS) Maintenance Strategies

There are two kinds of commercial OTS (COTS) software: Open Source Software (OSS) and proprietary software. In either case they may be maintained by a single maintainer or by a community. In community maintenance there is often a single organization who determines if proposals should be accepted, but the key here is that the work tends to be distributed among those affected.

Today, where there is GOTS software at all, it tends to be developed and maintained by a single maintainer. This tends to reduce GOTS' applicability. Many government programs might potentially use a GOTS component if certain changes were made, but cannot make the changes to the GOTS component directly, and even if they did, there is no structure by which those changes could be merged back into the main GOTS product for all to use. In contrast, most OSS projects are maintained by communities, where different organizations actively work together to develop software that is useful to them all. Single-maintainer OSS project exist, but they are less common.

An Open GOTS (OGOTS) project is a GOTS project which uses multiple-organization collaborative development approaches to develop and maintain software, in a manner similar to OSS. Such a project within the DoD is sometimes termed "DoD community source software" or "Government Open Source Software" (GOSS). One goal of this paper is to increase the number of GOTS projects that are OGOTS projects. A project may become OGOTS instead of OSS because its leaders want the innovation, speed of development, and lowered cost that can come from co-development by many parties, yet:

1. The government lacks the intellectual rights to make it more open (e.g., the government may have government-purpose rights (GPR) and not unlimited rights), and/or
2. The government wishes to maintain a national security advantage by not making that software available to potential adversaries (typically such software will be classified and/or export controlled).

In addition, GOTS projects should determine when they should become COTS (e.g., as community-supported OSS projects). In particular, GOTS projects should seriously consider switching to OSS

maintenance after a system has been deployed. There are various reasons why the government should keep certain software in-house, e.g., because sole possession of the software gives the U.S. a distinct advantage over its adversaries. However, technological advantage is usually fleeting. Often there is a commercially-developed item available to the public that begins to perform similar functions. As it matures, other organizations begin using this non-GOTS solution, potentially rendering the GOTS solution obsolete. Such cases often impose difficult decisions, as the government must determine if it will pay the heavy asymmetrical cost to switch, or if it will continue "as usual" with its now-obsolete GOTS systems (with high annual costs and limitations that may risk lives or missions). This means that there is considerable risk to the government if it tries to privately hold GOTS software within the government for too long.

Open Technology Development (OTD) involves community development among government users, and thus includes both OSS and OGOTS.

1.4 Key Prerequisites for OTD

1.4.1 Intellectual Rights

By definition, an OTD project is built on collaboration. Such collaboration is *only* possible when there is a legal framework that permits it (including the necessary intellectual rights) and when the basics of this framework can be understood by ordinary laymen.

The term "intellectual rights" means the set of rights over an intellectual work that are held by various individuals through relevant laws, regulations, licenses, contracts, and so on. Relevant laws include copyright, patent, and trademark law. Copyright law is especially important; those who hold the copyright of a work, or have the key rights that a copyright holder has, can impose conditions on when the work can be used, modified, or redistributed. For purposes of this document, all laws and regulations impinging on how the works can be shared and used affect intellectual rights; including the laws and regulations for classification and export control.

A key issue in any OTD project is to work out the intellectual rights so that the software can be used, examined, modified, and redistributed among the community.

When done correctly, requiring the intellectual rights necessary for an OTD approach does not conflict with DoD policy. DoD Federal Acquisition Regulation (FAR) Supplement (DFARS) 227.7203 could be misunderstood as forbidding an OTD approach, but when closely examined it does not forbid OTD at all. DFARS 227.7203-1(c) (as revised on January 20, 2011) says that "Offerors shall not be required, either as a condition of being responsive to a solicitation or as a condition for award, to sell or otherwise relinquish to the Government any rights in computer software developed exclusively at private expense except for the software identified at 227.7203-5(a)(3) through (6)." Similarly, DFARS 227.7203-1(d) states that "Offerors and contractors shall not be prohibited or discouraged from furnishing or offering to furnish computer software developed exclusively at private expense solely because the Government's rights to use, modify, release, reproduce, perform, display, or disclose the software may be restricted." Note, however, that these policies only apply to software developed *exclusively at private expense*. The government *can* require that it receive rights to software that the taxpayer paid to develop (in part or in whole), and such software is the focus of this document.

Acquiring broad intellectual rights (as well as the intellectual products themselves) eliminates a major barrier to competition, one that the GAO and others have noted for many years and are actively working to address:

- Government Accountability Office (GAO) report GAO-06-839 of July 2006 [GAO 2006] reported that "The lack of technical data rights has limited the services' flexibility to make changes to sustainment plans that are aimed at achieving cost savings and meeting legislative requirements regarding depot maintenance capabilities... Unless DOD assesses and secures its rights for the use of technical data early in the weapon system acquisition process when it has the greatest leverage to negotiate, DOD may face later challenges in sustaining weapon systems over their life cycle."
- Government Accountability Office (GAO) report GAO-10-833 of July 2010 [GAO2010] found that for "services supporting DOD weapons programs, the government's lack of access to proprietary technical data and decades-long reliance on specific contractors for expertise limit—or even preclude the possibility of—competition." This is a serious problem, because competition "is a critical tool for achieving the best return on the government's investment." In nearly 60 percent of 47 noncompetitive Defense contracts examined by the GAO, the department was essentially stuck with a certain contractor since it lacked technical data behind the goods and services it has been purchasing.
- Ashton B. Carter in 2010 [Carter 2010] noted issues relating to competition. His first point on providing incentives is to "Avoid directed buys and other substitutes for real competition. Use technical data packages and open systems architectures to support a continuous competitive environment."
- David M. Van Buren's 2011 memo [Buren 2011] is the Air Force implementation of USD(AT&L) direction for a one-page competitive strategy. This Air Force memo requires that every Air Force program describe how it will "obtain technical data, computer software documentation, and associated intellectual property rights necessary for operation, maintenance, long-term sustainment, upgrades, and future competition" and a summary of the business case analysis if it is not obtaining them.

An OTD project requires certain intellectual rights; requiring such rights is consistent with DoD policy and direction.

1.4.2 Simplicity

Collaboration is inhibited by unnecessary complexity. Continuously strive to simplify the project. In particular, ensure that intellectual rights issues are simple and clear (e.g., through clear and common licenses), that material is easily available to all who should be able to access it, and that technical designs are clearly modular.

Chapter 2. Running OTD Projects

This chapter lays out the basic framework for running an OTD project. The first subsection describes how to establish an OTD program once a project proposal has been accepted. The next subsections discuss establishing a technical infrastructure for collaboration, communication issues, technical management/technical criteria, and continuous delivery. In the DoD these must fit into the DoD acquisition processes; how this can occur is discussed in chapter 3.

2.1 Establishing an OTD Program

Once a project proposal has been accepted, an OTD program can be established using the steps outlined below. Much more information on how to do this from an OSS project perspective can be found in chapter 2 of [Fogel2009].

2.1.1 Step 1: Determine reuse options

First, search for existing OSS projects that have relevant functionality. A simple web search of the string "open source software" plus a desired capability will often turn up something close to what you need. Also review OSS repositories sites such as http://www.sourceforge.net, http://www.freshmeat.net, http://www.github.com, http://directory.fsf.org and http://code.google.com. Even if there is nothing available to use directly, there might be piece-parts that can be integrated or useful ideas.

Opportunistic adoption of OSS is important because technological innovation is primarily occurring on the unclassified internet, not within the military sphere. Most of the piece-parts for any given project are already out there, and there is an expanding wave front of OSS software that can rapidly advance the needs of government projects. Careful evaluation, selection, and participation in these external projects is the most effective way to evolve capabilities over the life cycle of a government program. Existing GOTS software may quickly become obsolete once there is a public COTS project (including an OSS project) with the same goal.

You should also examine existing GOTS software options. There is no single place to find such software, but using public search engines can be valuable, as well as searching on Intellipedia and DTIC.

If you have software that was previously developed as part of a government contract, determine if you have sufficient intellectual rights to release or transition the software as an OTD project. For development and maintenance as an OTD project, the software (per figure 1.1) needs to be released as an OGOTS or preferably a community-maintained public OSS project. Appendix B clarifies the legality of releasing GOTS as OSS, depending on the authorities invoked by the original contract.

Many government programs have existing technology that was originally funded by the government. If the intellectual rights over those technologies is inadequate or cannot be determined, the government should consider negotiating with the appropriate integrators/vendors to release the source code under less restrictive data rights sufficient for an OGOTS or OSS project. An easy way to do this is to simply fund the conversion process for the contractor(s).

2.1.2 Step 2: Identify the Projects to be Established

Given the reuse options, identify what new projects are necessary and which existing projects need to be transitioned to OTD. In some cases, the "new project" may be a project to extend some existing OTD project and get that extension integrated into the original project. Where possible, split up the project into several smaller projects with clear interfaces. These smaller projects may be divided according to

various criteria, including the likelihood of reuse (to maximize the number of participants in at least some of the projects) and the need to limit access (classified or export-controlled modules may need to separated from other components, e.g., by creating an unclassified "framework" into which controlled "plug-ins" can be placed).

Name each project so that is not easily confused with other projects. It should be pronounceable and easy to find on a web search (ideally, it would be the only result from a search; certainly avoid unsearchable names like "the" or "why").

Each new project (including any existing project transitioning to OTD) needs a statement of intent that references the OTD software maintenance philosophy. As recommended in [Fogel2009] "the mission statement should be concrete, limiting, and above all, short." The mission statement should make it clear that the goal is to use open development principles (e.g. avoiding lock-in to a single supplier) and what the resulting products should do. Here's an example of a good one, from http://www.openoffice.org:

> *To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format.*

In a DoD project, the software maintenance philosophy statement might reference DFARS 227.7203-2 ("Acquisition of noncommercial computer software and computer software documentation"), and in particular the text at DFARS 227.7203-2(b)(1) (bold and underlining added):

> *Data managers or other requirements personnel are responsible for identifying the Government's minimum needs. In addition to desired software performance, compatibility, or other technical considerations,* ***needs determinations should consider*** *such factors as multiple site or shared use requirements, whether the Government's* ***software maintenance philosophy will require the right to modify or have third parties modify the software****, and any special computer software documentation requirements.*

Determine, for each project, whether it must be limited to only DoD or general government access as an OGOTS project. By default, projects should become COTS OSS instead of OGOTS. In some cases (e.g., due to classification or export control) a project must be limited to DoD or U.S. government access. GOTS projects present a higher risk than COTS projects, because by definition there are fewer potential contributors (decreasing competition and potentially increasing cost), and contractors (other than their copyright owners) are disincentivized from using GOTS projects because they cannot reuse those components or knowledge about them in other commercially viable ways. In many cases it is possible to split the project into two projects, one that is OSS (e.g., a "framework") and one that is OGOTS (e.g., a "plug-in" to the framework). This method was used successfully in the imagery and mapping domain when the National Reconnaissance Office sponsored the development of the Open Source Imagery and Mapping (OSSIM) framework. The framework is widely distributed and maintained by private sector developers, while classified plug-ins are developed on secure DoD networks and limited to those environments.

2.1.3 Step 3: Choose and Apply a Common License

Each project must have a clear and simple license that enables legal collaboration. A license lays out the rights and responsibilities of software developers and users.

If the project is to be an OSS project, be sure to choose a well-known pre-existing OSS license, one that has already been widely certified as being OSS. It should be GPL-compatible, as the GPL is the most common OSS license. If the software pre-exists, it is usually wise to include its previous license as one of the options. See Appendix D for more information on how to pick an OSS license.

2.1.4 Step 4: Establish Governance

Projects that use OTD need to be governed. The governance process for each project needs to encourage collaborative development, but it must also allow the rejection of contributions where warranted. The OTD governance process must enable multiple organizations to work together to improve each component undergoing shared development (including its software, tests, and documentation), instead of re-developing separate independent components with similar functionality. Before discussing different governance models, it is important to note that *forkability* is necessary, as described next.

2.1.4.1 Forkability

A *fork* is a competing project established using a copy of an existing project's software.

It is critically necessary that an OTD project be *forkable.* That is, it must be possible to create a viable competing project using a copy of the existing project's software source code. Creating a fork is similar to a call for a "*vote of no confidence*" in a parliament. The fork creator is essentially asking developers and users to stop supporting the original project, and support the new forked project instead (supporting both projects is typically impractical over time).

Forks can also occur because the existing community doesn't plan to include a feature set part of the community deems important, reasons could include: support for a different operating system or middleware or inclusion of a new programming language. Whatever the reason, every effort should be made to keep forked projects somewhat as coordinated as possible.

Forkability is a necessary part of OTD governance. As long as a project is forkable, project leadership will strive to be responsive to users and developers. This is because if leadership decisions are particularly egregious, a forked project can be started under more responsive stewardship. Easy forkability actually reduces the risk of a fork, because leadership will be forced to listen to users and developers (because if they do not, a viable fork will emerge). In addition, easy forkability increases the likelihood of contributions; easy forkability provides significant protection to would-be contributors, because if they later disagree with project governance, they can create a fork.

Having said this, avoid creating forks where possible. Most fork attempts fail, because the situation must be especially bad for many developers and users to defect to the new fork. Someone who attempts to create a fork and fails to establish the fork as a viable project will often end up maintaining their own project at tremendous cost. Normal business in both projects often grinds to a halt for a time, as discussions focus on whether or not the fork is justified.

For more about forking, see [Fogel2009] chapters 4 and 8.

2.1.4.2 Governance Models

Projects need some process for governance. Two common approaches are:

- Benevolent dictator (BD). In this case, there is one person in charge of final decisions. The BD may delegate decisions to others (while retaining a veto), but in the end, a single person is in

charge. This is particularly common in smaller projects where one person (typically the initiator) understands the project best, but it is even used by some large OSS projects (e.g., the Linux kernel).

- Group decision-making. In this case, a group makes final decisions. This may be through simple majority, or through some other mechanism. Some projects allow group members to give a veto as well. Typically this group is not the set of *all* contributors, but some subset, and this group must agree to the addition of a new member. A common process for doing is from the Apache Software Foundation. In this system a "+1" means approve, a "-1" means veto, and by default a proposal can only pass if it receives a total of at least +3 with no vetoes. In some cases Apache permits "lazy consensus" (aka "silence is consent"), in which a proposal is accepted unless someone vetoes (objects) within a period of time. In any such process there is a risk that vetoes will be overused; to counter this, Apache requires that all vetoes include a justification or they are invalid [Apache2010]. Many experts argue that voting should be a last resort (e.g., [Collins2007] and [Fogel2009]), and that you should try to obtain a rough consensus without a formal vote where possible.

A key issue is the number of people who truly understand the technical details of the project. If one person clearly understands it better than others (this is often true at project initiation), the BD model has much to recommend it. If many understand the project equally well, which is often the case for larger projects, the group decision-making process has much to recommend it.

Regardless of the governance model, the decision-maker(s) must avoid making a decision between alternatives too soon. If there is a disagreement, there may be a compromise or alternative approach that would be better than the immediately-obvious options. Therefore, decision-makers should try to get parties to find those compromises and alternatives. However, if a reasonable compromise cannot be found and a decision must be made, the decision-maker(s) should make that decision after listening to all sides. That decision should be announced clearly, along with sound rationale. The decision-maker(s) must also be willing to change a decision given important new information, new options, or a change in circumstances.

The project should be managed and structured to be constantly evolving and opportunistic. Adhering to open standards and interfaces will provide the flexibility for the program to rapidly adopt new solutions as they appear.

As noted in section 2.1.4.1, a key to any governance approach is that the project must be forkable. Any governance model can eventually fail if the decision-makers have no need respond to others. If the project is forkable, then the leadership (regardless of the governance model) must in the end respect the needs of users and developers.

More information can be found in [Fogel2009] chapter 4 and [Bacon2010] chapter 8.

2.1.5 Step 5: Establish Collaboration

Establishing collaboration isn't the same as creating a one-way communications strategy. Collaboration involves an easy interchange of ideas among many perspectives (including industry, academia and other government agencies offices and labs) to produce a better result than any one of them could have achieved separately. Section 2.2 discusses further how to establish the necessary technical infrastructure to enable collaboration; section 2.3 discusses the social dimension of communication.

When opening a formerly closed project, be sensitive to the magnitude of the change. Ensure that all its existing developers understand that a big change is coming. Explain it, tell them that the initial discomfort is perfectly normal, and reassure them that it's going to get better. Work to counter lapses into private discussions between long-time developers, and encourage their migration to community forums such as mailing lists. [Fogel2009]

Since some people will struggle with the openness of an OTD project, it is important to stress the need for openness. Point to guidance such as the current administration guidelines and mandates on transparency, and on the DoD 2009 memo on open source software which mandates that software be treated as data and shared appropriately. To quote the 2009 memo:

> f. Software source code and associated design documents are "data" as defined by DoD Directive 8320.02 (reference (h)), and therefore shall be shared across the DoD as widely as possible to support mission needs.

There are of course discussions that must be kept closed to the public, such as company source selection and company proprietary data. But every attempt should be made to open up the software development process as much as possible. To simplify governance, the preferred method is to use an OSS license unless national interest dictates otherwise. The government should also require contractors and software integrators to organize their projects so that they are continuously transparent and open to the government for remote inspection.

2.1.6 Step 6: Create Project Technical Direction

For each project, determine key technical issues, such as which major components will be reused, what components the system must interact with, how it will be implemented (such as what implementation languages to use), what platforms it must work on, and basic developer guidelines.

Each project should stress *modularity*. A modular system is a system built from smaller interacting projects that can be developed in parallel and individually replaced without affecting other components. Modularity is key and simplifies technology and software IP reuse, eases and separates classification and export control issues, simplifies management, speeds deployment, reduces maintenance costs, and increases agility. A great military reference to modularity can be found here: http://www.acq.osd.mil/osjtf/docsmemo.html. Well-known design patterns and architectural patterns can be used to divide problems into smaller components [Martin2000].

Section 2.4 discusses technical management and technical criteria further.

2.1.7 Step 7: Announcing

When a project is established and presentable (not perfect), or a significant event such as a major release occurs, tell others who would want to know.

If you know of mailing lists where an announcement of your project would be on-topic and of interest, then post there, but be careful to make exactly *one* post per forum, and to direct people to your project's own forums for follow-up discussion. If there are related projects (e.g., ones that might likely use it or be impacted by it), be sure to provide them the news, and invite them to post web links to your project website. Many people develop web pages with lists or reviews of certain kinds of software; provide them the information so that they can improve their web pages. Post an update on Intellipedia and the DoD Techipedia (this is especially important for OGOTS projects, since it can be difficult to find them

if they are not publicly known). If it is a public OSS project, submit such announcements to freshmeat (http://freshmeat.net/).

2.1.8 Continuously Review Steps 1-7

Steps 1-6 should are the start of a continuous process where projects should constantly be cycling through the search for new components, growing the community, maturing the technologies and seeking to scale the size, heft and maturity of the community. Over time the community should grow, thereby bringing in new people and ideas and leading to an increase in competence and competition for government contracts.

2.2 Technical Infrastructure for Collaboration

Since collaboration among widely-distributed contributors is key to OTD, projects must establish a project site with the technical infrastructure needed for collaboration. The project site must enable the shared development of the software, test suites, and documentation (including user, installation, administration, and design documentation), though the details of how these occur often vary between projects. Useful guidelines for establishing the technical infrastructure can be found in [Fogel2009] chapter 3; the following are a few key points. In DoD contract language this technical infrastructure is the set of "data repositories" (see DFARS 227.7108), but the need for collaboration means that these repositories must implement additional requirements not always mandated by contracts.

It must be possible for all potential contributors and users to use the tools easily. For example, if security restrictions make it too difficult for people to participate, they will not participate. Projects should prefer to use widely-used OSS collaboration tools that work well with any standards-compliant web browser. Unusual tools create an unnecessary barrier to entry, as they require users to learn how to use new tools instead of simply contributing. (Even if users have learned how to use a tool, users will be more willing if the tool is widely used because their learning time is amortized.) OSS tools should be strongly preferred; they can be configured for special needs, tend to be inexpensive to deploy, and tend to be especially good at OTD-style collaboration since they are often used for that purpose. Maximizing access via standards-compliant web browsers increases the ability for others to interact with the product, e.g., they can interact from the field.

Contractors must expect that this technical infrastructure means that the government and other contractors will have continuous access to intermediate progress. This transparency is by design.

2.2.1 Key Functions

The central project site must support the collaboration of ongoing improvements and should provide the following functions:

- *Front door (web site)*. The central project site must provide a single starting point for those interested in the project, enabling people to learn about the project and find all related information (in particular, how to obtain it and/or become part of its development community). This is normally a web site with a simple fixed URL.
- *Bug and feature tracking*. The central project site must provide a mechanism for users to submit bug reports and feature requests, and for developers to determine how (or if) to resolve them. This is often implemented through specialized tools such Bugzilla, Trac, or Redmine, but other tools (such as wikis) can be used. There may need to be a special process for reporting security vulnerabilities to prevent their disclosure before a repair is available.

- *Software Configuration Management (SCM)*. The central project site must provide a mechanism for tracking changes, including at least the software and often test suites and some documentation. It should at least provide a method for seeing and tracking the "main development branch" and each major release. The SCM must make it possible to see who made each change, when, and what the change was. Do not use CVS in new projects; CVS was historically popular but has been superseded by better tools. There are two major types of SCM systems, centralized SCM (e.g., subversion aka SVN) and distributed SCM (e.g., git and mercurial). As discussed below, in a military environment distributed SCM systems (such as git) have significant advantages and should be preferred for SCM of OTD projects.
- *Community interaction (mailing list, wiki, and/or IRC)*. The central project site must provide a mechanism for users and developers to discuss issues. Community interaction is often implemented through mailing lists (e.g., Mailman), wikis, or chat systems such as Internet Relay Chat (IRC). Many projects use mailing lists for general discussions because they make it easy to involve geographically-distributed people, capture discussion for later use, and give people time to create thoughtful responses. In a military context mailing lists and wikis tend to be easier to use, because both are already widely implemented on military computers. If using chat systems, be sure to archive discussions in a way that later participants can find them, or important discussions will be lost. Also, be sure that the chat system supports any necessary security requirements (such as authentication and confidentiality).
- *Release downloads*. The central project site must provide a mechanism for download of major releases; if these are large, mirroring may be necessary.

Wikis (such as MediaWiki, MoinMoin, PmWiki, and PhpWiki) are flexible programs and can be used to fulfill several of these functions.

2.2.2 Public access, classification, and export control

Where possible, determine which components can be released to the public as OSS ahead-of-time, and establish the project *outside* in the public from the beginning. This eliminates many problems, as this makes it easy for others to find and contribute to the project, and greatly increases the number of potential contributors. This also enables the use of commercially-available hosting services (see section 2.2.3). There is no legal requirement wait until the project is "feature complete" before this release occurs, and if it is to be public anyway, the sooner it is publicly released the better.

Some components are classified, and thus their development may only take place on systems authorized for classified processing. Similarly, some components will be under export control under ITAR or EAR. Where possible, divide components based on their sensitivity to limit what must be protected by classification or export control. In these cases, work to establish an OGOTS project that allows collaboration between those authorized to have access to the project. Such projects must work especially hard to ensure that potential users and collaborators know about their existence.

In many cases software development will occur that is intended to be available to the public as OSS, but must undergo classification and/or export control review first. This can be a significant barrier to collaboration, but in many cases it is a necessary step. Consider setting up an internal OGOTS project to "stage" these proposed changes until they can be released to the OSS project. Staging is especially important if it is determined that some changes are needed for government use and *cannot* be released to the public. Such proposals should ideally be set up as separate "branches" so that even if some are deemed to be unreleasable, other changes can still be independently released.

Projects should seriously consider using a distributed SCM (such as "git") where there may be export control issues or varying levels of classification. Distributed SCM systems make it much easier to create separate external branches and later provide them to others for merging if approval is granted.

2.2.3 Hosting

Each project must determine if it will be based on some existing collaboration hosting service that provides pre-configured functionality, or if it will establish its own system, obtain the necessary tools, and host the project itself. Whatever the decision about hosting, you must be able to compete and change who does hosting support down the line. Do not be locked into a single supplier. If it is not possible to easily export all code, documentation and discussion materials from a hosting service and import that information into a different hosting environment, then your OTD project is effectively held hostage by a proprietary hosting service (an unfortunate irony that may undermine the legitimacy of the project itself). A key criterion for the evaluation of hosting services is to determine how difficult it would be to copy the data (e.g., bug reports, source code, etc.) elsewhere so that if the hosting service is inadequate, the project can easily move.

Most larger or widely-used OSS projects (e.g., the Linux kernel) establish their own collaboration infrastructure using existing OSS tools instead of using a hosting service (though they may use a hosting service like SourceForge to distribute final releases). At first glance, using an existing collaboration hosting service may seem like the simplest and easiest option, but hosting services are not always the best way to go. A hosting service is typically set up to provide generic services, which may not be well suited to a particular project. Small projects are often well-served by using an existing collaboration hosting service, as they cannot justify the resources to specially configure their infrastructure. But medium-sized and large projects often benefit from the increased configurability of their own collaboration infrastructure, particularly if it's OSS.

Ideally, if a project is to be publicly available as OSS, it should be established as a public project before the design is created. But regardless of when the project is to be made public, it is important to evaluate the appropriateness of commercially-available hosting services. Such hosting services include SourceForge (http://www.sourceforge.net), github (http://www.github.com), gitorious (http://gitorious.org), and Google Code (http://code.google.com/). The GSA has worked out terms of service and agreement with SourceForge; for more information, see: https://apps.gov/cloud/advantage/cloud/sa_details.do?BV_UseBVCookie=Yes&clid=160&catId=66. Wikipedia has a page that compares such services at http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities.

If the project is to be OGOTS, then it must establish collaboration infrastructure in a way that allows the government and its authorized subcontractors (at any tier) to fully access it. One generic collaboration hosting service is DISA's Forge.mil (http://www.forge.mil/). Forge.mil can be useful for OGOTS projects, but there is no requirement to use Forge.mil. Depending on your project needs, it can be a good or a bad choice. One advantage of Forge.mil is that little needs to be done to start using it. But Forge.mil also has a number of drawbacks. It can often be time-consuming for potential participants to get access, especially if they do not work directly for a DoD customer, because a CAC card or ECA certificate is required for access (access must be government sponsored). This can be a particular problem for members of the intelligence community who aren't credentialed by DoD. Forge.mil lacks a number of services; in particular, its lack of strong support for git or any other distributed version control system makes it unsuitable for many federal government projects. Finally, as with other

commercial collaboration hosting services, it provides generic services which may not be tailored to meet the needs of a particular project.

Many OGOTS projects may determine that, just like many public OSS projects, they would be better served by establishing their own host. In this case, they may find that a service like Forge.mil's "SoftwareForge" is a useful way to *distribute* final releases, even if they do not use it for collaboration during development (similar to how many OSS projects distribute releases via SourceForge, even if they do not use its collaboration services).

The government does not need to host OTD sites using government networks or facilities. The government may create an RFP for a contractor to run a hosting site that can be transitioned to the government or to another contractor. Community sites may also be maintained by federally funded research and development corporations (FFRDC). The key is for the government to set the expectation of transparency and collaboration around taxpayer-funded technology.

2.3 Communication

Effective communication is the key to success with any project or program. Leaders need good communication and motivational skills to ensure that developers and users receive the information they need. They also need the ability to rapidly assess opportunities and feedback from developers and end users. Finally, they need to enable all involved, including developers and users, to collaborate.

This is especially evident in successful OSS projects where users and various levels of developers are in constant communication through mailing lists, chat rooms, and bug tracking systems. Examine successful OSS projects and consider adopting their practices. More information can be found in [Fogel2009] chapters 6 and 8, and in [Bacon2010] chapter 3.

2.3.1 Be Inclusive

In general, collaboration should be encouraged, and the number of effective participants should be maximized.

Integrate end-users and developers from the beginning, versus developing in isolation and involving users only at the end. Successful open technologies co-evolve between their developers and end-users, with user comments and feedback taken into account at every stage of development. An OTD technical manager should make special efforts to recruit end users into the central project site as members of the project's community – particularly technical end-users who can provide constructive feedback about how the project helps or conflicts with existing workflows. In some cases, technically savvy end-users end up making small-scale (or larger) technical contributions to the project, from bug tracking to module development. This end-user investment of time and thought creates a bridge to the user community that facilitates transition, and makes it easier to scope and specify follow-on work that the customer will embrace. The technical manager should constantly strive to build and maintain a close working relationship between developers and the end users.

Encourage newcomers. Where their comments are emotionally-laden, try to identify the technical or procedural issue behind the feedback so it can be acted on (e.g., and turn it into a real bug report). Try to give others (especially newcomers) the benefit of the doubt.

2.3.2 Avoid Private Discussions

Unless legally forbidden, perform all discussions leading to a decision using forums such as mailing lists that are (a) recorded for later review and (b) available for all potential participants (e.g., developers and users). Many projects have a rule that all decisions must be made on a mailing list, not face-to-face, to ensure that everyone who wants to can participate in the process, and to ensure that decisions and rationale are recorded for later use.

For OSS projects these discussions should be truly public. OGOTS projects typically require some sort of log-in process before allowing others to view project information such as discussions. However, once someone has been authorized to view project data, they should normally be able to see all of the project's discussions. Similarly, once someone has become part of the project, they should be able to see and participate in all of the project's discussions.

Many founders are tempted to settle difficult issues via private communication. Yet more-public discussions are almost always preferable in the long run. If important decisions are made in private, community support for the project will wane. In addition, opportunities to receive others' feedback, and to explain to others *why* the decision was made, will be lost.

Strictly speaking, this is all part of being inclusive (see section 2.3.1), but lapsing into private discussions is a particularly common mistake.

2.3.3 Use Communication Mechanisms Effectively

Ensure that the project website is updated as appropriate so that potential users and developers can quickly learn what they need to know. As rules are established and frequently-asked questions are answered, ensure that these rules and answers are written down and posted in an easily-accessible location so that others can quickly learn that information.

Ensure that every change proposal includes a (brief) explanation of what the change does. That way, people can understand what it is supposed to do, and if the change is accepted, that information can then be provided to users so that they know what changed and why. When a proposal (e.g., a contribution) is rejected, ensure that there is a clear explanation of *why* it was rejected, so that the contributor can change it into something acceptable and/or so others won't waste their time re-doing the same thing.

Maximize the use of communication methods that are recorded and available to others later (e.g., mailing lists). Ask people to look for previously-resolved issues *before* posting on a topic. If they raise previously-discussed topics anyway, point them to the previous discussion. This way, people's time is not wasted by old, previously-resolved issues.

2.3.4 Practice Conspicuous Code Review

Conspicuously review and comment on code. This peer review improves quality both directly (through those comments) and indirectly (by alerting others that they must do good work since others *will* examine their proposals). As with any other discussion, code review should avoid private discussions.

2.3.5 Nip Rudeness in the Bud

Don't tolerate rude or insulting behavior. For example, when someone posts a technical comment mixed with an *ad hominem* attack, address the *ad hominem* attack *first* (making it clear that it is unacceptable) and then address the technical issue. In extreme cases people may need to be removed from the project, but in most cases, occasional reminders will stop rudeness.

2.3.6 Counter Poisonous People

One issue that leadership must deal with is "poisonous people". These are not necessarily rude people, but simply people who inhibit instead of enabling progress. For example, perfectionism can be a problem, because the perfect can be the enemy of the good. High quality is very important, but if striving for perfection is resulting in no progress, it is not acceptable. Some people agitate for the project to change in radical directions not within its mission; remind such people about the project mission, and keep people focused on that mission. Some hostile people attempt to deliberately make others angry; ignore or remove them from the project so that the project can continue. If someone is continuously draining attention and focus, leadership should determine if the person should be removed based on whether or not the person is likely to eventually benefit the project, and if the person is paralyzing the project right now. Leadership should encourage politeness, respect, trust, and humility among participants to maximize effectiveness [Collins2007].

2.3.7 Be Aware of Roles

Basic shared development typically involves people with the following roles:

- Developers. These people develop the software and related materials (e.g., tests) as a series of small proposed changes ("patches"), each small enough to be reviewed by others. They also review proposed changes.
- Trusted developer(s)/leader(s). These people act as final reviewers of proposed changes, to determine which ones enter the "main development branch" of the software and when formal releases are made. These people should be trusted to be "honest brokers" and not favor a particular organization, and must be trusted to be sufficiently technically knowledgeable to act as reviewers of each proposed change.
- Users. These people use the resulting technical capability, and have the option to submit bug reports or feature requests. Some may determine that they wish to change the software, becoming developers.
- Distributors. These people help users deploy the component with other components. They may package up the component, often with other components, and may provide additional services (such as training, installation, and support).

2.4 Technical Management/Technical Criteria

Technical leadership should work to ensure that the project's results are of high quality.

2.4.1 Goals

Technical leadership should strive for the following in the components being developed in an OTD project:

1. *Flexibility*. A component that can be used in a variety of ways tends to have more potential users, some of whom may aid the project (e.g., via bug reports and development time).
2. *Portability*. A component that can be used on more platforms tends to have more potential users.
3. *Modularity*. A component that is modular (e.g., with clearly-defined sub-components and perhaps support for a "plug-in" architecture) is more flexible, as well as being easier to review for correctness.

4. *Use of open standards*. Where possible, avoid depending on interfaces that are controlled by a single vendor.
5. *Reuse and collaborate with existing OTD projects*. A project should focus on building new software, not re-implementing OTD projects that already exist. This is further discussed below.
6. *Avoiding non-OTD dependencies*. Depend only on widely-used OTD platforms, libraries, and development tools. If a component depends on a non-OTD component, and that component then needs to be changed, it may be difficult to make the change or have that change incorporated. Similarly, it may be difficult to get support for unusual libraries and development tools. It's fine if a component *may* be used on a non-OTD platform (enhancing its portability), as long as it doesn't *depend* on it. If a proprietary component must be depended on, isolate it through plug-ins or an interface defined by an open standard.
7. *Correctness*. Strive for simplicity in the code, as simpler code is more likely to be correct. Demand that an automated regression test suite be included in development, so that as changes are made many errors will be immediately detected.

Where there are alternative approaches, a simple analysis of alternatives should be performed, and discussed among the project so that key issues or alternatives are not overlooked.

2.4.2 Reuse and Collaborate on OTD components

A distinctive characteristic of OTD technical management (besides community development) is the opportunistic adoption of open technologies developed by others.

Leaders of OTD projects must work to encourage the reuse of existing OTD components, including collaboration in their development when necessary. It is sometimes difficult to adopt components developed by "outsiders". Nevertheless, leaders should enforce the adoption of best-of-breed OTD projects, including efficient modification of pre-existing projects. A project's time should be focused on developing new software instead of developing software that already exists.

When considering the use of existing OTD components, evaluate them first. [Wheeler2010e] provides an approach for evaluating OTD projects, including how to get the information for such evaluations.

2.4.3 Don't Fork OSS Solely for Government Use

A common mistake made by companies and government projects that begin to adopt OTD approaches is to start by creating a fork by taking a snapshot of the source code and modifying it for their own needs, in isolation from the community surrounding that code.

This is a mistake because successful OTD projects are constantly evolving and improving. Creating a fork isolates all fork users from the main OTD project, including the improvements it makes. Refreshing OTD components is a very effective way of evolving the baseline for the project. It is important to remain synchronized with latest formal releases of the selected projects for system reliability, technological relevance, and obtaining the maximum benefit of an OTD approach.

In some cases, there is no need to modify the component itself. The component's application programmer interface (API) or plug-in system may provide the necessary flexibility without changing the component at all.

If a component must be changed, fixes and key enhancements to the baseline should be developed in consultation with original project and then submitted back to the original project. Unique government

interfaces and functionality should be segregated through plug-in mechanisms or with application programming interfaces (APIs) at a higher level. Taking this approach allows the government project to painlessly upgrade when new releases are made by the external project. Most useful components are continuously improved, so the ability to perform periodic upgrades must be built into the development and maintenance process.

In some cases, a project must make significant modifications to an OTD component it will depend on. First make sure that this is really the case; sometimes it is not. But if it is the case, discuss with that component's project the changes that need to be made, and look for ways to submit those changes incrementally to the upstream project. This will increase the likelihood that these changes will be accepted by that component's project. It is best if there is a contract incentive that changes to external projects be accepted back into those projects, to encourage the contractor to work with those external projects.

In all cases, focus development on those requirements that are truly unique.

Note that this is not a contradiction with section 2.1.4.1, which emphasized forkability. It is important that a project be *forkable* to encourage project leaders to manage the project properly. But since the process of forking drastically interferes with reuse, all parties should try to avoid actually *creating* a fork, and should instead find a way to work together.

2.4.4 Open Standards

Use open standards. For purposes of this paper, an "open standard" is a specification that at *least* meets the European Union's definition as adopted in the European Interoperability Framework:

- The standard is adopted and will be maintained by a not-for-profit organization, and its ongoing development occurs on the basis of an open decision-making procedure available to all interested parties (consensus or majority decision etc.).
- The standard has been published and the standard specification document is available either freely or at a nominal charge. It must be permissible to all to copy, distribute and use it for no fee or at a nominal fee.
- The intellectual property - i.e. patents possibly present - of (parts of) the standard is made irrevocably available on a royalty-free basis.
- There are no constraints on the re-use of the standard.

Sometimes extensions are needed, but they should only be used with consideration as it can be easy to become accidentally locked into a proprietary extension. Being locked into a proprietary extension can be a problem, particularly if it is only implemented by a proprietary program (since this effectively eliminates competition, raising costs long-term).

Consider requiring tests (as part of the contract) with an alternative implementation of a standard to increase the likelihood of staying within standard.

Where appropriate, create or work to extend open standards.

The DoD IT Standards Registry (DISR) is an online repository of IT standards, and may be of some use. It is available at https://disronline.disa.mil/ but access requires a CAC card. Note that not all standards listed in the DISR are open standards.

2.4.5 Managing Contributions

Project leaders should review contributions or ensure that they are reviewed. The contribution process should ensure that contributors meet technical requirements (e.g., that it compiles and appears to provide value) as well as any legal requirements (e.g., that they have a right to submit it). If the project requires rights assignments, make sure those are in place too.

First-time contributions are often rejected, because the contributors are often unaware of details about the project. Where practical, explain in detail *why* a contribution was rejected (focusing on the contribution and not on the contributor), and help the contributor understand how to make the necessary changes to produce an acceptable contribution.

2.5 Continuous Delivery

Development should be a continuous evolution through relatively small tracked changes. That way, others can effectively review these changes. These changes should not prevent a system from building or running. In some cases, a change will not have a user-visible effect, e.g., it may be an architectural change to prepare for future functionality. Daily builds followed by automated regression tests are highly recommended; these make problems immediately apparent.

In many ways formal release (and delivery) of an OTD product is a non-event due to its continuous development, testing, and feedback among all participants. Still, users typically need to know that a particular version is "ready for use", and typically specialized testing is performed that cannot be performed on each revision (e.g., field testing) before a release. Such formal releases are a good time for upper-tier contractors and the government to ensure that:

1. They actually receive the source code to all software they have rights over.
2. The source code and documentation have accurate rights markings. If the government receives unlimited rights, it should say so.
3. The software should automatically build using a single short command (e.g., "make").
4. The software should follow installation standards and guidelines for its platform. For Unix and Linux, see [Wheeler2009].
5. The regression test suite should be included.

On a formal delivery, announce the existence or new release widely so that potential users can know about it. If its existence can be known to the public, ensure that the announcement can be found by common search engines such as Google. Update the project's Intellipedia page(s) if it is OGOTS. If it is a DoD project, provide a copy of its source code and related material to DTIC to reduce the risk of data loss.

See [Fogel2009] chapter 7 for more information.

2.5.1 Managing Intellectual Rights

Ensure that each contribution includes the necessary intellectual rights (including "data rights") that enable the project developers and users to continue in their use, modification, and redistribution as appropriate. In particular, examine copyright markings on contributions, and look for the insertion of new dependencies on proprietary tools and components. Incorrect markings are often copied to other material, so incorrect marking can "spread" to other projects.

An OSS project must reject any contribution that does not meet the OSS project's chosen license(s). Similarly, an OGOTS project must reject contributions that do not permit OTD development. In particular, an OGOTS project should reject contributions with only "restricted rights" as defined in DFARS 252.227-7014(a)(14) as these do not provide the government and contractors with sufficient rights to reuse the software in arbitrary government circumstances.

OGOTS projects should normally accept any contribution with government unlimited rights. An OGOTS project *may* choose to accept contributions with Government Purpose Rights (GPR), especially if there is a clear expiration time after which the contribution reverts to government unlimited rights. Such contributions can only be freely used for government purposes, but since OGOTS projects are already only available to government users, this is not a real change. Accepting GPR contributions should be specifically approved by the government, since accepting such contributions limits what the government may do with the project results. The specific contract used to create the GPR software governs, but in many cases GPR software is GPR for 5 years after the signature of the contract or contract modification signature per DFARS 252.227-7014(b)(2)(ii); after that time the contribution becomes government unlimited rights. Each contribution should clearly marked as GPR and include the date when the contribution will change to unlimited rights, so that it will be easy for the government to determine when it will have unlimited rights instead of GPR. OGOTS projects that accept GPR contributions must ensure that all recipients are aware of and have agreed to the relevant GPR restrictions *before* they receive access to the GPR material.

For more information on intellectual rights on OSS, see [Fogel2009] chapter 9 ("Licenses, Copyrights, and Patents").

Chapter 3. OTD Programmatics: Tactics, Tools & Procedures

The best chance for a military program to become (and stay) open is when the rules of the "open road" are laid down clearly and succinctly early for contractors and most importantly program office personnel. This chapter will explain how to establish these rules in an acquisition. See DoD Instruction 5000.02 (Operation of the Defense Acquisition System) for background information on the DoD acquisition process.

3.1 Initiation and/or Transition to OTD

It is vital to include industry (and other interested parties) as early as possible, for example, by hosting Industry Days, coupled with a request for information (RFI) process. If possible, try to create an open web portal detailing ideas and solutions proposed for developing the desired capability. The goal is to gather as many ideas as possible to make the best informed decision for the government. Program managers may also attend community conferences and host brain storming sessions to gather ideas. All material generated in those sessions should be openly published for all potential bidders to see and utilize (as long as all bidders are informed, many legal issues disappear). If properly managed, the community that forms at project inception can grow and evolve the technological capabilities of the system through transition and beyond. The collaboration site can also eventually grow into a more formal technical data sharing site where project designs, specifications, data formats, test procedures, etc. can be hosted.

The following subsections delve into more detail about the specific government acquisitions process, such as Analysis of Alternatives (AoA), Requests for Information (RFI) from industry and the Request for Proposal (RFP).

Key points to remember:

- ✓ Review intellectual rights associated with the project; ensure that they enable community development and release.
- ✓ Prefer community-developed approaches (OGOTS or OSS approaches) for developing new software.
- ✓ Decide when/how to develop OSS, as well as when to transition & how.
- ✓ When talking with lawyers, ask "how can I achieve goal X?", never "can I do X?". Lawyers may simply answer "no" to the latter, without helping you legally achieve the larger objectives.
- ✓ Reach out to wider communities and on-line forums, e.g., Intelink and the MIL-OSS group.
- ✓ Determine the security requirements and required processes, e.g., Certification & Accreditation, FIPS, and Common Criteria.
- ✓ Review export control (EAR, ITAR) and classification projects needs. Use existing OSS tools to keep the project open.
- ✓ Use/modify/create open standards, in that order. Verify that the standards used are open; a simple test for openness is to determine if the standard is implemented by open source software.

3.1.1 Analysis of Alternatives (AoA)

Early versions of the Analysis of Alternatives (AoA) can usually be developed in an open fashion. Beyond simply funding a study team, program managers should make an effort to publish findings in the

open on the web (or if sensitive on Intelink) and to actively solicit outside opinions and ideas from people not normally associated with the military or federal government, as there is often more expertise outside than inside.

The official references for AoA can be found here: https://dap.dau.mil/policy/Pages/overview.aspx or https://acquisition.navy.mil/content/view/full/3486. A good practical reference for AoA can also be found in "Analysis of Alternatives (AoA) Handbook A Practical Guide to Analyses of Alternatives" (July 2008).

The AoA should identify existing standards that can be used as well as areas where new standards may need to be created. Any new standards must be published as open standards unencumbered by intellectual rights. Proper use of application programming interfaces (APIs) should also be reviewed and recommendations made. APIs must be owned and controlled by the government or an independent standards body because they are potential points for lock-in.

Existing open source software (OSS) is often neglected during AoA development. This is contrary to law and policy, which requires market research of commercial items (including publicly-available OSS). A program office must search out OSS alternatives for use, and if appropriate, experiment with these tools. Places to check for OSS tools include: http://www.github.com, http://www.freshmeat.com, http://www.sourceforge.net, and simply searching the web with the string "open source software" plus the capability you are trying to develop.

AoA Sample Questions:

1. What are the purposes, goals and requirements of the system?	2. Are there any existing relevant COTS and GOTS capabilities, including OSS and OGOTS?
3. Are there other military programs building or developing similar capabilities? Can you team with them (e.g., to share technology)?	4. Are there other government programs building or developing similar capabilities?
5. Are there other international partners (NATO, etc.) programs building or developing 'like' capabilities?	6. Are there other existing communities your program can engage with (IEEE, NIST, ASME, NDIA, etc.)?
7. If your program needs to build a custom solution (i.e., GOTS) can and should you release it as open source software?	8. What standards should/could used?

3.1.2 Request for Information (RFI)

After an initial AoA is developed, a next step is to officially publish a request for information (RFI), which is a good first opportunity to formally engage with industry. The process for creating an RFI is laid out in the DoD 5000 series.

The RFI should have a detailed description of what capabilities the government desires. Some issues and general questions to be asked in the RFI may include:

1. Intellectual Rights

   a. *The government desires unlimited data rights for all software source code developed at taxpayer expense, so that it can release that software as open source software (OSS) to implement a software maintenance philosophy of OSS community development (per DFARS 227.7203-2(b)(1)).*
   b. *The government may direct the contractor to release source code as OSS. Describe how this may occur and what OSS license would be best.*

2. Data Formats, Standards & Interfaces
   a. *The government will only use Data Formats, Standards & Interfaces that are open and openly accessible.*
   b. *What Data Formats, Standards & Interfaces are proposed for use? For a given data standard, is there an open source software implementation of that standard?*
   c. *Where open standards are not available, discuss proprietary standards that exist, how new open standards would be created, and what groups would govern them (e.g. IETF, W3C, OASIS, etc.).*
3. Off-the-Shelf (OTS) Technologies
   a. *Per government law and policy, COTS (including open source software) is acceptable for use in this program.*
   b. *Please identify technologies that can be used:*
   c. *Adopt: What technologies are available for use unmodified?*
   d. *Modify: What existing capabilities that can be refined and/or modified?*
   e. *Create: What technologies may need to be created?*
4. Open Technology Development Practices
   a. *The government would like to engage with a wider developer and user audience for this capability; please describe how this could occur.*
   b. *If portions or all the capability is export controlled and/or classified, explain how a contractor /integrator could create and govern source code and effectively engage with a community.*
5. References
   a. If possible the AoA could also be published to help the community understand potential solutions already explored and more importantly which ones haven't been examined.
   b. The various open references mentioned in this guide could be included as well.

The RFI should ask for specific examples of open standards previously identified and proposed for use. For proposed software that does not exist and will need to be funded the contractor must detail how it will be delivered and how the contractor will develop a community around that software capability.

3.2 Request for Proposal (RFP)

After the initial AoA and RFI are complete, the program office should have a pretty solid idea about what technologies are available, what competition exists, and what technological direction the program may wish to go. One key decision that the government can make is to decide whether being open is a

key performance metric and requirement. If technology transparency in software source code is a key metric and parameter that the government needs to fulfill its mission it should state so in the RFP.

The following subsections detail RFP objectives that help ensure openness in a military capability.

3.2.1 Statement of Objectives (SOO) & Intent

A clear statement of intent and program objectives is necessary to establish the context of what being open will mean for a project. As discussed previously, being open has a number of different facets, from open standards to open source software to open development. Each program's leadership will need to decide on their own where in this continuum they can be and want to end up longer term.

The SOO should make statements about OTD, such as defining OTD, noting that it is the intended "software maintenance philosophy" (the DFARS allows specifying the software maintenance philosophy as justification for these kinds of activities), and requesting offerors explain how they will implement OTD.

For example:

The government intends to use an Open Technology Development (OTD) software maintenance philosophy (see DFARS 227.7203-2(b)(1)) to increase agility, increase competition, lower cost, and speed deployment. OTD is an approach to software/system development in which developers (who need not be in the same organization) collaboratively develop and maintain software or a system in a decentralized fashion. OTD depends on open standards and interfaces, open source software and designs, collaborative and distributed online tools, and technological agility. The Offeror shall describe how they will implement Open Technology Development (OTD), including how they intend to grow, evolve, and/or contribute to a multi-organization development community. The Offeror shall discuss their plans for using, implementing, and developing open standards and interfaces, open source software and designs, and collaborative and distributed online tools.

The software/system should continually evolve as a common open capability jointly developed and upgraded by the military, and where appropriate, by industry. Over time the capability should become cheaper to own, operate and upgrade via increased competition through the rigorous adherence to open standards, commercialization of technologies, increased use of open source software and exercising of unlimited government purpose rights. It should also be flexible to enable the capability to scale rapidly in response to new, evolving and future threats.

3.2.2 Intellectual Rights

Ensure that the government gets the necessary rights to implement OTD.

Example text:

The government will use open technology development (OTD) as its software maintenance philosophy (see DFARS 227.7203-2 (b)(1)). This means that the software not solely developed at private expense will be maintained through collaboration between multiple organizations, either solely within the government or with the commercial sector, at the government's discretion. Therefore, the government must have unlimited rights to all computer software (including source code, formal & informal design documents) and technical data (including user manuals) developed under this contract unless the government provides specific written permission to do otherwise. Delivered software must provide unlimited rights or provide an open source software license unless the government provides specific written permission to do otherwise. Furthermore, the government desires that the contractor release

any developed software source code under an open source software license unless the government classifies the software or specifies otherwise in the interest of national security.

3.2.3 Data Formats, Standards & Interfaces

Appendix A provides pointers to some of the many U.S. government directives and policy related to open data formats, standards, and interfaces.

Example Text:

Per U.S. Government directives and policy this program will prefer and use openly accessible data formats, standards and interfaces. If any of these are not available the contractor should detail how it could adopt and/or modify existing standards or how it will create new ones and release them in a way that ensures that they are free and unencumbered. In particular:

1. The following data formats, standards and interfaces shall be used: <LIST ONES APPROPRIATE TO THE PROJECT, AND DESCRIBE HOW THEY MUST BE USED>.

2. The resulting software/system must be modular. The offeror must identify the major modules and how those modules will interact.

3. The contractor shall use and/or extend open unencumbered standards during the course of this program. If they are extended, they shall be documented in a form (and with the necessary rights) so that the extensions can be submitted to an appropriate standards body.

4. If open standards for key external interfaces do not exist, the contractor will create and publish open unencumbered accessible standards. What standards does the contractor propose to create, what are their scopes, and how will they be created?

5. What other data formats, standards & interfaces are proposed for use? In each case, is there an open source software implementation that processes or implements it?

3.2.4 Off-the-Shelf (OTS) Technologies

Example Text:

Per government law and policy, the use of commercial off-the-shelf (COTS), including open source software, is preferred, and reuse is mandated over (re)development. Contractors should evaluate technologies for use in this order:

1. *Adopt existing technologies. This includes COTS (including OSS) and GOTS (including OGOTS)*
2. *Modify: What existing technologies that can be refined and/or modified?*
3. *Create: What technologies may need to be created?*
4. *Offeror shall list what GOTS products are to be used and examine the possibility of converting and releasing it for collaborative development, as either open source software (OSS) or Open GOTS (OGOTS).*

3.2.5 Open Technology Development Practices

Example Text:

The offeror should use open technology development practices:

1. *The government would like to engage with a wider developer and user audience for this capability. Describe how this would occur.*
2. *If portions or all the capability is export controlled and/or classified, explain how a contractor /integrator could create and govern source-code and effectively engage with a community (e.g., framework with plug-ins, distributed configuration management system, and so on).*
3. *Source code releases must follow OSS community norms. They must be easily and in an automated way be compiled/built, tested, and run using widely-available tools and libraries. It must be possible to separately and quickly update libraries and other dependencies (instead of them being embedded and difficult to update).*
4. *The program office and/or contractor should deploy, as data repositories (DFARS 227.7108), collaboration sites where all software source code, informal design information (e.g., emails, wikis, chat logs), bug tracker, formal design documents, and executable code will be accessible to the government and/or whomever the government see fit to grant access. Performer should detail:*
    1. *What Integrated Development Environments (IDE) used*
    2. *Communities to engage with and/or create*
    3. *Potential export control (ITAR/EAR) and classification issues with release of software source code and data*
    4. *Explanation of how government and other personnel will be provided continuous remote network access to current work in progress (and not just final results)*
5. *How and where does the performer plan to transition these technologies and software programs after the period of performance?*
6. *The offeror should identify several of the most relevant recent examples in which they used OTD practices to maintain software, and show how its experiences and successes demonstrate that it will be capable of implementing OTD on this project.*

3.2.6 Deliverables

Example Text:

1. *Intellectual Rights*
    1. *The government must receive unlimited data rights for all software source code developed at taxpayer expense unless the offeror has received written permission otherwise.*
    2. *The government must receive OSS licenses or unlimited rights for all deliverables unless the offeror has received written permission otherwise. Where the software is OSS licensed:*
        1. *All OSS licenses must be legally compatible in the way they are used.*
        2. *All OSS licenses must be certified as OSS licenses by the OSI and as Free Software licenses by the FSF.*
2. *Data Formats, Standards & Interfaces*
    1. *A demonstration of compliance with key open standards is required. Where possible, this shall be demonstrated by replacing each component that implements an open standard with an independent implementation that also implements the open standard. For example, if a web*

application is built on open standards such as HTTP, HTML, and CSS, it must be demonstrated with alternative common web browsers which implement those standards.

2. *Verify that Data Formats, Standards & Interfaces that are unencumbered (e.g., by royalty-bearing patents) and openly accessible.*
3. *If new standards have been created, verify that they are being maintained by an independent body.*

3. *Verify that data repositories are enabling community development. Verify that any export-controlled or classified information is properly marked and protected.*

3.3 Source Selection: Evaluating Proposals

The DoD 5000 series is very clear about how proposal efforts are to be run and scored; this section makes recommendations only.

3.3.1 Evaluate how well proposal responds to RFP

The evaluation should consider and score how well the proposal responded to the RFP, including the areas listed above:

1. Does it meet the overall objectives?
2. Intellectual Rights
3. Data Formats, Standards & Interfaces
4. Off-the-Shelf (OTS) Technologies
5. Open Technology Development Practices

When making source selections it is key to develop criteria that are unambiguous and easy to score. For example:

1. Has the contractor listed what standards they plan to use, and are they appropriate?
2. Is a coherent and workable technical community strategy laid out?
3. Are the intellectual rights, formats/interfaces, and OTS components for the resulting works clearly described?

3.3.2 Acceptance/Approval Criteria for Deliverables

Even when the government and a contractor reach an agreement that includes unlimited rights, modular architecture, open standards and interfaces, and a community strategy, it is still possible to lose the benefits of OTD if the contractual deliverables have pieces missing. If a vendor deploys the software as binaries on a military system, the government cannot verify that all the source code the government paid for was actually compiled into the final deliverable. Government customers should adopt a "trust but verify" approach to compiling government-funded software by insisting that all source code be delivered and compiled on the computing system where it is meant to run. Compiling software on the target system should be a condition of contractual acceptance for the deliverable. Likewise, delivery of an archived copy of all formal and informal design documents should be a condition of contractual acceptance of the deliverable. These elements include:

1. Architecture and system schematics and documentation

2. All requisite software libraries
3. Documentation of language, development environment, and compiler versions used
4. Forum and mailing list discussions and chat logs
5. Bug tracking reports
6. Version control logs

All OTD elements in the project proposal should be enumerated as separate tasks in the contract, and documentation created and related to those elements must be delivered before the system is deemed approved and accepted for contract and payment purposes.

3.3.3 Pitfalls to avoid

Watch for the following pitfalls:

1. Co-mingling government-funded software with privately-funded software (especially if it is patented), such that the government does not received unlimited rights.
2. Incorporating proprietary (especially non-OTS) components that incur licensing fees, especially if the system is designed to depend these components.
3. Co-mingling export-controlled and classified software with other software. Developers should instead devise a "plug-in" architecture (where possible) that allows use and sharing of software not restricted by export controls or classification.
4. Failure to include elements necessary to compile the source code on the target system (hence the requirement that deliverables be delivered as source code and compiled on the target system).
5. Lack of planning for management of the project's community and maintenance of source code as an O&M element. Maintenance and stewardship of a community around a particular software component can deliver great benefits (i.e. improvements in security and interoperability as technologies evolve inside and outside the government) at marginal cost. But someone, preferably a senior developer, needs to dedicate time to this task, and that is more likely to happen if the community manager is compensated for this work. It is legitimate for a vendor to object to the open-ended task of managing a multi-developer open source or GOTS developer community as a volunteer activity, even though it may a long-run benefit for their business. Likewise, it is naïve and optimistic for the government to assume that smart stewardship and continued maintenance and updating of an OTD project, particularly a GOTS (vs. public OSS) project, will magically happen without resources dedicated to the task. Program managers should identify resources to employ even a fraction of a full time equivalent to maintain OTD communities after projects are delivered. If program funds cannot be made available for O&M (vs. development) as a matter of policy, program managers should make sure that OTD community management is built into the transition plan and that cost (i.e. the community manager's time plus the cost of maintaining collaboration infrastructure) is built into the O&M budget for the deployed capability. Use of OSS collaborative infrastructure, vs. proprietary hosted services, may reduce O&M costs for OTD projects in the operations and maintenance phase, ensuring that O&M funds are used to retain the time and attention of a senior developer or community manager who keeps the project energized.

Chapter 4. Continuous Development & Delivery

Once a military capability is up and running, the project isn't complete. In fact, its only starting. The operations and maintenance phase of software is characterized by constant changes in the codebase, for any number of reasons: hardware changes, software bug fixes, updates and changes, IT infrastructure changes, bandwidth and updates in warfighter needs. Contractors and government personnel will change as well.

Ultimately, if the steps below are followed, the program should be in a good position to rapidly upgrade the software capability to stay current with new threats and technological change.

4.1 Rapid Development Cycles

Historically, government software projects result in lengthy requirements reviews followed by task breakdowns, waterfall schedules, testing and evaluation, and formal delivery. It is not unusual for these cycles to result in minimum delivery cycles of a year or more – with little flexibility for adopting new functionality or technologies within the cycle.

In sharp contrast, most open source software projects are in a constant state of evolution with very short build and test cycles. Best of breed practices are quickly adopted across established open source projects. In many cases, this has established de facto standards for software development projects.

It is typical for open source projects to version their formal releases with even numbering – e.g. 1.8.2. The frequency of formal builds and releases varies by project – but typically is planned on quarterly, semi-annual, or an annual basis. By convention, development releases – sometimes named unstable releases, are released in between for community testing and comment. Development releases, i.e. informal releases, typically have odd version numbers – e.g. 1.7.0. For government programs, it typically will make sense to deliver formal releases.

4.2 Testing, Certification and Accreditation

OTD software must meet the same standards for certification and accreditation as closed or proprietary software. The advantage to OTD is that components may have already been pre-certified or incorporated into certified and accredited systems. Particularly in service-oriented architectures, pre-existing component or software-service certification reduces the size of the new code-base that has to be certified and accredited.

Most open source projects now support unit testing and automated nightly builds with integrated testing. Problems are thus discovered and repaired rapidly. Development and deployment of software solutions is accelerated by integrated unit testing, build, and packaging flows. It will become increasingly important to certify development, testing, and distribution processes versus specific software releases. Program management should be encouraged to actively participate with outside open source groups by developing and promoting unit testing.

The challenge for a program manager whose software capability is continually being evolved by a community is: how and when do you identify a particular version of the capability as an official release for C&A purposes, and what is the frequency of those releases. It is critical to maintain awareness of both end-user needs (i.e. how badly do users in the field want or need the latest version of the software) and significant events on the development side (e.g. a vulnerability has been identified and addressed).

Likewise, it is important to build and maintain a relationship with whomever is certifying and accrediting the system, so they know in advance how large or small the next-version changes are going

to be. It is highly advisable to recruit certification and accreditation experts and C&A "old hands" into the developer community to provide feedback on proposed changes and fixes. The C&A authority for an OTD system should have full visibility into the collaboration infrastructure, either as a passive viewer or as an active participant during the C&A process, defining issues and C&A criteria that need to be addressed.

4.3 Transition to Operations & Maintenance

A common entry point for open source software technologies is through laboratories, projects, and prototypes. Typically, the configuration management and certification processes of these projects are less stringent than critical mission operational systems. Transition of these capabilities and solutions to operations is always a challenge – for both open and proprietary technologies. It is wise to begin transition planning from the beginning of any project. Robust documentation of functionality, administration guides, and the configuration management, testing and validation process, as well as documentation of the governance of the project, will help smooth the path towards operations. Most projects fail due to a lack of a formal transition plan. Recently, the government has begun to formally identify transition managers early on in the process.

The government has well-worn procedures for requirements, reviews, and contractor support for the technologies it acquires. Integrators, subject matter experts, and contractors supporting open technologies can simply be thought of as COTS providers. Projects will typically need to budget and fund operations and maintenance support and professional services for maintenance of an OTD software capability and its community.

4.4 Findability

How do you find things? In this day and age if your software program and/or project isn't "findable" it doesn't exist.

Findability is defined as the ease at which something on the world wide web can be found, i.e., is it easily searchable and findable. [Morville2005]

This goes for government programs. For a variety of reasons, the government and military have a tenancy to spend scarce resources and time either rebuilding or recreating pieces or all of existing technologies.

Military programs also can be so focused on meeting their requirements that making others aware of the project not specifically identified as a customer are ignored. When in fact every effort should be made to be inclusive and make others aware of the capabilities under development.

This is a very pragmatic step because it can lead to additional funds being put toward the program and can increase the size of the user base served by the military program office.

A few simple rules to make a military project "findable":

1. World Wide Web: Create a website listing the capability being developed. This website could be created by a contractor or the program office.
2. Use the Defense Technical Information Center (http://www.dtic.mil/dtic/) to publicize work.
3. Put out formal press releases.
4. Social media: if the program is going to be around a long time use social media tools.

5. Intelink: create a page on the Intelligent Communities Intellipedia service
6. Attend, speak and present at military and non-military conferences

4.5 Lessons Learned

Another key point about being open is to heavily document lessons learned so that other may follow success and more importantly avoid common pitfalls. Being frank, open and honest about what worked and didn't work is a cornerstone of military operations, we need to ensure that we have the same level of trust and honesty in software deployment and acquisitions.

Also be vocal about successes, publishing lessons learned in industry trade journals and conference proceedings.

4.6 OTD Success Checklist

The following is a suggested checklist for an OTD development projects:

- ✓ Community first, technology second. Often the military will focus on creating technology solutions when stakeholders aren't onboard or are non-existent. Technologist must be controlled and not allowed to build and/or deploy software until a community and user base is identified.
- ✓ Default to open, closed only when required.
- ✓ Your program is not special. Yes, the military has special warfighting needs and capabilities, but (especially in IT) we are not. Search for existing IT projects and industries and use their solutions.
- ✓ Set simple rules about how to share and how to access GOTS. Creating a software source code sharing scheme that works and is to the governments advantage is important.
- ✓ Intellectual rights . Using open source software licenses greatly simplify rights management for the government.
- ✓ Negotiate and demand unlimited rights in software and source code. Government purpose rights are basically crippled license scheme that should be avoided.
- ✓ Do not create new software licenses, use existing licenses – they are understood in commercial industry and have been approved by corporate counsels.
- ✓ Greatly limit co-mingling of government-funded software with privately-funded software (especially if it is patented). If co-mingling is required develop in a modular fashion and require unlimited rights.
- ✓ Co-mingling export-controlled and classified software with other software. Developers should instead devise a “plug-in” architecture (where possible) that allows use and sharing of software not restricted by export controls or classification.
- ✓ Limit incorporating proprietary (especially non-OTS) components that incur licensing fees, especially if the system is designed to depend these components.
- ✓ Plan and fund management of the project's community and maintenance of source code as an O&M transition element.

- ✓ Continue and encourage vigorous debates and discussion about the software among users, and among developers.
- ✓ Do not forget to document. This includes user, installation, administration, and design documentation. It also includes guidelines on how to install and maintain the software in a way that provides security.
- ✓ Configuration management: Information is maintained about what external tools/evaluations have been run, on what version of the software, and what the results were.
- ✓ Run the project as a guild: users occasionally become developers, at least for small defects.

Appendix A: U.S. Government Policy & Guidance on Openness

This appendix presents pertinent U.S. Government guidance and policies related to Open Technology Development (OTD).

OTD is an approach to software/system development in which developers (who need not be in the same organization) collaboratively develop and maintain software or a system in a decentralized fashion. OTD is a way to reduce the rising cost of warfare systems and speed the delivery of new capabilities. OTD also increases opportunities for competition and innovation, enabling rapidly fielded and upgradeable systems while optimizing software asset reuse. The 2006 OTD Roadmap plan [OTD 2006] states that OTD combines:

1. Open standards and interfaces (including open data formats)
2. Open Source Software (OSS) and designs
3. Collaborative/distributive culture and online support tools
4. Technological agility (including open systems/open architecture)

Software acquisitions and development processes should ***use***, ***extend***, and ***create*** each of these key elements to achieve open technology development:

OTD Key Element	Use	Extend	Create
1. Open standards and interfaces (including open data formats)	Favor the use of open standards/ interface/ data formats over proprietary ones; test to ensure alternative implementations can be used	Submit proposed changes to open standards' maintainers where standards do not meet requirements	Initiate new open standards specifications where appropriate. For example, help develop a specification that can be used as a "base document"
2. Open Source Software (OSS) and designs	Use extant OSS	Modify OSS (e.g., fix defects or add capabilities) without forking the code base	Initiate new OSS projects
3. Collaborative/ distributive culture and online support tools	Participate in existing community's use of collaborative project management tools	Tailor existing collaborative projects and project management tools/infrastructure (e.g., for classified software development)	Establish new cross-organization collaborative projects and project management tools/ infrastructure, but avoid duplication. This may be OGOTS or OSS
4. Technological agility (open systems/ open architecture)	Favor the use of products and services that are modular and technologically agile. Avoid products and services that are monolithic or locked into one platform	Improve modularity, platform support, and configurability of products and services in use	Create products, services, or infrastructure that (1) aids or improves technological agility and (2) is more modular

Open standards and interfaces (including open data formats) are a key mechanism for avoiding vendor lock-in and enabling collaboration. They enable collaboration because developers on either side of the interface are free to make changes as long as they comply with the relevant standards.

Open Source Software (OSS) can be used as-is, modified, or created. However, extensions of publicly-available OSS components should default to public release, to avoid incurring the massive costs and loss of innovation due to stove-piped maintenance. Creating a separate project by starting from an existing project is called "forking". Whoever starts a fork becomes responsible for a separate O&M burden, because the forked version is not compatible with the public version supported by the existing development community.

A collaborative/distributive culture with online support tools enables multi-organization collaboration. It should be relatively easy to establish a new collaborative project; in many cases these should be OSS projects to maximize the collaboration (this reduces O&M costs in the long term). However, some components must not be released to the public (typically these would be classified and/or be subject to export controls), or the government may not have the necessary data rights. In the last case, the government should at least strive to have Government Purpose Rights (GPR) so that collaboration and wide use inside the government can occur.

Technological agility is important, and this is greatly enabled by the modularity that comes from an open systems/open architecture approach. In an open systems/open architecture approach, systems are broken down into modular components, and open standards and interfaces are used to connect the components. The interfaces of many components will probably not be solely defined by open standards, but where there are open standards they should be used, and where there is no appropriate open standard, the interface should be documented and there must be adequate rights to ensure that others could re-implement the component without restriction (e.g., by royalty payments).

The following sections delve into each area and identify some of the government policies about them.

A.1 Open Standards and Interfaces (including open data formats)

The federal government and DoD have recognized the value of open non-proprietary data standards and have set policies and regulations accordingly. This section lays out some of the current government and DoD policies around information technology (IT) standards, and issues related to them.

However, a few general notes are in order:

1. Remember that standards include both interfaces and data formats; do not forget the need for data format standards. It is easy for the government to end up in a circumstance where it has the data, but the government is effectively locked into a specific vendor, because data is locked into a specific proprietary application that is necessary to view, use or update that data.
2. Beware of patent issues in standards. Any standards that are used in a DoD or federal project should be evaluated to ensure that the standards it uses is unencumbered from an intellectual rights perspective (i.e., that they are available for anyone to use or re-implement, without encumbrances such as patent royalty payments). Beware that some standards bodies claim that an "open" standard is merely one where the process to reach consensus was open; this can lead to standards that require licensing agreements and substantial payments to companies to meet the so-called "open" standards. In this document, a "voluntary consensus standard" is developed by consensus, and may or may not require a royalty payment. An "open standard" is a voluntary consensus standard that does *not* impose royalty payments or other restrictions on its use.

3. Similarly, beware of "de facto standards" that are not specifications at all, or where the published specifications have little relationship to the interfaces or formats that are actually used.

A.1.1 U.S. Government

A.1.1.1 Voluntary Consensus Standards

U.S. federal government agencies must use "voluntary consensus standards" in their procurements. As described in Office of Budget & Management (OMB) Circular No. A-119, "Federal Participation in the Development and Use of Voluntary Consensus Standards and in Conformity Assessment Activities (February 10, 1998, http://www.whitehouse.gov/omb/circulars/a119/a119.html):

> "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, your 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)."

This document also explains why the government should use 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:
>
> a) 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.
>
> b) Provide incentives and opportunities to establish standards that serve national needs.
>
> c) Encourage long-term growth for U.S. enterprises and promote efficiency and economic competition through harmonization of standards.
>
> d) Further the policy of reliance upon the private sector to supply Government needs for goods and services.

The OMB policy also defines the term "voluntary consensus standards":

a) For purposes of this policy, "voluntary consensus standards" are standards developed or adopted by voluntary consensus standards bodies, both domestic and international. These standards include provisions requiring that owners of relevant intellectual property have agreed to make that intellectual property available on a non-discriminatory, royalty-free or reasonable royalty basis to all interested parties. For purposes of this Circular, "technical standards that are developed or adopted by voluntary consensus standard bodies" is an equivalent term.

    1. "Voluntary consensus standards bodies" are domestic or international organizations which plan, develop, establish, or coordinate voluntary consensus standards using agreed-upon procedures. For purposes of this Circular, "voluntary, private sector, consensus standards bodies," as cited in Act, is an equivalent term. The Act and the Circular encourage the participation of federal representatives in these bodies to increase the

likelihood that the standards they develop will meet both public and private sector needs. A voluntary consensus standards body is defined by the following attributes:

i. Openness.

ii. Balance of interest.

iii. Due process.

iv. An appeals process.

2. Consensus, which is defined as general agreement, but not necessarily unanimity, and includes a process for attempting to resolve objections by interested parties, as long as all comments have been fairly considered, each objector is advised of the disposition of his or her objection(s) and the reasons why, and the consensus body members are given an opportunity to change their votes after reviewing the comments.

b) Other types of standards, which are distinct from voluntary consensus standards, are the following:

1. ”Non-consensus standards,” “Industry standards,” “Company standards,” or “de facto standards,” which are developed in the private sector but not in the full consensus process.
2. “Government-unique standards,” which are developed by the government for its own uses.
3. Standards mandated by law, such as those contained in the United States Pharmacopeia and the National Formulary, as referenced in 21 U.S.C. 351.

Note that the OMB policy does permit the use of standards where there is a “reasonable royalty basis”, but implementers should wary of such non-open standards. Such royalty payments prohibit the use of many OSS products (and thus become a barrier to competition), and can often make their use uneconomical.

A.1.1.2 Tagging (Schedule 70)

In order for agency buyers to easily select products and services supporting open standards, vendors need a way to tag each available item as to which open standards it supports. This tagging of products and services in the software market is much the same idea as the kind of “plug-compatibility” that buyers have long depended on when buying hardware components.

In 2005, a change to the guidance on government contracts specified how vendors should tag their products so that buyers can easily check what interoperable standards are supported. The change is in a “NOTE to Offers” in a major source of product catalogs for commercial off-the-shelf software, “Schedule 70”.

Schedule 70, managed by the General Services Administration, encompassed more than 4,000 negotiated contracts in 2004, and accounted for about $11 billion in information technology sales to the government. Also, Schedule 70 contracts may be used by State, Tribal, local, or regional government entities, and by any local educational agency or institution of higher learning. This fact can be very important in that U.S. Federal systems often interoperate not only across agencies but with other levels of government, e.g., in the context of the National Spatial Data Infrastructure.

Offerors are encouraged to identify within their software items any component interfaces that support open standard interoperability. An item’s interface may be identified as interoperable on the basis of

participation in a Government agency-sponsored program or in an independent organization program. Interfaces may be identified by reference to an interface registered in the component registry located at http://www.core.gov.

A.1.2 The Department of Defense

In addition to U.S. Federal Government guidance, DoD has further defined standards policy. Here are seven documents that provide additional information about standards within DoD[1]:

1. The DoD Information Technology Standards Program (ITSP) Management Plan, 19 January 2007
2. OSD Memo: Subject: Department of Defense Information Technology Standards Program Executive Agent Management Plan, 19 Jan 2007
3. OSD Memo, Subject: DoD Executive Agent (EA) for Information Technology (IT) Standards, 21 May 2007
4. CJCSI 6212.01D, Interoperability and Supportability of Information Technology and National Security Systems, 8 March 2006
5. DoD Directive 5101.7 dated May 21, 2004, Subject: DoD Executive Agent for Information Technology Standards
6. DoD Instruction 4630.8 dated, June 30, 2004, Subject: Procedures for Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)
7. DoD Directive (DoDD) 4630.05 (formerly 4630.5), Subject: Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS), May 5, 2004

Of these seven documents, the two that give the most detail about the DISA standards repository (the DoD IT Standards Registry - DISR, see detail below) and the use of mandated standards are #1 and #4.

The other five memos, DoD Directives, and DoD Instructions are more general in nature, but they do refer to the DISR and define the various responsibilities of the DoD components in the standards management and implementation process.

DOD Directive 5101.7 makes DISA the executive agent for IT standards; DISA's responsibilities are (among other things) to:

> 6.1.10. Promote adoption of selected non-Government (national and international) standards as Federal standards, in coordination with the National Institute of Standards and Technology (NIST). Manage the process for coordinating DoD positions on IT standards with the NIST and other Federal Agencies. Facilitate the harmonization and consolidation of IT standards agreements and Memoranda of Understanding/Memoranda of Agreement on behalf of the Department of Defense for the purpose of nation-to-nation and multinational systems interoperability. Encourage the use of Government-Off-The-Shelf, Non-Developmental Items, and Commercial-Off-The-Shelf products wherever possible in accordance with reference (e).
>
> 6.1.11. Develop, with the Heads of the DoD Components, IT military standards under DISA's purview only when non-Government standards or Federal standards fail to meet DoD requirements.

[1]This information from DISA GE331, ITSC Collaboration TWG.

The DoD Executive Agent for IT Standards shall conduct validation testing on all standards for which they are the Standards Development Organization or Standards Setting Organization.

6.1.12. Maintain the DoD IT Standards Registry (DISR) consisting of approved IT standards and standards profiles to aid program and project managers, acquisition authorities, and systems and technical architects in the development and fielding of interoperable and net-centric enabled systems and products. Establish process and procedures for life-cycle configuration management of IT standards contained in the DISR. Provide on-line Non-Classified"

The DoD IT Standards Registry (DISR) maintained by DISA is currently only available to DoD and government personnel; those authorized to access it can view it at: https://disronline.disa.mil. The DISR is an online repository for a minimal set of primarily commercial IT standards (formerly captured in the Joint Technical Architecture (JTA)). These standards are used as the "building codes" for all systems being procured in the Department of Defense. When building systems, requests for proposals and contract statements of work should be reviewed as part of approved acquisition processes to ensure IT standards established in Initial Capabilities Documents, Capability Development Documents, and Capability Production Documents are translated into clear contractual requirements. In addition, requests for proposals and contract statements of work should contain additional requirements for contractors to identify instances where cost, schedule, or performance impacts may preclude the use of IT standards mandated in DISR. To insure interoperability among systems, DODD 4630.05 requires contractors to use only approved standards from the DISR for their systems.

There is a Cross-Service effort called NESI (Net-Centric Enterprise Solutions for Interoperability)[2] between Air Force (ESC), Navy (PEO C4I), and Defense Information Systems Agency (DISA). NESI is developing a body of architectural and engineering knowledge that guides the design, implementation, maintenance, evolution, and use of the IT portion of net-centric solutions for military application. NESI provides specific technical recommendations that a DoD organization can use as references. NESI also serves as a reference set of compliant instantiations of these directives.

A.2 Open Source Software (OSS)

The use of open source software within the Federal government and the DoD specifically has been codified by the following policy documents:

1. Office of Management and Budget (OMB) M-04-16's "Software Acquisition" (http://www.whitehouse.gov/omb/memoranda/fy04/m04-16.html). This memo simply states that the existing federal policies on software acquisition apply equally to both proprietary and open source software. Note that there is *no* preference for proprietary software over OSS.
2. DoD CIO's "Clarifying Guidance Regarding Open Source Software (OSS)" of October 2009 (http://cio-nii.defense.gov/docs/OpenSourceInDoD.pdf). This policy memo specifically notes that "There are positive aspects of OSS that should be considered when conducting market research on software for DoD use". It even states conditions under which "Software items, including code fixes and enhancements, developed for the Government should be released to the public (such as under an open source license)". This memo supersedes the previous memorandum, "Open Source Software (OSS) in the Department of Defense (DoD)", May 28, 2003.

[2] http://nesipublic.spawar.navy.mil/docs

The following documents provide important guidance about OSS for the federal government and DoD (respectively):

1. CENDI's "Frequently Asked Questions about Copyright and Computer Software Issues Affecting the U.S. Government with Special Emphasis on Open Source Software" (http://www.cendi.gov/publications/09-1FAQ_OpenSourceSoftware_FINAL_110109.pdf)
2. DoD CIO's "DoD Open Source Software (OSS) FAQ: Frequently Asked Questions regarding Open Source Software (OSS) and the Department of Defense (DoD)" (http://cio-nii.defense.gov/sites/oss/Open_Source_Software_%28OSS%29_FAQ.htm).

For general policy or legal questions about OSS in the U.S. government, consult these policy and guidance documents.

All of these policies and guidance documents indicate that while a specific OSS product may or may not be appropriate for a particular purpose, just being OSS is not a problem at all under U.S. federal and DoD acquisition rules. All of these policies and guidelines make it clear that OSS is acceptable to use in software acquisitions.

Two issues involving OSS are especially important in government acquisition: OSS is commercial, and OSS is *not* prohibited by information assurance controls on freeware, shareware, and/or warranty-less software.

A.2.1 Nearly all OSS is Commercial and Commercial-off-the-shelf (COTS)

Nearly all publicly-available open source software (OSS) is a "commercial item" and is "commercial off-the-shelf (COTS)".

This is important because many laws and regulations have far-reaching requirements about commercial items and COTS items. For example, U.S. law (10 USC 2377) establishes a "preference for acquisition of commercial items", and the FAR requires government agencies to "(a) Conduct market research to determine [if] commercial items or non-developmental items are available … (b) Acquire [them] when… available … [and] (c) Require prime contractors and subcontractors at all tiers to incorporate, to the maximum extent practicable, [them] as components..." An agency that fails to consider OSS options would be in direct violation of these laws and regulations.

In particular, OSS that has at least one non-governmental use, and has been or is available to the public, is by definition commercial software. If OSS is already available to the public and is used unchanged, it is usually commercial-off-the-shelf (COTS). There are many other conditions for being a commercial item or COTS, and publicly-available OSS tends to meet them. This is true by U.S. law, U.S. acquisition regulations (specifically the Federal Acquisition Regulation (FAR) and the Defense Federal Acquisition Regulation Supplement (DFARS)), and by policy.

U.S. law governing federal procurement (U.S. Code Title 41, Chapter 7, Section 403) defines "commercial item" as including "Any item, other than real property, that is of a type customarily used by the general public or by non-governmental entities for purposes other than governmental purposes (i.e., it has some non-government use), and (i) Has been sold, leased, or licensed to the general public; or (ii) Has been offered for sale, lease, or license to the general public ...". Thus, as long as the software has at least one non-governmental use, software released (or offered for release) to the public is a commercial item for procurement purposes.

Similarly, U.S. Code Title 41, Chapter 7, Section 431 defines the term "Commercially available off-the-shelf (COTS) item"; software is COTS if it is (a) a "commercial item", (b) sold in substantial quantities in the commercial marketplace, and (c) is offered to the Government, without modification, in the same form in which it is sold in the commercial marketplace. Thus, OSS available to the public and used unchanged is normally COTS.

Finally, U.S. Code Title 17, section 101 defines "financial gain" as including "receipt, or expectation of receipt, of anything of value, including the receipt of other copyrighted works". Most OSS projects are specifically established to encourage others to contribute improvements (which are copyrighted works), a form of financial gain and thus commercial.

These definitions in U.S. law govern U.S. acquisition regulations, namely the Federal Acquisition Regulation (FAR) and the Defense Federal Acquisition Regulation Supplement (DFARS). For example, DFARS 252.227-7014 Rights in Noncommercial Computer Software and Noncommercial Computer Software Documentation similarly defines "Commercial computer software" as "software developed or regularly used for non-governmental purposes which: (i) Has been sold, leased, or licensed to the public; (ii) Has been offered for sale, lease, or license to the public; (iii) Has not been offered, sold, leased, or licensed to the public but will be available for commercial sale, lease, or license in time to satisfy the delivery requirements of this contract; or (iv) Satisfies a criterion expressed in paragraph (a)(1)(i), (ii), or (iii) of this clause and would require only minor modification to meet the requirements of this contract."

Although OMB memorandum M-04-16 doesn't specifically state that OSS is typically commercial software, OMB Memo M-03-14 "Reducing Cost and Improving Quality in Federal Purchases of Commercial Software" specifically says that its SmartBuy initiative for commercial software includes "open source software support". Thus, is wrong to presume that OSS is not commercial.

The DoD CIO October 2009 memo "Clarifying Guidance Regarding Open Source Software (OSS)" specifically notes that in "almost all cases, OSS meets the definition of 'commercial computer software' and shall be given appropriate statutory preference in accordance with 10 USC 2377 (reference (b)) (see also FAR 2.101(b), 12.000, 12.101 (reference (c)); and DFARS 212.212, and 252.227-7014(a)(1) (reference (d)))".

This misunderstanding became so common that on June 5, 2007, the Department of the Navy issued a memo titled "Open source Software Guidance". This memo notes that "misconceptions about whether or not open source software qualifies as COTS (commercial off-the-shelf) or GOTS (government off-the-shelf) software has hindered the Navy's ability to fully utilize open source software. Because of this misconception, OSS has not received equal consideration during the software acquisition process. [The Department of the Navy] will treat OSS as COTS when it meets the definition of a commercial item".

Additional rationale explaining why OSS is commercial is given in "Free-Libre / Open Source Software (FLOSS) is Commercial Software".[3]

[3]Wheeler, David A. "Free-Libre / Open Source Software (FLOSS) is Commercial Software" http://www.dwheeler.com/essays/commercial-floss.html

A.2.2 OSS is not Inhibited by Information Controls on Freeware, Shareware and Warranties

One misinterpretation of OSS is important to dispel. There are both federal and DoD information assurance (IA) controls regarding freeware, shareware, and warranty-less software, but these controls do not apply to OSS. Part of the problem is that many people refer, incorrectly, to OSS as 'freeware' or 'shareware'. In addition, many people do not understand the federal and DoD policy on software warranties. As a result, they incorrectly believe that these policies inhibit the use of OSS.

The controls that are misinterpreted this way are:

1. NIST Special Publication 800-53, "Recommended Security Controls for Federal Information Systems and Organizations" revision 3 (http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final_updated-errata_05-01-2010.pdf), control SA-6 aka "Software Usage Restrictions". This control says:
   "[An organization must prohibit] the use of binary or machine executable code from sources with limited or no warranty without accompanying source code". The enhancement supplemental guidance for this control states that, "Software products without accompanying source code from sources with limited or no warranty are assessed for potential security impacts. The assessment addresses the fact that these types of software products are difficult or impossible to review, repair, or extend, given that the organization does not have access to the original source code and there is no owner who could make such repairs on behalf of the organization."
2. DoD Instruction 8500.2 "Information Assurance (IA) Implementation" (http://www.dtic.mil/whs/directives/corres/pdf/850002p.pdf) dated February 6, 2003, control DCPD-1 aka "Public Domain Software Controls". This control says:

   "Binary or machine executable public domain software products and other software products with limited or no warranty, such as those commonly known as freeware or shareware are not used in DoD information systems unless they are necessary for mission accomplishment and there are no alternative IT solutions available. Such products are assessed for information assurance impacts, and approved for use by the DAA. The assessment addresses the fact that such software products are difficult or impossible to review, repair, or extend, given that the Government does not have access to the original source code and there is no owner who could make such repairs on behalf of the Government."

But careful reading shows that these controls do not apply to OSS. Both the federal and defense controls restrict the use of software where the government cannot "review, repair or extend, given that the Government does not have access to the original source code" *and* there is limited warranty. By definition, the government *does* have access to the source code of OSS, so this entire control does not apply to OSS. Although the terms "freeware" and "shareware" are not explicitly defined by DCPD-1, note that the text states that it is a *fact* that the government does not have access to the original source code, so therefore OSS is not freeware nor shareware.

It's important to understand this control in context. Both controls fundamentally require *either* source code (to review, repair, or extend the software) *or* a warranty; the idea is that the government wants to ensure that it can either self-support software it needs, or be able to depend on others to do so.

The 2009 DoD memo on OSS specifically notes in item (c) that DCPD-1 ("Public Domain Software Controls") "should not be interpreted as forbidding the use of OSS, as the source code is available for review, repair and extension by the government and its contractors".

A.3 Collaborative/ Distributive culture and online support tools

Historically, collaborative/distributive development of software and systems has been relatively rare in the federal and DoD communities. This document aims to change that. Still, there have been some instances of it, and of tools and approaches that enable it.

Still, the U.S. government has sometimes become involved in establishing or becoming involved in distributive development. This is particularly obvious in open source software (OSS) projects. The government has sometimes established or participated in OSS projects that began or were transformed into community projects. Examples include Expect and Security-Enhanced Linux. See the previous section on OSS for more.

The data rights the U.S. government receives by default typically enable collaborative development, even though this opportunity is often squandered. In many cases, the government receives unlimited rights, which enables the government to establish collaborative development with anyone it wishes. In other cases, the U.S. government at least receives Government Purpose Rights (GPR); these are enough rights to enable collaborative development by the government and its contractors (at all tiers), when the use is for U.S. government purposes.

In 2006 the Navy OA Office set up a software repository called SHARE[18] where contractors who are working on a Navy contract can access the repository to deposit and browse source code.[4] More recently, DISA stood up forge.mil (http://www.forge.mil) to enable "the collaborative development and use of open source and DoD community source software" (the latter is what this document refers to as OGOTS software).

A.4 Technological Agility (Open Systems and Open Architecture)

All too often software and systems have historically been developed as monoliths. Such software or systems are difficult to change and adapt as technology and needs change.

As software has become increasingly networked, design and engineering methodologies have evolved towards services-based architectures that communicate through open and standardized interfaces. Once this type of open, service-based architecture is implemented, the system naturally decomposes into a modular design — each service is free to improve and evolve independently as long as it communicates through the standard interfaces.

The key is to divide the system into smaller, modular components that can be used separately and have standard interfaces. This allows change and adaptation as technology and needs change. The government must also have the necessary rights to adapt or replace these components and their interconnections as necessary.

In this context, any given software service may be implemented by Commercial off-the-shelf (COTS) or Government off-the-shelf (GOTS) — the best implementation can be chosen, modified or replaced if a better technological option is available. Properly implemented, open standards and solutions create a level playing field that allows the underlying technologies to evolve while minimizing interface complexity. In addition, the modularity afforded by open standards and interfaces reduces technological risk by eliminating cascading software dependencies, and reduces financial risk by eliminating the need to re-engineer or re-integrate the entire system when new capabilities or requirements are introduced.

[4]See PEO-IWS Library, Software, Hardware Asset Reuse Enterprise (SHARE), https://viewnet.nswc.navy.mil

DoD Directive 5000.1 (dated May 12, 2003) paragraph E1.27 states that, "A modular, open systems approach shall be employed, where feasible." An open system is a system "that employs modular design, uses widely supported and consensus based standards for its key interfaces, and has been subjected to successful validation and verification tests to ensure the openness of its key interfaces".[5]

[5]http://www.acq.osd.mil/osjtf/whatisos.html

Appendix B: Legal requirements for OSS release to the public by government or contractors

This section summarizes when the U.S. federal government or its contractors may publicly release, as open source software (OSS), software developed with government funds. This section is intended for non-lawyers, to help them understand the basic rules they must follow.

Before going further, a few definitions and warnings are necessary. In this section, the term "government" means the U.S. federal government. "You" means the government organization or contractor who wants to release software to the public as OSS. "Releasing to the public as OSS" means (1) releasing the software source code to the general public (such as through a public website) ***and*** (2) giving its users the freedom to use it (for *any* purpose), study it, modify it, and redistribute it (modified or not)[6]. Note that these freedoms can be given by releasing the software under an OSS license,[7] or by releasing it without any copyright protection. This section is not legal advice, and variations of specific facts can produce different results. Also, note that government contracting is *very* different from commercial practices; do not presume that commercial practices apply.

To determine if you can release to the public some software developed with government funds as OSS, you must answer the following questions:

1. **What contract applies, what are its terms, and what decisions have been made?**
2. **Do you have the necessary copyright-related rights?**
3. **Do you have the necessary other intellectual rights[8] (e.g., patents)?**
4. **Do you have permission to release to the public (e.g., classification and export control)?**
5. **Do you have the materials (e.g., source code) and are all materials properly marked?**

Each of these questions is explained below, followed by a discussion of who can make certain decisions.

1. What contract applies, what are its terms, and what decisions have been made?

First, find the contract and find what terms apply, particularly which data rights clauses apply. Most contracts use one of a small set of standard data rights clauses, but you need to find out *which* clauses apply, and if the contract grants exceptions. If the clause text is different (e.g., older) than the clauses discussed here, or makes an exception, then the contract (if legal) governs. Also, determine what data rights decisions have been made by the contracting officer.

2. Do you have the necessary copyright-related rights?

The following table shows the default copyright-related rights for common circumstances. The first row is a special case, where a federal employee develops the software as part of his or her official duties. Later rows discuss the typical impact of common data rights clauses from the Federal Acquisition Regulation (FAR) or the Department of Defense FAR Supplement (DFARS) (but note the dates):

---

[6] This is, in summarized form, the Free Software Definition (http://www.gnu.org/philosophy/free-sw.html) from the Free Software Foundation. A similar definition is in the DoD's "Clarifying Guidance Regarding Open Source Software (OSS)" (http://cio-nii.defense.gov/sites/oss/2009OSS.pdf). A more detailed definition of OSS is the Open Source Definition (http://www.opensource.org/osd.html) from the Open Source Initiative.

[7] To release under an OSS license you must have the copyright-related rights (listed in 17 USC §106) to reproduce the work, to prepare derivative works, to distribute copies, and to permit others to perform those actions.

[8] Some lawyers use the term "intellectual property rights," but many believe this term is misleading. Intellectual works (like software) are fundamentally different from physical property, because someone can receive an intellectual work (or rights to it) without its first possessor losing the work or their rights. The authors recommend using the term "intellectual rights" instead.

Circum-stance	Other Conditions (if any)	Case	Can government release as OSS?	Can contractor release as OSS?
U.S. federal government employee (including military personnel) develops software as part of his/her official duties. This makes it a "Work of the U.S. government."		A	**Effectively yes**. The software is not subject to copyright protection in the U.S. per 17 USC §105, so if released, anyone in the U.S. can read, use, modify, and redistribute it. The government may apply for copyright outside U.S., but still release the software as OSS.	N/A
FAR 52.227-14 contract clause defaults (December 2007), software first produced in performance of contract.	Government has ***not*** granted the contractor the right to assert copyright (default).	B	**Yes**. The government normally has unlimited rights (essentially the same rights as a copyright holder) per (b)(1). In the FAR source code is software, and software is data, so source code is data.	**No**. The contractor may request permission to assert copyright.
	Government ***has*** granted the contractor the right to assert copyright (e.g., via specific written permission or via clause alternate IV).	C	**No**. The government does not have sufficient rights, per (c)(1)(iii); it cannot distribute copies to the public. The government should be wary of granting a request to assert copyright, as it permanently loses many rights to data it paid to develop.	**Yes**. The contractor may assert copyright.
DFARS 252.227-7014 contract clause defaults (June 1995).	Developed exclusively with government funds.	D	**Yes**. The government has unlimited rights (essentially the same rights as a copyright holder). Per (b)(2)(ii), the 5-year period from mixed funding can be negotiated to a different length of time, and it starts "upon execution of the contract, subcontract, letter contract (or similar contractual instrument), contract modification, or option exercise that required development of the computer software."	**Yes**. Copyright is held by the contractor/ supplier.
	Developed by mixed funding (government partly paid for its development) *and* (sub)contract execution/mod more than 5 years ago.	E		
	Developed by mixed funding (government partly paid for its development) *and* (sub)contract execution/mod less than 5 years ago.	F	**No**. The government does not have sufficient rights. Per (b)(2)(ii), the 5-year period from mixed funding can be negotiated to a different length of time; during this time the government only has "government purpose rights." If software is developed exclusively at private expense, by default the government only has "restricted rights"; the government should be wary of dependencies on such components. The government can negotiate for greater rights per (b)(3) and (b)(4).	
	Developed exclusively at private expense.	G		

Circum-stance	Other Conditions (if any)		Case	Can government release as OSS?	Can contractor release as OSS?
DFARS 252.227-7018 contract clause defaults (June 1995): Small Business Innovation Research (SBIR) Program.	Not developed exclusively at private expense.	Less than five years after *completion* of the project	H	**No**. The government does not have sufficient rights, per (b)(4)(i).	**Yes**. The contractor has copyright.
		More than five years after *completion* of the project *and* alternate I is *not* used.	I	**Yes**. The government has unlimited rights (essentially the same rights as a copyright holder) per (b)(1)(vi). Unfortunately, it is sometimes difficult to determine when the time period has expired.	
		More than five years after *completion* of the project *and* alternate I *is* used.	J	**Sometimes**. Under alternate I the Government cannot exercise its rights to release if, within certain time limits, the software is published and the contracting officer is notified. This limitation continues as long as it is reasonably available to the public for purchase (after which the government *can* release it as OSS). See alternate I for details.	
	Developed exclusively at private expense.		K	**No**. The government does not have sufficient rights, per (b)(2).	
FAR 52.227-17 "Special works" contract clause defaults (December 2007)	Government has ***not*** granted the contractor the right to assert copyright (default), and the software was first produced in perform-ance of the contract.		L	**Yes**, either through unlimited rights or by holding copyright. By default, the government has unlimited rights in all data delivered under the contract, and in all data first produced in the performance of the contract, per (b)(1)(i). Per (c)(ii), if the contractor has not been granted permission to assert copyright rights, the contracting officer can direct the contractor to assign copyright to the government.	**No**. Contractor cannot assert copyright rights per (c)(1)(i). The contractor may request permission to assert copyright; if granted see below.

Circum-stance	Other Conditions (if any)	Case	Can government release as OSS?	Can contractor release as OSS?
	Government **has** granted the contractor the right to assert copyright, and the software was first produced in perform-ance of the contract.	M	**No**. The government only has the more limited rights listed at the end of (c)(1)(i), and these rights are limited to uses "by or on behalf of the Government."	**Yes**. Contractor has copyright.
	Software not first produced in the performance of this contract.	N	**It depends.** Note that a contractor cannot include copyrighted software into a deliverable without written permission of the contracting officer, see (c)(2) for more.	**It depends.**
DFARS 252.227-7020 "Special works" contract clause defaults (June 1995).	Work first produced, created, or generated and required to be delivered under the contract.	O	**Yes**. The government receives the copyright, per (c)(2).	**No**. The government has copyright.
	Other copyrighted works incorporated into a required deliverable (unless written approval granted for an exception).	P	**Normally yes**. Per (c)(3) and (d), the contractor must normally grant to the government a long list of data rights when incorporating other copyrighted works, and these rights permit OSS release. The contractor may only incorporate software without those rights into a deliverable if the government contracting officer gives written approval, per (d).	**Normally yes.** The contractor must already have the rights for OSS release to incorporate it, unless given written approval.

These are the general rules, but you must examine your specific circumstances to determine exactly what you can do. There are details in the FAR and DFARS clauses not emphasized here, and the contract can change from these defaults to something very different. Some contracts will use different versions of the FAR and DFARS clauses, so check to see if there are any relevant differences. Note that some other agencies (like NASA) have FAR supplements, which are not covered here.

The table above *only* applies to software that was either (1) developed by a government employee as part of his or her official duties or (2) developed by a government contractor (directly or indirectly) as part of a government contract. Such software may include or depend on *other* software, such as commercial software, that does *not* meet these criteria. When a system includes commercial software, the *commercial license* applies to those components, and everyone must comply with their license terms. Commercial software includes any software that is used for at least one non-governmental use and has been sold, leased, or licensed to the general public (per 41 USC §403 and DFARS 252.227-7014(a)(1)), so nearly all publicly-available OSS is commercial software. Commercial software with minor modifications is still considered commercial software.

In many cases the contractor receives copyright. When there are multiple contractors or suppliers (e.g., a lead integrator and subcontractors), the legal arrangements between the organizations determine *which* contractors/suppliers are legally allowed to assert copyright. Lead contractors do *not* necessarily receive copyrights from their subcontractors and suppliers. Note that the government can receive and hold copyrights transferred to it, per 17 USC §105.

In many cases the government is not the copyright holder but has *unlimited rights* (see rows B, D, E, and I). If the government has unlimited rights, it has essentially the same rights as a copyright holder for purposes of releasing the software as OSS[9]. Thus, it can release the software under any OSS license it chooses, including the GNU General Public License (GPL) and Lesser GPL (LGPL)[10]. When the government has unlimited rights but is not the copyright holder, there are a few actions it cannot take, e.g., the right to transfer or assign rights, and standing to sue in court over copyright infringement[11]. However, for the purposes here these are technicalities; the key point is that the government can release the software as OSS, under any OSS license it chooses, once it receives unlimited rights.

The government should be extremely wary of receiving less than unlimited rights for software or systems it paid to develop. For example, some contractors will intentionally embed components over which they have exclusive control, and then design the rest of the system to depend on those components. When the government has less than unlimited rights, it risks creating a dependency on a contractor, rendering competition for that system meaningless[12] and in some cases putting military capability at risk.[13] [14]

Some have misunderstood U.S. law and policy as requiring the government to mindlessly accept proposals which give less than unlimited rights for systems developed though government funding. It is true that 10 U.S.C. §2320(a)(2)(F) states that "a contractor or subcontractor (or a prospective contractor

[9]The Council on Governmental Relations (CAGR)'s "Technical Data and Computer Software: A Guide to Rights and Responsibilities Under Federal Contracts, Grants and Cooperative Agreements" states that "This unlimited license enables the government to act on its own behalf and to authorize others to do the same things that it can do, thus giving the government *essentially* the same rights as the copyright owner."

[10]CENDI's "Frequently Asked Questions about Copyright and Computer Software" at http://cendi.gov/publications/09-1FAQ_OpenSourceSoftware_FINAL_110109.pdf question 4.3 says: "an agency may distribute software created by a vendor to all users under an open source licensing scheme if it acquired sufficient rights from the vendor to do so in the software. For example, an "unlimited rights license" acquired under a DFARS procurement-type contract..." Similarly, the "DoD Open Source Software (OSS) FAQ" says that once the government has unlimited rights, it can "use those rights to release that software under a variety of conditions (including an open source software license), because it has the use and modify the software at will, and has the right to authorize others to do so."

[11]The government can probably take other measures against someone who does not comply with the license, though. For example, the government may be able to sue for breach of license. Also, an infringer may lose any ability to enforce rights over the resulting work in U.S. court due to the doctrine of unclean hands.

[12]Ashton B. Carter, "Memorandum to Acquisition Professionals Subject: Better Buying Power: Mandate for Restoring Affordability and Productivity in Defense Spending" https://dap.dau.mil/policy/Documents/Policy/Carter%20Memo%20on%20Defense%20Spending%2028%20Jun%202010.pdf - His first point on providing incentives is to "Avoid directed buys and other substitutes for real competition. Use technical data packages and open systems architectures to support a continuous competitive environment."

[13]GAO GAO-06-839 "WEAPONS ACQUISITION: DOD Should Strengthen Policies for Assessing Technical Data Needs to Support Weapon Systems" (July 2006) http://www.gao.gov/new.items/d06839.pdf reported that "The lack of technical data rights has limited the services' flexibility to make changes to sustainment plans that are aimed at achieving cost savings and meeting legislative requirements regarding depot maintenance capabilities... Unless DOD assesses and secures its rights for the use of technical data early in the weapon system acquisition process when it has the greatest leverage to negotiate, DOD may face later challenges in sustaining weapon systems over their life cycle."

[14]See, for example, "Fire support's dependence on contractors," Sgt Timothy Caucutt, http://www.mca-marines.org/gazette/article/paying-pirates

or subcontractor) may not be required, as a condition of being responsive to a solicitation or as a condition for the award of a contract, to … sell or otherwise relinquish to the United States any rights in technical data [except in certain cases, and may not be required to] refrain from offering to use, or from using, an item or process to which the contractor is entitled to restrict rights in data"[15]. However, this is not the whole story. "If the Government has properly required certain data or software in a solicitation, it is entitled to certain rights in accordance with the statute and an offer failing to propose at least those rights could be held unacceptable." What is more, the government may (and should) evaluate proposals "on the basis of data rights and giving higher ratings to offerors willing to provide more than the bare minimum rights"[16]

Under many of the FAR (but not DFARS) clauses, if the government agrees to allow contractors to assert copyright, the government *loses* many of its rights, forever, to software that the government paid to develop (see rows B, C, L, and M). This loss of rights can be quite detrimental to the government. What's more, it creates a difficult decision for a contracting officer to make, as the contracting officer must anticipate all possible future uses to make a good decision (something that is difficult in practice). The usual DFARS clause (252.227-7014) avoids this problem; in this clause, typically the government ends up with unlimited rights to software it paid to develop (in some cases after a delay), and the contractor has copyright, enabling *both* parties to take actions such as releasing the software as OSS.

Here are a few notes about specific clauses:

- Under FAR 52.227-14 (rows B and C), the government can grant a contractor the right to assert copyright, at which point the contractor gains rights but the government permanently *loses* rights. Per FAR 27.404-3(a)(2), the government should grant this request *only* "when [that] will enhance appropriate dissemination or use." Government officials should *not* grant this automatically, as doing so dramatically reduces the government's rights to software that the government paid to develop. The government could choose to grant this permission on condition that the software be immediately released to the public under some specific OSS licenses (with the license agreed upon as part of the condition for release). In such a case, public release as OSS would be used as a method to enhance dissemination and use. Deliverables can include data not first produced in the performance of the contract, per (c)(2), but in this case it is not clear to this author if the government can release software as OSS.
- Under DFARS 252.227-7014 (rows D-G), the contractor normally gets copyright. The government gets the same rights as a copyright holder (via unlimited rights) if (1) the software was developed exclusively with government funding or (2) the funding was mixed and five years have passed after the relevant contract or contract modification that caused its development was signed. The government should beware of situations where the contractor attempts to deliver software that vitally depends on some component that they developed entirely at private expense. Such a dependency can inhibit any future competition for maintenance, as by default the government only has restricted rights to such components.
- Under DFARS 252.227-7018 (rows H-J), the government typically gets unlimited rights to software not exclusively developed at private expense, but only after five years after the project has *completed* (note that this is a different starting time than DFARS 252.227-7014).

[15]This U.S. law does not cover software, but the DoD also applies this to software per DFARS 227.7203-1(c) and (d).

[16]George O. Winborne, Jr., "Who's Killing the Goose?" American Bar Association Section of Public Contract Law Program Intellectual Property in Government Contracts—What You Didn't Learn in Kindergarten, November 11-12, 2010, Seaport Hotel, Boston, Massachusetts. https://acc.dau.mil/adl/en-US/401584/file/54029/Winborne_ABAPCL_paper_Who%27s_Killing_the_Goose_For%20_Release.pdf

Amendment I can remove this right as long as the product is "reasonably available to the public for purchase."

- FAR 52.227-17 (rows L-N) is, according to FAR 27.409(e), to be used for software for the "government's internal use" or where "there is a need to limit distribution" or to "obtain indemnities for liabilities." However, purposes change; software originally developed for the "government's internal use" may become software that should be publicly released as OSS. This document simply describes what is allowed, rather than the expectations of the original contract authors.
- DFARS 252.227-7020 (rows O-P, the special works clause) is discussed in DFARS 227.7106. That discussion does not specifically mention software, but the -7020 clause can be used for software. DFARS 227.7106(2) says it can be used for "a work" and is not just limited to "technical data." This clause should be used if the government must own or control copyright. For example, it might be appropriate if the government wishes publicly release OSS and be able to (1) directly enforce copyright in court, and/or (2) provide indemnification.

3. Do you have the necessary other intellectual rights (e.g., patents)?

You need to make sure that you have any other necessary intellectual rights. Most importantly, determine if there are any relevant patents, and if so, what the rights to them are.

Other potential issues are trademark, trade dress, government seals, and trade secrets. Trademark issues, if relevant, can often be easily addressed by simply removing the trademark marking. If the contractor has granted copyright or unlimited rights to the government, then the government already has the rights to release that information to the public and is thus not barred from public release by trade secret law.

4. Do you have permission to release to the public?

In particular, for public release the material must not be restricted by:

- *Classification*. Classified data cannot be legally released to the public. Where this is not obvious, a classification review may be required.
- *Distribution statements*. A government contracting officer may require certain clauses be included in data (including software) to limit its release; contractors must obey these clauses or cause them to be rescinded.
- *Export controls*. The Export Administration Regulations (EAR) are issued by the Department of Commerce), and the International Traffic in Arms Regulations (ITAR) are issued by the Department of State. These prohibit the unlicensed export of specific technologies for reasons of national security or protection of trade. Note that cryptography can invoke export control issues.

Export controls can be particularly confusing, and the penalties for failing to comply with them can be stiff (including large fees and jail time). Thus, here are some basics about export control:

- More information about export control regulations under the EAR are provided by the U.S. Department of Commerce Bureau of Industry and Security (BIS) (http://www.bis.doc.gov/licensing/). In particular, see their pages on "Export Control Basics" and "Licensing Guidance." Any item (including software) that is sent from the US to a foreign destination, including to any foreign national, is an export – even if the item originally came from outside the US. Certain U.S. exports/re- exports require a license from BIS. A key is knowing whether the item you are intending to export has a specific Export Control Classification Number (ECCN) as listed in the Commerce Control List (CCL), available on the EAR website. In addition, a license is required

for virtually all exports and many re-exports to embargoed destinations and countries designated as supporting terrorist activities.

- Similarly, more information about the export control regulations under the ITAR, which implements the Arms Export Control Act (AECA), are provided by the U.S. Department of State Directorate of Defense Trade Controls (DDTC) (http://www.pmddtc.state.gov). In particular, see their page on "Getting Started." The US regulates exports and re-exports of defense items and technologies, so if what you wish to export is covered by the U.S. Munitions List (USML), a license from DDTC is required. You may file a commodity jurisdiction request (CJ) to determine whether an item or service is covered by the U.S. Munitions List (USML) and therefore subject to export controls related to AECA and ITAR.

The Department of Defense (DoD) does ***not*** have authority to grant export control licenses. A contractor may be liable if he or she relies on a DoD official's permission for export control, because in most cases the DoD does not have this authority. Note that even when an export-controlled release of software is granted, it is often contingent on not releasing the source code, making such "releases" useless for open technology development among all parties.

However, ***if the DoD determines that something it has purview over is releasable to the public, it is no longer subject to export control***. This is because 15 C.F.R. 734.3(b)(3) says that "The following items are not subject to the EAR . . . Publicly available technology and software...." Similarly, 22 CFR 125.4 (13) notes that technical data is exempt from ITAR export controls if it is "approved for public release (i.e., unlimited distribution) by the cognizant U.S. government department or agency or Office of Freedom of Information and Security Review." Thus, ***if software is intended to be released to the public, having the cognizant U.S. government department or agency (such as the DoD) approve its public release is often the best way to fully comply with export control regulations***.

5. Do you have the materials (e.g., source code) and are all materials properly marked?

The government and upper-tier contractors should ensure that they receive *all* material, including source code, that they are entitled to. It is all too common to have the right to the source code or related materials, yet not have it and thus be unable to exercise your rights. Source code is necessary for potential OSS release, and it is also necessary to enable competition for future software maintenance bids. Both the government and contractors should make sure that they do not lose the source code, but instead treat it as valuable data (e.g., by creating multiple backup copies in different locations).

Under DFARS 252.227-7014, the definition of "computer software" includes not only "computer programs" but also "source code, source code listings... and related material that would enable the software to be reproduced, recreated, or recompiled." Thus, a delivery of developed software is *supposed* to include source code by default. Also, (b)(1)(i) and (b)(2)(i) state that the government has rights to software (whether it was a deliverable or not) if its funds were used.

Source code should only be accepted if is ready for use. Material should only be considered acceptable as source code if it is the preferred form of the work for making modifications to it. Source code should *not* be accepted if it is just a printout or electronic images of a printout. It should not be accepted unless it is easy to automatically rebuild, e.g., a "make" or similar simple command should be sufficient to recreate an executable. Build documentation should be included with any deliverable, including information on what is required to rebuild it.

It would be best if the source code also included the historical record (e.g., a complete record of each change, who made it, and when), in an electronic form adequate for transfer to another configuration

management system. Ideally, the government should have sufficient access to the software engineering environment (SEE) of the contractor, so that the government could monitor changes as they are made.

Ensure that the source code and other materials are marked appropriately. Companies may include restrictive markings on materials, and if those markings are inappropriate, then the markings need to be fixed. Government contract clauses include processes for fixing incorrect markings; follow them. Government and upper-tier contractors need to promptly challenge improperly marked materials due to time limits. For example, contracts using DFARS 227.7203-13 include, in item (d)(3)(i), a challenge time limit of three years after either the final payment or the delivery of software, whichever is later. Also, improper markings tend to be copied into other materials; fixing markings early greatly reduces the effort to fix them later.

Who has authority?

Unfortunately, it is not always obvious *who* in government or the various contractors can make these decisions. It would be best if the government and contractors could clarify roles, policies, and procedures. In the meantime, the following may be helpful:

- As noted above, when there are multiple contractors or suppliers, the legal arrangements between the organizations determine *which* contractors/suppliers are legally allowed to assert copyright. Lead contractors do *not* necessarily receive copyrights from their subcontractors and suppliers. By U.S. law (17 USC §201), "Copyright... vests initially in the author or authors of the work... In the case of a work made for hire, the employer or other person for whom the work was prepared is considered the author [and holds the copyright] unless the parties have expressly agreed otherwise in a written instrument signed by them."
- The 2009 DoD OSS memo *does* clarify who in the DoD can determine when it should release software as OSS, and under what conditions. It says that "Software items, including code fixes and enhancements, developed for the Government should be released to the public (such as under an open source license) when all of the following conditions are met:
    1. The project manager, program manager, or other comparable official determines that it is in the Government's interest to do so, such as through the expectation of future enhancements by others.
    2. The Government has the rights to reproduce and release the item, and to authorize others to do so. For example, the Government has public release rights when the software is developed by Government personnel, when the Government receives "unlimited rights" in software developed by a contractor at Government expense, or when pre-existing OSS is modified by or for the Government.
    3. The public release of the item is not restricted by other law or regulation, such as the Export Administration Regulations or the International Traffic in Arms Regulation, and the item qualifies for Distribution Statement A, per DoD Directive 5230.24 (reference (i))."
- Some organizations do not have a review process for software source code but *do* have a process for reviewing documents. In these cases, it may be appropriate to submit the source code to the document review process. This is especially relevant for classification review.

Final notes

If the government and relevant contractors intend to release software as OSS, it's best if that is explicitly stated ahead of time. For example, OSS could be identified as the planned software maintenance philosophy per DFARS 227.7203-2(b)(1). However, since many contracts do not discuss releasing software as OSS, it's important to understand the default rules for commonly-encountered cases.

If software is released to the public as OSS and it becomes "customarily used by the general public or by nongovernmental entities for purposes other than governmental purposes," then that software becomes *commercial software*. This is by both law (41 USC §403) and regulation (e.g., DFARS 252.227-7014(a)(1)). It does not matter if the software was originally developed with government funds, or not. Thus, releasing software as OSS can be a commercialization approach.

The U.S. government and its contractors have released many programs as OSS. We hope that this material provides a path to responsibly releasing software as OSS in a manner consistent with laws, regulations, and contracts.

Appendix C: Basics of Open Source Software (OSS)

This appendix describes the basics of open source software (OSS). In particular, it gives the common definitions of OSS, describes their basic legal underpinnings, describes common types of OSS licenses, and discusses how OSS can be legal combined and recombined.

C.1 Defining OSS

Defining terms is difficult; this section presents the DoD definition as well as the two most common industry definitions (the "Free Software Definition" and "Open Source Definition").

In the DoD, Open Source Software (OSS) is defined as "software for which the human-readable source code is available for use, study, reuse, modification, enhancement, and redistribution by the users of that software" [DoD2009].

The Free Software Foundation (FSF) defines the term "Free Software" in its "Free Software Definition". While the FSF intentionally uses a different term, for purposes of this paper this definition is a useful definition of OSS. A program is free software if users have the freedom "to run, copy, distribute, study, change and improve the software. More precisely, it means that the program's users have the four essential freedoms:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this."

The non-profit Open Source Initiative (OSI)[17] is set up to review potential licenses and judge if they are an OSS license. The ten tenants of its "Open Source Definition" are:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

[17] http://www.opensource.org

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

C.2 Intellectual Rights Law

In the U.S. and most other countries there is a body of law that governs intellectual works including software. This body of law is often called “intellectual rights” law or “intellectual property” law. Since the term “intellectual property” is often misleading, this paper will use the term “intellectual rights” instead. This body of law includes copyright, patent, and trademark law. The following sections discuss the basics of these laws in the U.S.

C.2.1 Copyright & License

Currently, copyright law comes into play the moment an original work is created and put in a "fixed form" (that is, written or typed). Since OSS licenses are governed by copyright law, it is important to understand copyright law. background on copyright is required.

A copyright provides copyright holders exclusive rights to do certain things with the intellectual work that other cannot do without your permission. The full list of rights can be found in the U.S. Copyright Act, 17 USC §106; for software these include the exclusive right to:

- make copies
- prepare derivative works
- distribute copies of the original or derived works

The copyright holder can also grant a license to someone else to copy, modify, or distribute a piece of software, possibly with certain restrictions or conditions.

C.2.2 Patents

Patents dictate and enforce how an original invention (in practice, an idea) is controlled. Patents give an owner the right to exclude others from doing certain things with patented intellectual property [Rosen2005, page 23]. The full list of rights can be found in the U.S. Patent Act, 35 USC §154, but the include the right of the patent-holder to exclude others from:

- making products embodying their patented invention.
- using products embodying their invention.
- selling or offering for sale products embodying their invention.
- importing products embodying their patented invention.

Applying patents to software is highly controversial; many countries explicitly forbid it. Software could not be patented in the U.S. for many years, and during these years a wide number of innovations were produced, so there is excellent evidence that patents are not necessary for software innovation.
However, at the time of this writing, patents on software are permitted in the U.S., and there are a vast number of software patents.
Since software sometimes has patents associated with its use, some of the OSS licenses provide a license to users of that software to use and extend the software without conditions except those embodied in the license.

C.2.3 Trademark

Trademark provides exclusivity over the use of a name (e.g., a brand). Many successful open source software projects claim trademark over the name of an OSS project. Trademarks need not be registered, but certain trademark-related actions cannot be performed unless the trademark is registered.

Registering a trademark can be a relatively easy process or expensive process depending on how much protection a company desires. Details on how to register a trademark can be found on the U.S. Patent and Trademark Office's website: http://www.uspto.gov.

C.3 OSS License Types and Combinations

OSS is often combined and recombined with other OSS to produce new and useful combinations. Combining software requires that developers and users obey *all* of the licenses simultaneously. Great care should be taken when picking a license to ensure that a software program can be used and re-used by the widest group of users as possible (if that is the goal of making a software program OSS).

Only some OSS licenses can be combined with other types of licenses while meeting the requirements of all the licenses. The following figure and text is derived from [Wheeler2007], which summarizes how some OSS licenses can be combined:

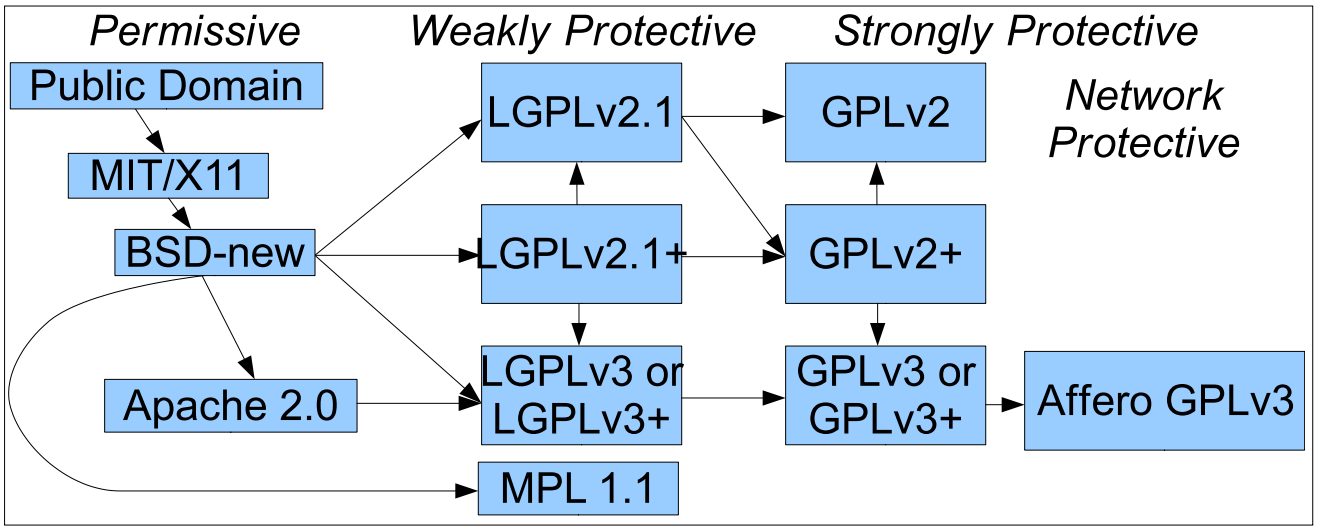

Figure C.1. OSS Licenses Interaction [Wheeler2007]

In figure C.1, the shaded boxes are the names of different FLOSS licenses (the "+" means "or any later version"). An arrow from box A to box B means that you can combine software with these licenses; the combined result effectively has the license of B, possibly with additions from A. To see if software can be combined, just start at their respective licenses, and find a common box you can reach following the arrows (aka "following the slide"). F or example, Apache 2.0-licensed software and GPLv2+-licensed software can both reach "GPLv3 or GPLv3+", so they can be combined using GPLv3 or GPLv3+. This figure has been carefully crafted so following a path determines if two licenses are compatible. For more information you must examine the license text, but this gives the basic answer quickly.

The figure shows the OSS licenses, organized into three groups:

1. At the left are the "permissive" licenses, which permit the software to become proprietary (i.e., not FLOSS). At the top left is "Public Domain" (meaning in this case "no copyright protection"), which strictly speaking isn't a license but in effect it works like one. You can do anything with public domain software, but it is rare; the software must be explicitly released to the public domain or be created by a U.S. Government employee in their official capacity. Next is the so-called "MIT" or "X11" license, which is very permissive (you can do just about anything except sue the author). Software under the MIT license is easily combined with the modern 3-clause Berkeley Software Distribution (BSD-new) license, which compared to the MIT license adds a clause forbidding the use of the author's name to endorse or promote products without permission (it's debatable if this clause actually does anything, since you typically have to have such permission anyway). Finally we have the Apache version 2.0 license.

2. At the right are the "strongly protective" ("strong copyleft") licenses, which prevent the software from becoming proprietary. This includes the most popular FLOSS license, the GNU General Public License (GPL). The GPL has a version 2 (GPLv2) and 3 (GPLv3); a "+" afterwards means "version X or later". GPLv2-only cannot be combined with the network-protective Affero GPLv3, but GPLv2+ ("version 2 or later") can via GPLv3. The most popular OSS license, by far is the GPL; most OSS software is released under the GPL [Wheeler2010g].
3. In the middle are the "weakly protective" ("weak copyleft") licenses, a compromise between permissive and strongly protective licenses. These prevent the software component (often a software library) from becoming proprietary, yet permit it to be part of a larger proprietary program. This figure shows the rules when you are making other software part of the weakly protected component; there are other possibilities if you are only using the component as a library. The GNU Lesser General Public License (LGPL) is the most popular weakly protective license, and has a version 2.1 (LGPLv2.1) and 3 (LGPLv3). Note that LGPLv2.1 gives you permission to relicense the code under any version of the GPL since GPLv2. Another such license is the Mozilla Public License 1.1 (MPL 1.1), but the MPL has the serious drawback of being incompatible with the widely-popular GPL; you can't even use an MPL module in a larger GPL'ed program."

Appendix D: How to Pick an OSS License

D.1 Key License Criteria

When picking an OSS license, keep the following criteria in mind:

1. *Use an existing OSS license; **do not** create a new OSS license*. Users hate dealing with many different OSS licenses, because of the problems they create. Every new license must be evaluated by every potential user or their legal department, raising costs substantially. Fragments of one program often cannot be used in another program due to license incompatibility, and even where they can be combined, the legal analysis to determine this with new licenses can be substantial. Creating an OSS license is also very risky; it requires specialized OSS legal knowledge that contract lawyers and contracting specialists typically do *not* have, and even experts have made mistakes that were difficult to fix later.
2. *Make sure it is actually OSS*. Choose an OSS license that has been certified as OSS by the Open Source Initiative (OSI) *and* as Free Software by the Free Software Foundation (FSF). It should be approved by the Debian and Fedora Linux distributions as well (both do legal analysis of licenses). Non-standard licenses are often not OSS, even when they are intended to be.
3. *Use a GPL-compatible license*. Most OSS is released using the GNU General Public License (GPL) version 2 or version 3. This does not mean that all OSS must be released using the GPL, but choosing a license *incompatible* with the GPL (both versions) is very unwise. For example, do *not* use licenses known to be incompatible with the GPL, such as the "NASA Open Source Agreement" version 1.3 or the "Mozilla Public License" (MPL). If you must use a GPL-incompatible license, simultaneously *also* release the software using a standard GPL-compatible license.
4. *Do not be confused by myths about the GPL*. The GPL only requires that you provide source code if you provided executable code to them; it does **not** require public release of anything. If it stays within the US government, it has not been distributed at all (there is only one US federal government). When GPL-licensed code is loaded onto a classified government network, that does *not* require the release of any data or software. Source code only has to be released if changes are made, and even then, the source code *only* needs to be released to those who received an executable program. Thus, there is no problem with taking a GPL'ed program and making classified modifications; by definition, the only recipients will have a clearance for classified information. Some vendors may claim that "if we use GPL'd code, we'd have to release this strategic military software to the Internet", but this claim is simply not true.
5. *Choose a license that meets your expected uses*. If the software is likely to be combined with another program, use at least one license that is compatible with the other program's. If the software will be export-controlled, ensure that the licensing regime can work within it. If a GPL'ed program is exported to another country, that recipient must be able to receive the source code to it (so this *is* an area where the GPL must be carefully considered).
6. *Use a common OSS license*. A nonstandard or uncommon license creates many legal problems. Lawyers around the world will have to carefully examine the nonstandard license and determine if it is acceptable and if it is compatible with other licenses (this can potentially cost millions of dollars, and such reviews can stymie use). Common OSS licenses (all of which are GPL-

compatible) include the MIT/X license, the new BSD license, the Apache 2.0 license, the Lesser GPL (LGPL), and the GPL.

D.2 Simple License Selection Process

Given the above, the following is a simple process for selecting an OSS software license. Simply answer the following questions until you find a match:

1. Was the software developed exclusively by US government personnel as part of their official duties?

If so, this software (if released) cannot have any copyright protection at all in the United States (US), per 17 USC 105 and 17 USC 101. Works without copyright protection are often referred to as being in the "public domain"; unfortunately, the phrase "public domain" also has other meanings, so that phrase can be confusing.

Simply releasing the software to the world without copyright protection is the simplest approach for releasing such software. However, such software *can* have copyright protection outside the US. If you (the US government) wish to enforce copyright outside the US, you should register the software's copyright in foreign countries, prepare to enforce the copyright license in the courts of foreign countries, and select a licensing alternative for when it is outside the US (e.g., from the options below). The government can also hire a contractor to modify the software, and only permit release of the combined work (without identifying which parts are which); since the modifications are copyrighted, copyright would apply to the work as a whole, and this modified work would not be a work without copyright protection.

2. Is this a modification or extension of existing OSS software?

If so, you should release the software either (1) *under the current license* of the existing OSS project, or (2) *without copyright protection* at all. That way, the new software can be incorporated into the existing project that is maintaining it.

If the project does not use a common OSS license, and the modification or extension has copyright protection, consider releasing the new software under a common OSS license as well (as described below). Releasing the software under both the unusual license and a common license (a "dual license") means that the project can use the modification or extension immediately (without changing licenses) and can more easily switch to a common OSS license in the future.

3. *Do you want anyone to be able to use the software for any purpose, including creating divergent incompatible proprietary versions and proprietary modules inside larger proprietary programs?*

If so, choose a common *permissive* (aka academic) license: specifically, the *MIT/X* license, the *new BSD* license, or the *Apache 2.0* license. Permissive licenses are especially useful when widespread use of a new technology or new standard is the goal.

These licenses are sometimes called "academic" licenses because of their value in getting academic concepts widely implemented in industry. For example, the Internet became widespread in large part because one of the first implementations of the Internet TCP/IP standards was released under this type of license. A risk of these licenses is that the software may be modified into one or more proprietary versions, which can no longer be shared and co-developed by those who developed the original version; economic forces can cause such projects

to fork into multiple projects over time. The MIT/X license is by far the simplest such license, so where that seems appropriate, use it. The new BSD (3-clause) license is also uncomplicated and widespread, so it is also a fine choice (do not use the 4-clause "old BSD" license that has been abandoned by Berkeley and others; the old version of the license is GPL-incompatible). The Apache 2.0 license is compatible with GPL version 3, but not with GPL version 2, which is a strike against it.

4. Do you want to encourage long-term maintenance of a common program by establishing a consortium-like legal framework, protecting the program from becoming a proprietary program or proprietary module?

If so, choose a common *strongly protective* (aka strong copyleft or reciprocal) license. In practice this would normally be a version of the *GNU General Public License (GPL)*, but if you wish to ensure co-development of web applications the *Affero GPL (AGPL)* might be chosen instead.

Strongly protective licenses are especially useful when trying to increase the likelihood that a given program will be maintained into the future by a consortium of interested parties. The Linux kernel is an example of such a program. There are two commonly-used versions of the GPL, version 2 and version 3. Most OSS software is released under "GPL version 2 or later", which maximizes compatibility; this is probably the *simplest* and most forward-looking route for strongly-protective licensing. If the text "any later version" is deemed unacceptable for a particular release, state "GPL version 2 or version 3", and be sure to designate who will have the authority to release the software under later versions of the license, and be sure to include that statement in the license so that others who contribute will be simultaneously bound to that decision. In some cases only one version of the GPL may be selected (e.g., version 2 or version 3), but this can inhibit combining software. Note that the Apache 2.0 license and the AGPL are compatible with GPL version 3, but not with GPL version 2.

5. Do you want to encourage long-term maintenance of a common library by establishing a consortium-like legal framework, protecting the program from becoming a proprietary library, but permitting proprietary programs to include the library?

If so, choose a common *weakly protective* (aka weak copyleft) license. This is typically implemented by using a version of the *GNU Lessor General Public License (LGPL)*. A common alternative is to license the library using the *GPL with a GPL linking exception* (the CLASSPATH linking exception is an especially common one).

Weakly protective licenses are compromises between the permissive and strongly protective licenses; they encourage co-development of the library, while allowing proprietary programs to include them. The LGPL requires that users be able to re-link to updated libraries; this is useful for users but in some programming languages this is not feasible (in which case the GPL with a GPL linking exception is the better choice).

Some organizations choose a "dual-licensing" approach in which the software is released under both an OSS license (typically the GPL) and a *non*-OSS license. The organization that holds rights to the software then requires "contributor agreements" from all external submitters; these agreements give the organization additional rights beyond the OSS license (enabling the non-OSS license to perpetuate). These agreements are normally non-exclusive, but nevertheless, as a collection these agreements give the organization additional asymmetrical rights not held by others. Since this relationship is

asymmetrical, a key issue to everyone else is who this organization is: is it a non-profit (e.g., a consortium or a government), or is it a for-profit company? Companies sometimes use this arrangement so that they can charge others for additional rights; this can be beneficial, but it can also enable a form of lock-in depending on what the company does with the additional rights and what customers need. This approach can also greatly weaken collaboration; many people cannot or will not contribute improvements under these asymmetrical arrangements. In short, these approaches weaken collaboration in exchange for some other perceived benefit; whether or not this is beneficial, and to whom, depends on the circumstances.

All of the text above presumes that *software* is being licensed. Community-developed works other than software are typically released under two licenses by the Creative Commons, which should be used for typical intellectual works *other* than software when you want to enable community maintenance:

1. Attribution (CC BY) license, which is a permissive license for works other than software
2. Attribution Share Alike (CC BY-SA) license, which is strongly protective (reciprocal) license for works other than software

References

[Apache2010] Apache Software Foundation. 2010. "Consensus Gauging through Voting". http://www.apache.org/foundation/voting.html

[Bacon2009] Bacon, Jono. August 2009. The Art of Community: Building the New Age of Participation. ISBN: 978-0-596-15671-8. http://www.artofcommunityonline.org/downloads/jonobacon-theartofcommunity-1ed.pdf

[Boyd1976] Boyd, Col. John, *Destruction and Creation - John Boyd - Winning and Losing*, Sept 3, 1976

[Burnette2006] Burnette, Ed. June 14, 2006. "HOWTO: Pick an open source license". ZDNet.com Blog. http://blogs.zdnet.com/Burnette/?p=130

[Carter 2010] Carter, Ashton B. June 2010. "Memorandum to Acquisition Professionals Subject: Better Buying Power: Mandate for Restoring Affordability and Productivity in Defense Spending" https://dap.dau.mil/policy/Documents/Policy/Carter%20Memo%20on%20Defense%20Spending%2028%20Jun%202010.pdf

[Collins2007] Collins-Sussman, Ben, and Brian W. Fitzpatrick. January 25, 2007. "How Open Source Projects Survive Poisonous People (And You Can Too)" also titled "How to Protect Your Open Source Project from Poisonous People". Google Tech Talks. Video. http://video.google.com/videoplay?docid=-4216011961522818645

[DeHaan2009] DeHaan, Mike. May 17, 2009. "Recognizing and Avoiding Common Open Source Community Pitfalls". http://michaeldehaan.net/2009/05/17/oss-pitfalls/ Retrieved 2010-08-19.

[DoDI5000] Department of Defense (DoD). December 8, 2008. *Operation of the Defense Acquisition System*. DoD Instruction 5000.02. http://www.dtic.mil/whs/directives/corres/pdf/500002p.pdf

[DoD2009] Department of Defense (DoD). October 16, 2009. *Clarifying Guidance Regarding Open Source Software (OSS)*. http://cio-nii.defense.gov/sites/oss/2009OSS.pdf

[Fogel2009] Fogel, Karl. 2009. *Producing Open Source Software: How to Run a Successful Free Software Project*. http://producingoss.com/

[GAO 2006] Government Accountability Office (GAO). July 2006. "WEAPONS ACQUISITION: DOD Should Strengthen Policies for Assessing Technical Data Needs to Support Weapon Systems". Report GAO-06-839. http://www.gao.gov/new.items/d06839.pdf

[GAO 2010] Government Accountability Office (GAO). July 2010. FEDERAL CONTRACTING: Opportunities Exist to Increase Competition and Assess Reasons When Only One Offer Is Received Report GAO-10-833. http://www.gao.gov/new.items/d10833.pdf

[Java.net] Choose a License. Undated. http://java.net/choose-license

[Lynn2010] Lynn, William J. III. September 2010. "Defending a New Domain: The Pentagon's Cyberstrategy". Foreign Affairs.

[Martin2000] Martin, Robert C. 2000. *Design Principles and Design Patterns*. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[MITRE2003] MITRE Corporation. January 2, 2003. Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense. http://cio-nii.defense.gov/sites/oss/2003Survey/dodfoss_pdf.pdf

[Morville2005] Ambient Findability: What We Find Changes Who We Become, Morville, Peter, O'Reilly Media, Sept. 2005

[National Academies 2008] Rising Above the Gathering Storm: Energizing and Employing America for a Brighter Economic Future. Report of the Committee on Science, Engineering, and Public Policy. National Academies Press, 2008

[OSI-Licenses] Open Source Initiative (OSI). Undated. Licenses by Name. http://www.opensource.org/licenses/alphabetical. Retrieved 2010-08-19.

[OSI OSD] Open Source Initiative (OSI). Open Source Definition (OSD). http://www.opensource.org/docs/osd

[OTD2006] J.C. Herz, Mark Lucas, and John Scott. April 2006. *Open Technology Development Roadmap Plan*. http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf.

[Raymond1999] Raymond, Eric. October 1999. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media. http://www.catb.org/~esr/writings/cathedral-bazaar/

[RedHat2009] Red Hat Community Architecture Team. 2009. *The Open Source Way*. http://www.theopensourceway.org

[Rosen2005] Rosen, Lawrence. 2005. *Open Source Licensing*. Prentice-Hall.

[Scott2010] Scott, John. 2010. Pentagon is Loosing the Softwar(e). Denfense News June 21, 2010. http://www.defensenews.com/story.php?i=4677662

[Van Buren2011] Van Buren, David M. January 14, 2011. SAF/AQ. "Present a Competitive Acquisition Strategy at Each Program Milestone" https://dap.dau.mil/policy/Documents/2011/Present%20a%20Competitive%20Acquisition%20Strategy%20at%20Each%20Program%20Milestone_AQ%20Memo.pdf

[VonHippel2005] Von Hippel, Eric. 2005. Democratizing Innovation. Cambridge, Massachusetts: The MIT Press. ISBN 0-262-00274-4. http://web.mit.edu/evhippel/www/democ1.htm

[Wegner 2002] Wegner, Etienne, Richard McDermott, and William M. Snyder. March 15, 2002. *Cultivating Communities of Practice*. Harvard Business Press. ISBN-10: 1578513308. ISBN-13: 978-1578513307.

[Wheeler2007] Wheeler, David A. September 27, 2007. *The Free-Libre / Open Source Software (FLOSS) License Slide*. http://www.dwheeler.com/essays/floss-license-slide.html

[Wheeler2009] Wheeler, David A. 2009. *Releasing Free/Libre/Open Source Software (FLOSS) for Source Installation*. http://www.dwheeler.com/essays/releasing-floss-software.html.

[Wheeler2010e] Wheeler, David A. Revised as of January 8, 2010. *How to Evaluate Open Source Software / Free Software (OSS/FS) Programs*. http://www.dwheeler.com/oss_fs_eval.html

[Wheeler2010g] Wheeler, David A. Released 2002-05-06, revised 2010-06-05. Make Your Open Source Software GPL-Compatible. Or Else. http://www.dwheeler.com/essays/gpl-compatible.html

Glossary

AoA	Analysis of Alternatives
DoD	Department of Defense
COTS	Commercial Off-the-shelf (OTS)
GOTS	Government Off-the-shelf (OTS)
OGOTS	Open Government Off-the-shelf (GOTS)
OSS	Open Source Software
OTD	Open Technology Development
OTS	Off-the-shelf
RFI	Request for Information
RFP	Request for Proposal