Exhibit 3

CHIEF INFORMATION OFFICER
U.S. DEPARTMENT OF DEFENSE

Search Chief Information Officer

HOME ABOUT DOD CIO TOP PRIORITIES IN THE NEWS OPPORTUNITIES LIBRARY CONTACT US

Open Source Software FAQ

DoD Open Source Software (OSS) FAQ

Frequently Asked Questions regarding Open Source Software (OSS) and the Department of Defense (DoD)

This page is an educational resource for government employees and government contractors to understand the policies and legal issues relating to the use of open source software (OSS) in the Department of Defense (DoD). The information on this page does not constitute legal advice and any legal questions relating to specific situations should be referred to legal counsel. References to specific products or organizations are for information only, and do not constitute an endorsement of the product/company.

A collaborative version of this document is published in Intellipedia-U at **https://www.intelink.gov/wiki/Open_Source_Software_(OSS)_FAQ**

Contents

Defining Open Source Software (OSS)

Q: What is open source software (OSS)?

The 16 October 2009 memorandum from the DoD CIO, "**Clarifying Guidance Regarding Open Source Software (OSS)**" defines OSS as "software for which the human-readable source code is available for use, study, re-use, modification, enhancement, and re-distribution by the users of that software".

Careful legal review is required to determine if a given license is really an open source software license. The following organizations examine licenses; licenses should pass at least the first two industry review processes, and preferably all of them, else they have a greatly heightened risk of not being an open source software license:

- Open source software licenses are reviewed and approved as conforming to the **Open Source Definition** by the **Open Source Initiative (OSI)**. The OSI publishes **a list of licenses which have successfully gone through the approval process and comply with the Open Source Definition**.
- In practice, an open source software license must also meet the **GNU Free Software Definition**; the GNU project **publishes a list of licenses that meet the Free Software Definition**.
- Fedora reviews licenses and publishes a list of **"good" licenses that Fedora has determined are open source software licenses**.
- **Debian-legal** also examines licenses (for Debian) to determine if they meet the **Debian social contract**; the **Debian license information** lists licenses that are known to pass (or not pass) these criteria.

In practice, nearly all open source software is released under one of a very few licenses that are known to meet this definition. These licenses include the **MIT license**, **revised BSD license** (and its 2-clause variant), the **Apache 2.0 license**, the **GNU Lesser General Public License (LGPL)** versions 2.1 or 3, and the **GNU General Public License (GPL)** versions 2 or 3. Using a standard license simplifies collaboration and eliminates many legal analysis costs.

Q: What are synonyms for open source software?

"Open source software" is also called "Free software", "libre software", "Free/open source software (FOSS or F/OSS)", and "Free/Libre/Open Source Software (FLOSS)". The term "Free software" predates the term "open source software", but the term "Free software" has been sometimes misinterpreted as meaning "no cost", which is *not* the intended meaning in this context. ("Free" in "Free software" refers to freedom, not price.) The term "open source software" is sometimes hyphenated as "open-source software".

The DoD has chosen to use the term "open source software" (OSS) in its official policy documents.

Q: What are antonyms for open source software?

Commercially-available software that is *not* open source software is typically called *proprietary* or *closed source* software.

Q: Is this related to "open source intelligence"?

No. In the Intelligence Community(IC), the term "open source" typically refers to overt, publicly available sources (as opposed to covert or classified sources). Thus, Open Source Intelligence (OSINT) is form of intelligence collection management that involves finding, selecting, and acquiring information from publicly available sources and analyzing it to produce actionable intelligence.

In software, "Open Source" refers to software where the human-readable source code is available to the users of the software. (see above)

Q: Is there a name for software whose source code is publicly available, but does not meet the definition of open source software?

At this time there is no widely-accepted term for software whose source code is available for review but does not meet the definition of open source software (due to restrictions on use, modification, or redistribution). Such software could be described as "source available software" or "open-box software" (such terms might *include* open source software, but could also include other software). Obviously, software that does not meet the definition of open source software is not open source software.

OSS and DoD Policy

Q: What policies address the use of open-source software in the Department of Defense?

The following policies apply:

1. The DoD CIO issued a memorandum titled "**Clarifying Guidance Regarding Open Source Software (OSS)**" on 16 October 2009, which superseded a May 2003 memo from John Stenbit.
2. The Department of Navy CIO issued a memorandum with guidance on open source software on 5 Jun 2007. This memorandum only applies to Navy and Marine Corps commands, but may be a useful reference for others. This memo is available at **http://www.doncio.navy.mil/PolicyView.aspx?ID=312** .
3. The Open Technology Development Roadmap was released by the office of the Deputy Under Secretary of Defense for Advanced Systems and Concepts, on 7 Jun 2006. It is available at **http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf**

It is available at **http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf** .

4. The Office of Management and Budget issued a memorandum providing guidance on software acquisition which specifically addressed open source software on 1 Jul 2004. It may be found at **http://www.whitehouse.gov/omb/memoranda/fy04/m04-16.html** .
5. US Army Regulation 25-2, paragraph 4-6.h, provides guidance on software security controls that specifically addresses open source software. This regulation only applies to the US Army, but may be a useful reference for others. The regulation is available at **http://www.army.mil/usapa/epubs/pdf/r25_2.pdf** .

In nearly all cases, OSS is commercial software, so the policies regarding commercial software continue to apply to OSS.

Q: Isn’t using open source software forbidden by DoD Information Assurance Policy?

No. This misconception comes from a misinterpretation of DoD Instruction 8500.2, “Information Assurance (IA) Implementation”, Enclosure 4, control DCPD-1.

The control in question reads:

DCPD-1 Public Domain Software Controls 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.

This control is intended to limit the use of certain kinds of “binary or machine executable” software when “the Government does not have access to the original source code”. As clarified in the 2009 DoD CIO Memorandum, this control does not prohibit the use of open source software, since with open source software the government *does* have access to the original source code.

In the **Desktop Application STIG version 3, release 1 (09 March 2007)**; in its section 2.4, it clearly states that DCPD-1 does not apply to open source software, for this very reason. The STIG first notes that "DoD has clarified policy on the use of open source software to take advantage of the capabilities available in the Open Source community as long as certain prerequisites are met. DoD no longer requires that operating system software be obtained through a valid vendor channel and have a formal support path, if the source code for the operating system is publicly available for review". It notes in particular that three cases for software are acceptable:

1. A utility that has publicly available source code is acceptable.
2. A commercial product that incorporates open source software is acceptable because the commercial vendor provides a warranty.
3. Vendor supported open source software is acceptable.

The DISA STIG also notes "4. A utility that comes compiled and has no warranty is not acceptable." Thus, a program must come with either source code or a warranty; if it has neither, then special dispensation is required, since it difficult to review, repair, or extend the program either directly or via someone else.

General information about OSS

Q: Is open source software commercial software? Is it COTS?

Open source software that has at least one non-governmental use, and has been or is available to the public, is commercial software. If it is already available to the public and is used unchanged, it is usually COTS.

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.

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)**. **DFARS 252.227-7014 Rights in Noncommercial Computer Software and Noncommercial Computer Software Documentation** 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."

There are many other reasons to believe OSS is commercial software:

- OSS is increasingly commercially developed and supported.
- OSS projects typically seek financial gain in the form of improvements. **U.S. Code Title 17, section 101** (part of copyright law) explicitly defines the term “financial gain” as including “receipt, or expectation of receipt, of anything of value, including the receipt of other copyrighted works.”
- OSS licenses and projects clearly approve of commercial support

Q: Why is it important to understand that open source software is commercial software?

It is important to understand that open source software is commercial software, because there are many laws, regulations, policies, and so on regarding commercial software. Failing to understand that open source software is commercial software would result in failing to follow the laws, regulations, policies, and so on regarding commercial software.

In particular, U.S. law (**10 USC 2377**) requires a preference for commercial items for procurement of supplies or services. 10 USC 2377 requires that the head of an agency shall ensure that procurement officials in that agency, to the maximum extent practicable:

1. "acquire commercial items or nondevelopmental items other than commercial items to meet the needs of the agency;
2. require prime contractors and subcontractors at all levels under the agency contracts to incorporate commercial items or nondevelopmental items other than commercial items as components of items supplied to the agency;
3. modify requirements in appropriate cases to ensure that the requirements can be met by commercial items or, to the extent that commercial items suitable to meet the agency’s needs are not available, nondevelopmental items other than commercial items;
4. state specifications in terms that enable and encourage bidders and offerors to supply commercial items or, to the extent that commercial items suitable to meet the

agency's needs are not available, nondevelopmental items other than commercial items in response to the agency solicitations;
5. revise the agency's procurement policies, practices, and procedures not required by law to reduce any impediments in those policies, practices, and procedures to the acquisition of commercial items; and
6. require training of appropriate personnel in the acquisition of commercial items."

Similarly, it requires preliminary market research to determine "whether there are commercial items or, to the extent that commercial items suitable to meet the agency's needs are not available, nondevelopmental items other than commercial items available" that "(A) meet the agency's requirements; (B) could be modified to meet the agency's requirements; or (C) could meet the agency's requirements if those requirements were modified to a reasonable extent." This market research should occur "before developing new specifications for a procurement by that agency; and before soliciting bids or proposals for a contract in excess of the simplified acquisition threshold."

An agency that failed to consider open source software, and instead only considered proprietary software, would fail to comply with these laws, because it would unjustifiably exclude a significant part of the commercial market. This is particularly the case where future modifications by the U.S. government may be necessary, since OSS by definition permits modification.

Q: Are "non-commercial software", "freeware", or "shareware" the same thing as open source software?

No.

Do not mistakenly use the term "non-commercial software" as a synonym for "open source software". As noted above, in nearly all cases, open source software is considered "commercial software" by U.S. law, the FAR, and the DFARS. **DFARS 252.227-7014** specifically defines "commercial computer software" in a way that includes nearly all OSS, and defines "noncommercial computer software" as software that does *not* qualify as "commercial computer software". In addition, important open source software is typically supported by one or more commercial firms.

Also, do not use the terms "freeware" or "shareware" as a synonym for "open source software". DoD Instruction 8500.2, "Information Assurance (IA) Implementation", Enclosure 4, control DCPD-1, states that these terms apply to software where "the Government does not have access to the original source code". The government *does* have access to the original source code of open source software, so these terms do not apply.

Q: How is OSS typically developed?

OSS is typically developed through a collaborative process.

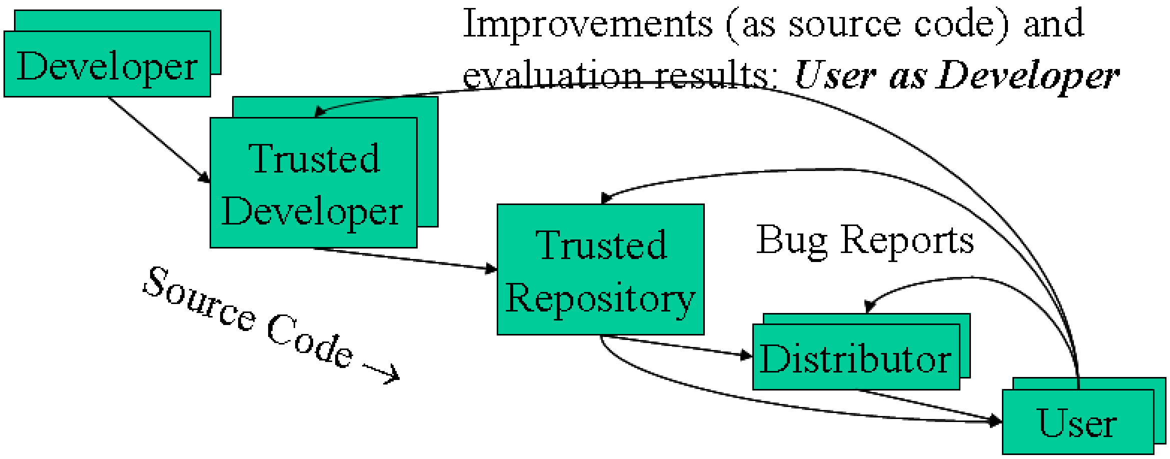
Most OSS projects have a "trusted repository", that is, some (web) location where people can get the "official" version of the program, as well as related information (documentation, bug report system, mailing lists, etc.). Users can get their software directly from the trusted repository, or get it through distributors who acquire it (and provide additional value such as integration with other components, testing, special configuration, support, and so on).

Only some developers are allowed to modify the trusted repository directly: the trusted developers. At project start, the project creators (who create the initial trusted repository) are the trusted developers, and they determine who else may become a trusted developer of this initial trusted repository. All other developers can make changes to their local copies, and even post their versions to the Internet (a process made especially easy by distributed software configuration management tools), but they must submit their changes to a trusted developer to get their changes into the trusted repository.

Users can send bug reports to the distributor or trusted repository, just as they could for a proprietary program. But what is radically different is that a user can actually make a change to the program itself (either directly, or by hiring someone to do it). Since users will want to use the improvements made by others, they have a strong financial incentive to submit their improvements to the trusted repository. That way, their improvements will be merged with the improvements of others, enabling them to use all improvements instead of only their own.

This can create an avalanche-like "virtuous cycle". As the program becomes more capable, more users are attracted to using it. A very small percentage of such users determine that they can make a change valuable to them, and contribute it back (to avoid maintenance costs). As more improvements are made, more people can use the product, creating more potential users as developers - like a snowball that gains mass as it rolls downhill.

This enables cost-sharing between users, as with proprietary development models. However, this cost-sharing is done in a rather different way than in proprietary development. In particular, note that the costs borne by a particular organization are typically only those for whatever improvements or services are used (e.g., installation, configuration, help desk, etc.). In contrast, typical proprietary software costs are per-seat, not per-improvement or service. However, it must be noted that the OSS model is much more reflective of the actual costs borne by development organizations. It costs essentially nothing to send a file or burn a CD-ROM of software; once it exists, all software costs are due to maintenance and support of software. In short, OSS more accurately reflects the economics of software development; some speculate that this is one reason why OSS has become so common so quickly.



Q: Isn't OSS developed primarily by inexperienced students?

Q: Isn't OSS developed primarily by inexperienced students?

No, OSS is developed by a wide variety of software developers, and the average developer is quite experienced. A **Boston Consulting Group** study found that the average age of OSS developers was 30 years old, the majority had training in information technology and/or computer science, and on average had 11.8 years of computer programming experience.

Q: Is open source software the same as "open systems/open standards"?

No, although they work well together, and both are strategies for reducing **"vendor lock-in"**. Vendor lock-in, aka lock-in, is the situation in which customers are dependent on a single supplier for some product (i.e., a good or service), or products, and cannot move to another vendor without substantial costs and/or inconvenience. Lock-in tends to raise costs substantially, reduces long-term value (including functionality, innovation, and reliability), and can become a serious security problem (since the supplier has little incentive to provide a secure product and to quickly fix problems found later).

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 V&V tests to ensure the openness of its key interfaces" (per the DoD OSJTF). Thus, open systems require standards that are widely-supported and consensus-based; standards that meet these (and possibly some additional conditions) may be termed "open standards". Open systems and open standards counter dependency on a single supplier, though only if there is a competing marketplace of replaceable components. Indeed, according to **Walli**, "Standards exist to encourage & enable multiple implementations". Many governments, not just the U.S., view open systems as critically necessary. DoD Directive 5000.1 states that open systems “shall be employed, where feasible”, and the European Commission identifies open standards as a major policy thrust.

There are many definitions for the term "open standard". Fundamentally, a standard is a specification, so an "open standard" is a specification that is "open". Public definitions include those of the **European Interoperability Framework (EIF)**, the **Digistan definition of open standard** (based on the EIF), and **Bruce Perens’ “Open Standards: Principles and Practice”**.

In the DoD, the **DISRonline** is a useful resource for identifying recommended standards (which tend to be open standards). DISRonline is a collection of web-based applications supporting the continuing evolution of the Department of Defense (DoD) Information Technology Standards Registry (DISR). **DAU has some information about DISRonline**. The **Open Systems Joint Task Force (OSJTF) web page** also provides some useful background.

Increasingly, many DoD capabilities are accessible via web browsers using open standards such as TCP/IP, HTTP, HTML, and CSS; in such cases, it is relatively easy to use or switch to open source software implementations (since the platforms used to implement the client or server become less relevant). As noted by the OSJTF definition for open systems, be sure to test such systems with more than one web browser (e.g., Internet Explorer and Firefox), to reduce the risk of vendor lock-in.

Q: How does open source software work with open systems/open standards?

Open standards can aid open source software projects:

- Open standards make it easier for users to (later) adopt an open source software program, because users of open standards aren’t locked into a particular implementation. Instead, users who are careful to use open standards can easily switch to a different implementation, including an OSS implementation.
- Open standards also make it easier for OSS developers to create their projects, because the standard itself helps developers know what to do. Creating any interface is an effort, and having a pre-defined standard helps reduce that effort greatly.

Note that open standards aid proprietary software in exactly the same way.

OSS aids open standards, too:

- OSS implementations can help create and keep open standards open. A FLOSS implementation can be read and modified by anyone; such implementations can quickly become a working reference model (a "sample implementation" or an "executable specification") that demonstrates what the specification means (clarifying the specification) and demonstrating how to actually implement it. Perhaps more importantly, by forcing there to be an implementation that others can examine in detail, resulting in better specifications that are more likely to be used.
- OSS implementations can help rapidly increase adoption/use of the open standard. OSS programs can typically be simply downloaded and tried out, making it much easier for people to try it out and encouraging widespread use. This also pressures proprietary implementations to limit their prices, and such lower prices for proprietary software also encourages use of the standard.

With practically no exceptions, successful open standards have OSS implementations.

So, while open systems/open standards are different from open source software, they are complementary and can work well together.

OSS Licenses

Q: What is the legal basis of OSS licenses?

Software licenses, including those for open source software, are typically based on copyright law. Under U.S. copyright law, users must have permission (i.e. a license) from the copyright holder(s) before they can obtain a copy of software to run on their system(s). Authors of a creative work, or their employer, normally receive the copyright once the work is in a fixed form (e.g., written/typed). Others can obtain permission to use a copyrighted work by obtaining a license from the copyright holder. Typically, obtaining rights granted by the license can only be obtained when the requestor agrees to certain conditions. For example, users of proprietary software must typically pay for a license to use a copy or copies. Open source software licenses grant more rights than proprietary software licenses, but they are still conditional licenses that require the user to obey certain terms.

Software licenses (including OSS licenses) may also involve the laws for patent, trademark, and trade secrets, in addition to copyright.

Export control laws are often not specifically noted in OSS licenses, but nevertheless these laws also govern when and how software may be released.

Q: Are OSS licenses legally enforceable?

Yes, in general. For advice about a *specific* situation, however, consult with legal counsel.

The **U.S. Court of Appeals for the Federal Circuit's 2008 ruling on Jacobsen v. Katzer** made it clear that OSS licenses are enforceable, even if money is not exchanged. It noted that a copyright holder may dedicate a "certain work to free public use and yet enforce an 'open source' copyright license to control the future distribution and modification of that work... Open source licensing has become a widely used method of creative collaboration that serves to advance the arts and sciences in a manner and at a pace that few could have imagined just a few decades ago... Traditionally, copyright owners sold their copyrighted material in exchange for money. The lack of money changing hands in open source licensing should not be presumed to mean that there is no economic consideration, however. There are substantial benefits, including economic benefits, to the creation and distribution of copyrighted works under public licenses that range far beyond traditional license royalties... The choice to exact consideration in the form of compliance with the open source requirements of disclosure and explanation of changes, rather than as a dollar-denominated fee, is entitled to no less legal recognition. Indeed, because a calculation of damages is inherently speculative, these types of license restrictions might well be rendered meaningless absent the less legal recognition. Indeed, because a calculation of damages is inherently speculative, these types of license restrictions might well be rendered meaningless absent the ability to enforce through injunctive relief." In short, it determined that the OSS license at issue in the case (the Artistic license) was indeed an enforceable license.

"Enforcing the GNU GPL" by Eben Moglen is a brief essay that argues why the GNU General Public License (GPL), specifically, is enforceable. U.S. courts have determined that the GPL does not violate anti-trust laws. In **Wallace vs. FSF**, Judge Daniel Tinder stated that "the GPL encourages, rather than discourages, free competition and the distribution of computer operating systems..." and found no anti-trust issues with the GPL. Similarly, in **Wallace v. IBM, Red Hat, and Novell**, the U.S. Court of Appeals for the Seventh Circuit found in November 2006 that the GNU General Public License (GPL) "and open-source software have nothing to fear from the antitrust laws". **German courts have enforced the GPL**.

Q: What are the major types of open source software licenses?

OSS licenses can be grouped into three main categories: Permissive, strongly protective, and weakly protective. Here is an explanation of these categories, along with common licenses used in each category (see **The Free-Libre / Open Source Software (FLOSS) License Slide**):

- Permissive: These licenses permit the software to become proprietary (i.e., not OSS). This includes the **MIT license** and the **revised BSD license**. The **Apache 2.0 license** is also a popular license in this category; note that the Apache 2.0 license is compatible with GPL version 3, but not with GPL version 2.
- Strongly Protective (aka strong copyleft): These licenses prevent the software from becoming proprietary, and instead enforce a "share and share alike" approach. In such licenses, if you give someone a binary of the program, you are obligated to give them the source code (perhaps upon request) under the same terms. This includes the most popular FLOSS license, the **GNU General Public License (GPL)**. There are two versions of the GPL in common use today: the older version 2, and the newer version 3.
- Weakly Protective (aka strong copyleft): These licenses are 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. The **GNU Lesser General Public License (LGPL)** is the most popular such license, and there are two versions in common use: the older version 2.1 and newer version 3. An alternative approach is to use the GPL plus a **GPL linking exception term (such as the "Classpath exception")**.

Q: How can you determine if different open source software licenses are compatible?

In general, legal analysis is required to determine if multiple programs, covered by different OSS licenses, can be legally combined into a single larger work. This legal analysis must determine if it is possible to meet the conditions of all relevant licenses simultaneously. If it is possible to meet the conditions of all relevant licenses simultaneously, then those licenses are *compatible*.

Thankfully, such analyses has already been performed on the common OSS licenses, which tend to be mutually compatible. Many analyses focus on versions of the GNU General Public License (GPL), since this is the most common OSS license, but analyses for other licenses are also available. Resources for further information include:

- **GPL FAQ** (Focuses on compatibility between versions of the GPL and LGPL)
- **The Free-Libre / Open Source Software (FLOSS) License Slide**
- **Various Licenses and Comments about Them**
- **Maintaining Permissive-Licensed Files in a GPL-Licensed Project: Guidelines for Developers (Software Freedom Law Center)**
- **Fedora Licensing**

In brief, the MIT and 2-clause BSD license are dominated by the 3-clause BSD license, which are all dominated by the LGPL licenses, which are all dominated by the GPL licenses. By "dominate", that means that when software is merged which have those pairs of licenses, the dominating license essentially governs the resulting combination because the dominating license essentially includes all the key terms of the other license. This also means that these particular licenses are compatible. The Apache 2.0 license is compatible with the GPL version 3 license, but not the GPL version 2 license. The GPL version 2 and the GPL version 3 are in principle incompatible with each other, but in practice, most released OSS states that it is "GPL version 2 or later" or "GPL version 3 or later"; in these cases, version 3 is a common license and thus such software is compatible.

Note that this sometimes depends on how the program is used or modified. For example, the LGPL permits the covered software (usually a library) to be embedded in a larger work under many different licenses (including proprietary licenses), subject to certain conditions. However, if the covered software/library is *itself* modified, then additional conditions are imposed.

This need for legal analysis is one reason why creating new OSS licenses is strongly discouraged: It can be extremely difficult, costly, and time-consuming to analyze the interplay of many different licenses. It is usually far better to stick to licenses that have already gone through legal review and are widely used in the commercial world.

Q: Can OSS licenses and approaches be used for material other than software?

Yes. The **Creative Commons** is a non-profit organization that provides free tools, including a set of licenses, to "let authors, scientists, artists, and educators easily mark their creative work with the freedoms they want it to carry". A copyright holder who releases creative works under one of the Creative Common licenses that permit commercial use and modifications would be using an OSS-like approach for such works. **Wikipedia** maintains an encyclopedia using approaches similar to open source software approaches. Note that **Creative Commons does not recommend that you use one of their licenses for software**; they encourage using one of the existing OSS licenses which "were designed specifically for use with software".

Computer and electronic hardware that is designed in the same fashion as open source software (OSS) is sometimes termed **open source hardware**. The term has primarily been used to reflect the free release of information about the hardware design, such as schematics, bill of materials and PCB layout data, or its representation in a hardware description language (HDL), often with the use of open source software to drive the hardware.

Software/hardware for which the implementation, proofs of its properties, and all required tools are released under an OSS license are termed **open proofs**(see the **open proofs website for more information**).

Where it is unclear, make it clear what the "source" or "source code" means.

(See **GPL FAQ, "Can I use the GPL for something other than software?"**.)

Q: Is it more difficult to comply with OSS licenses than proprietary licenses?

No, complying with OSS licenses is much easier than proprietary licenses if you only use the software in the same way that proprietary software is normally used. By definition, OSS software permits arbitrary use of the software, and allows users to re-distribute the software to others. The terms that apply to usage and redistribution tend to be trivially easy to meet (e.g., you must not remove the license or author credits when re-distributing the software). Thus, complex license management processes to track every installation or use of the software, or who is permitted to use the software, is completely unnecessary. Support for OSS is often sold separately for OSS; in such cases, every installation or use of the software, or who is permitted to use the software, is completely unnecessary. Support for OSS is often sold separately for OSS; in such cases, you must comply with the support terms for those uses to receive support, but these are typically the same kinds of terms that apply to proprietary software (and they tend to be simpler in practice).

It is only when the OSS is modified that additional OSS terms come into play, depending on the OSS license. Since it is typically not legal to modify proprietary software at all, or it is legal only in very limited ways, it is trivial to determine when these additional terms may apply. The real challenge is one of education - some developers incorrectly believe that just because something is free to download, it can be merged or changed without restriction. This has never been true, and explaining this takes little time.

Q: Who can enforce OSS licenses?

Typically enforcement actions are based on copyright violations, and only copyright holders can raise a copyright claim in U.S. court. In the commercial world, the copyright holders are typically the individuals and organizations that originally developed the software. Under the current DoD contracting regime, the contractor usually retains the copyright for software developed with government funding, so in such cases the contractor (not the government) has the right to sue for copyright violation. In some cases, the government obtains the copyright; in those cases, the government can sue for copyright violation.

However, the government can release software as OSS when it has unlimited rights to that software. The government is not the copyright holder in such cases, but the government can still enforce its rights. Although the government cannot directly sue for copyright violation, in such cases it can still sue for breach of license and, presumably, get injunctive relief to stop the breach and money damages to recover royalties obtained by breaching the license (and perhaps other damages as well).

In addition, a third party who breaches a software license (including for OSS) granted by the government risks losing rights they would normally have due to the "doctrine of unclean hands". The **doctrine of unclean hands**, per law.com, is "a legal doctrine which is a defense to a complaint, which states that a party who is asking for a judgment cannot have the help of the court if he/she has done anything unethical in relation to the subject of the lawsuit. Thus, if a defendant can show the plaintiff had 'unclean hands,' the plaintiff's complaint will be dismissed or the plaintiff will be denied judgment." So if the government releases software as OSS, and a malicious developer performs actions in violation of that license, then the government's courts need not enforce any of that malicious developer's intellectual rights to that result. In effect, the malicious developer could lose many or all rights over their license-violating result, even rights they would normally have had! Since OSS licenses are quite generous, the only license-violating actions a developer is likely to try is to release software under a more stringent license... and those will have little effect once they cannot be enforced in court. In short, the government can enforce its licenses, even when it doesn't have the copyright.

See **GPL FAQ, "Who has the power to enforce the GPL?"**

OSS and Security/Software Assurance/System Assurance/Supply Chain Risk Management

Q: Does the DoD use OSS for security functions?

Yes. The **2003 MITRE study, "Use of Free and Open Source Software (FOSS) in the U.S. Department of Defense"**, for analysis purposes, posed the hypothetical question of what would happen if OSS software were banned in the DoD, and found that OSS "plays a far more critical role in the DoD than has been generally recognized... (especially in) Infrastructure Support, Software Development, Security, and Research". In particular, it found that DoD security "depends on (OSS) applications and strategies", and that a hypothetic ban "would have immediate, broad, and in some cases strongly negative impacts on the ability of the DoD to analyze and protect its own networks against hostile intrusion. This is in part because such a ban would prevent DoD groups from using the same analysis and network intrusion applications that hostile groups could use to stage cyberattacks. It would also remove the uniquely (OSS) ability to change infrastructure source code rapidly in response to new modes of cyberattack".

Q: Doesn't hiding source code automatically make software more secure?

No. Indeed, vulnerability databases such as CVE make it clear that merely hiding source code does not counter attacks:

- Dynamic attacks (e.g., generating input patterns to probe for vulnerabilities and then sending that data to the program to execute) don’t need source or binary. Observing the output from inputs is often sufficient for attack.
- Static attacks (e.g., analyzing the code instead of its execution) can use pattern-matches against binaries - source code is not needed for them either.
- Even if source code is necessary (e.g., for source code analyzers), adequate source code can often be regenerated by disassemblers and decompilers sufficiently to search for vulnerabilities. Such source code may not be adequate to cost-effectively *maintain* the software, but attackers need not maintain software.
- Even when the original source is necessary for in-depth analysis, making source code available to the public significantly aids defenders and not just attackers. Continuous and broad peer-review, enabled by publicly available source code, improves software reliability and security through the identification and elimination of defects that might otherwise go unrecognized by the core development team. Conversely, where source code is hidden from the public, attackers can attack the software anyway as described above. In addition, an attacker can often acquire the original source code from suppliers anyway (either because the supplier voluntarily provides it, or via attacks against the supplier); in such cases, if only the attacker has the source code, the attacker ends up with another advantage.

Hiding source code *does* inhibit the ability of third parties to respond to vulnerabilities (because changing software is more difficult without the source code), but this is obviously *not* a security advantage. In general, “Security by Obscurity” is widely denigrated.

This does *not* mean that the DoD will reject using proprietary COTS products. There are valid business reasons, unrelated to security, that may lead a commercial company selling proprietary software to choose to hide source code (e.g., to reduce the risk of copyright infringement or the revelation of trade secrets). What it does mean, however, is that the DoD will not reject consideration of a COTS product merely because it is OSS. Some OSS is very secure, while others are not; some proprietary software is very secure, while others are not. Each product must be examined on its own merits.

Q: What are indicators that a specific OSS program will have fewer unintentional vulnerabilities?

As noted in the **Secure Programming for Linux and Unix HOWTO**, three conditions reduce the risks from unintentional vulnerabilities in OSS:

1. Developers/reviewers need security knowledge. Knowledge is more important than the licensing scheme.
2. People have to actually review the code.
    1. This has a reduced likelihood if the program is niche/rarely-used, few developers, rare computer language, or not really OSS. Conversely, if it widely-used, has many developers, and so on, the likelihood of review increases. Examine if it is truly community-developed - or if there are only a very few developers.
    2. Review really does happen. Several static tool vendors support analysis of OSS (such as Coverity and Fortify) as a way to improve their tools and gain market use. There are many general OSS review projects, such as those by OpenBSD and the Debian Security Audit team. And of course, individual OSS projects often have security review processes or methods (such as Mozilla's bounty system). If there are reviewers from many different backgrounds (e.g., different countries), this can also reduce certain risks. When examining a specific OSS project, look for evidence that review (both by humans and tools) does take place.
3. Problems must be fixed. It is far better to fix vulnerabilities before deployment - are such efforts occuring? When the software is already deployed, does the project develop and deploy fixes?

Q: Is there a risk of malicious code becoming embedded into OSS?

The use of any commercially available software, be it proprietary or OSS, creates the risk of executing malicious code embedded in the software. Even if a commercial

The use of *any* commercially-available software, be it proprietary or OSS, creates the risk of executing malicious code embedded in the software. Even if a commercial program did not originally have vulnerabilities, both proprietary and OSS program binaries can be modified (e.g., with a "hex editor" or virus) so that it includes malicious code. It may be illegal to modify proprietary software, but that will normally not slow an attacker. Thankfully, there are ways to reduce the risk of executing malicious code when using commercial software (both proprietary and OSS). It is impossible to completely eliminate all risks; instead, focus on reducing risks to acceptable levels.

The use of software with a proprietary license provides absolutely no guarantee that the software is free of malicious code. Indeed, many people have released proprietary code that is malicious. What's more, proprietary software release practices make it more difficult to be confident that the software does not include malicious code. Such software does not normally undergo widespread public review, indeed, the source code is typically not provided to the public and there are often license clauses that attempt to inhibit review further (e.g., forbidding reverse engineering and/or forbidding the public disclosure of analysis results). Thus, to reduce the risk of executing malicious code, potential users should consider the reputation of the supplier and the experience of other users, prefer software with a large number of users, and ensure that they get the "real" software and not an imitator. Where it is important, examining the security posture of the supplier (e.g., their processes that reduce risk) and scanning/testing/evaluating the software may also be wise.

Similarly, OSS (as well as proprietary software) may indeed have malicious code embedded in it. However, such malicious code cannot be directly inserted by "just anyone" into a well-established OSS project. As noted above, OSS projects have a "trusted repository" that only certain developers (the "trusted developers") can directly modify. In addition, since the source code is publicly released, anyone can review it, including for the possibility of malicious code. The public release also makes it easy to have copies of versions in many places, and to compare those versions, making it easy for many people to review changes. Many perceive this openness as an advantage for OSS, since OSS better meets Saltzer & Schroeder's "Open design principle" ("the protection mechanism must not depend on attacker ignorance"). This is not merely theoretical; **in 2003 the Linux kernel development process resisted an attack**. Similarly, SourceForge/Apache (in 2001) and Debian (in 2003) countered external attacks.
As with proprietary software, to reduce the risk of executing malicious code, potential users should consider the reputation of the supplier (the OSS project) and the experience of other users, prefer software with a large number of users, and ensure that they get the "real" software and not an imitator (e.g., from the main project site or a trusted distributor). Where it is important, examining the security posture of the supplier (the OSS project) and scanning/testing/evaluating the software may also be wise.

The example of Borland's InterBase/Firebird is instructive. For at least 7 years, Borland's Interbase (a proprietary database program) had embedded in it a "back door"; the username "politically", password "correct", would immediately give the requestor complete control over the database, a fact unknown to its users. Whether or not this was intentional, it certainly had the same form as a malicious back door. When the program was released as OSS, within 5 months this vulnerability was found and fixed. This shows that proprietary software can include functionality that could be described as malicious, yet remain unfixed - and that at least in some cases OSS is reviewed and fixed.

Note that merely being developed for the government is no guarantee that there is no malicious embedded code. Such developers need not be cleared, for example. Requiring that all developers be cleared first can reduce certain risks (at substantial costs), where necessary, but even then there is no guarantee.

Note that most commercial software is not intended to be used where the impact of *any* error of any kind is *extremely* high (e.g., a large number of lives are likely to be immediately lost if even the slightest software error occurs). Software that meets very high reliability/security requirements, aka "high assurance" software, must be specially designed to meet such requirements. Most commercial software (including OSS) is not designed for such purposes.

Using OSS in DoD systems

Q: Does the DoD already use open source software?

Yes, extensively. The **2003 MITRE study, "Use of Free and Open Source Software (FOSS) in the U.S. Department of Defense"**, identified some of many OSS programs that the DoD is *already* using, and concluded that OSS "plays a more critical role in the [Department of Defense (DoD)] than has generally been recognized".

Intellipedia is implemented using MediaWiki, the open source software developed to implement Wikipedia. This Open Source Software FAQ was originally developed on Intellipedia, using a variety of web browsers including Mozilla Firefox. Thus, this FAQ was developed using open source software.

Q: Is a lot of pre-existing open source software available?

Yes. Widely-used programs include the Apache web server, Firefox web browser, Linux kernel, and many other programs. **Estimating the Total Development Cost of a Linux Distribution** estimates that the Fedora 9 Linux distribution, which contains over 5,000 software packages, represents about $10.8 billion of development effort.

Q: Is there an "approved", "recommended" or "Generally Recognized as Safe/Mature" list of Open Source Software? What programs are already in widespread use?

No, the DoD does not have an official recommendation for any particular OSS product or set of products, nor a "Generally Recognized as Safe/Mature" list. The **2003 MITRE study, "Use of Free and Open Source Software (FOSS) in the U.S. Department of Defense"** did suggest developing a "Generally Recognized As Safe" (GRAS) list, but such a list has not been developed.

Commercial software (including OSS) that has widespread use often has lower risk, since there are often good reasons for its widespread use. The MITRE study did identify some of many OSS programs that the DoD is *already* using, and may prove helpful. Examples of OSS that are in widespread use include:

- Apache - Web server
- Mozilla Firefox - Web browser
- Mozilla Thunderbird, Evolution - Email client
- OpenOffice.org - Office document suite
- OpenSSH - Secure Shell
- OpenSSL - SSL/cryptographic library implementation
- bind - DNS server
- Postfix, Sendmail - Mail servers
- gcc - Compiler suite
- GNAT - Ada compiler suite (technically this is part of gcc)
- perl, Python, PHP - Scripting languages
- Samba - Windows - Unix/Linux interoperability
- Mailman - mailing list manager
- MySQL and PostgreSQL - Relational Database System
- GIMP - Bitmap graphics editor
- MediaWiki - Wiki

There are many "Linux distributions" which provides suites of such software such as Red Hat Enterprise Linux, Fedora, Novell SuSE, Debian and Ubuntu. Other open source software implementations of Unix interfaces include Solaris, OpenBSD, NetBSD, and FreeBSD.

Again, these are examples, and not official endorsements of any particular product or supplier.

Q: What are some military-specific open source software programs?

Some more military-specific OSS programs used in the military include:

- **FalconView** - PC-based mapping application
- Open Source Software for Imagery & Mapping (**OSSIM**) - geospatial image viewing (with classified plugins)
- OSSIM Mapping ARchieve System (**OMAR**) - video indexing
- **BRL-CAD** - solid modeling (Army)
- **Optics** - MASINT toolset (with classfied plugins)
- **Delta3d** - Game/Simulation engine for modeling and simulation (e.g., for military training/exercises)

There are many others.

Q: Is there any quantitative evidence that open source software can be as good as (or better than) proprietary software?

Yes; **Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!** is a survey paper that "provides quantitative data that, in many cases, using open source software / free software (abbreviated as OSS/FS, FLOSS, or FOSS) is a reasonable or even superior approach to using their proprietary competition according to various measures.. (its) goal is to show that you should consider using OSS/FS when acquiring software". It points to various studies related to market share, reliability, performance, scalability, security, and total cost of ownership.

This is in addition to the advantages from OSS because it can be reviewed, modified, and redistributed with few restrictions (inherent in the definition of OSS).

That said, this does *not* mean that all OSS is superior to all proprietary software in all cases by all measures. Each government program must determine its needs, and then evaluate its options for meeting those needs.

Q: When a DoD contractor is developing a new system/software as a deliverable in a typical DoD contract, is it possible to include existing open source software?

Yes, it's possible. In nearly all cases pre-existing OSS are "commercial components", and thus their use is governed by the rules for including any commercial components in the deliverable. The use of commercial components is generally encouraged, and when there are commercial components, the government expects that it will normally use whatever license is offered to the public. Depending on the contract and its interpretation, contractors may be required to get governmental permission to include commercial components in their deliverable; where this applies, this would be true for OSS components as well as proprietary components. As with all commercial items, organizations must obey the terms of the commercial license, negotiate a different license if necessary, or not use the commercial item.

An alternative is to not *include* the OSS component in the deliverable, but simply *depend* on it, as long as that is acceptable to the government. This is often done when the deliverable is a software application; instead of including commercially-available components such as the operating system or database system as part of the deliverable, the deliverable could simply state that what it requires.

Q: When a DoD contractor is developing a new system/software as a deliverable in a typical DoD contract, is it possible to use existing software licensed using the GNU General Public License (GPL)? Can the DoD used GPL-licensed software?

Yes. There is no DoD policy forbidding or limiting the use of software licensed under the GNU General Public License (GPL).

The DoD already uses a wide variety of software licensed under the GPL. A **2003 MITRE study, "Use of Free and Open Source Software (FOSS) in the U.S. Department of Defense"**, identified many OSS programs that the DoD is *already* using that are licensed using the GPL. These included the Linux kernel, the gcc compilation suite (including the GNAT Ada compiler), the OpenOffice.org office suite, the emacs text editor, the Nmap network scanner, OpenSSH and OpenSSH for encryption, and Samba for Unix/Linux/Windows interoperability. This should not be surprising; the DoD uses OSS extensively, and the GPL is the most popular OSS license.

As with all commercial items, the DoD must comply with the item's license when using the item. There are two versions of the GPL: version 2 and version 3. The key issue with both versions of the GPL is that, unlike most other OSS licenses, the GPL licenses require that a recipient of a binary (executable) must be able to demand and receive the source code of that program, and the recipient must also be able to propogate the work under that license. The Free Software Foundation (FSF) interprets linking a GPL program with another program as creating a derivative work, and thus imposing this license term in such cases.

In most cases, this GPL license term is not a problem. After all, most proprietary software licenses explicitly forbid modifying (or even reverse-engineering) the program, so the GPL actually provides *additional* rights not present in most proprietary software. So if the program is being used and not modified (a very common case), this additional term has no impact. Even for many modifications (e.g., bug fixes) this causes no issues because in many cases the DoD has no interest in keeping those changes confidential.

However, if the GPL software must be mixed with other proprietary/classified software, the GPL terms must still be followed.

Q: Under what conditions can GPL-licensed software be mixed with proprietary/classified software?

Software licensed under the GPL *can* be mixed with software released under other license terms (e.g., proprietary or classified software), but only under conditions that do not violate any license. Such mixing can normally only occur when certain kinds of separation are maintained - and thus this becomes a design issue.

The **2003 MITRE study section 1.3.4** outlines several ways to legally mix GPL with proprietary/classified software:

- Distribution Mixing – GPL and other software can be stored and transmitted together. Example: GPL software can be stored on the same computer disk as (most kinds of) proprietary software.
- Execution Mixing – GPL and other software can run at the same time on the same computer or network. Example: GPL and (unrelated) proprietary applications can be running at the same time on a desktop PC.
- Application Mixing – GPL can rely on other software to provide it with services, provided either that those services are either generic (e.g., operating system services) or have been explicitly exempted by the GPL software designer as non-GPL components. Examples include GPL applications running on proprietary operating systems or wrappers, and GPL applications that use proprietary components explicitly marked as non-GPL. Windows Services for UNIX 3.0 is a good example of commercial use of GPL application mixing.

GPL application mixing.

- Service Mixing – GPL can provide generic services to other software. These services must be genuinely generic in the sense that the applications that use them must not depend on the detailed design of the GPL software to work. An example is (connecting) a GPL utility to a proprietary software component by using the Unix "pipe" mechanism, which allows one-way flow of data to move between software components. This is the tightest form of mixing possible with GPL and other types of software, but it must be used with care to ensure that the GPL software remains generic and is not tightly bound to any one proprietary software component.

Often such separation can occur by separating information into data and a program that uses it, or by defining distinct layers. As long as a GPL program does not embed GPL software into its outputs, a GPL program can process classified/proprietary information. Thus, GPL'ed compilers can compile classified programs (since the compilers treat the classified program as data), and a GPL'ed implementation of a virtual machine (VM) can execute classified software (since the VM implementation runs the software as data). Similarly, a GPL'ed "engine" program can be controlled by classified data that it reads. In addition, a GPL'ed program can run on top of a classified/proprietary platform when the platform is a separate "System Library" (as defined in GPL version 3). Note that enforcing such separation has many other advantages as well.

The U.S. government can directly combine GPL and proprietary/classified software into a single program arbitrarily, as long as the result is never conveyed outside the U.S. government, but this approach should not be taken lightly. This approach may inhibit later release of the combined result to other parties (e.g., allies). When taking this approach, contractors hired to modify the software must not retain copyright or other rights to the result (else the software would be conveyed outside the U.S. government); see **GPL version 3 section 2, paragraph 2** which states this explicitly.

It can be argued that classified software can be arbitrarily combined with GPL code, beyond the approaches described above. The argument is that the classification rules are simply laws of the land (and not "additional" rules), the classification rules already forbid the release of the resulting binaries to those without proper clearances, and that the GPL only requires that source code be released to those who received a binary. While this argument may be valid, we know of no general counsel ruling confirming this. Anyone who is considering this approach should obtain a ruling from general counsel first (and please let the FAQ authors know!).

If a legal method for using the GPL software for a particular application cannot be devised, and a different license cannot be negotiated, then the GPL-licensed component cannot be used for that particular purpose. Note that this also applies to proprietary software, which often have even stricter limits on if/how the software may be changed.

Q: Is the GPL compatible with Government Unlimited Rights contracts, or does the requirement to display the license, etc, violate Government Unlimited Rights contracts?

The GPL and government "unlimited rights" terms have similar goals, but differ in details. This isn't usually an issue because of how typical DoD contract clauses work under the DFARS.

Any software that has a non-government use and is licensed to the public is *commercial software*, by definition, including OSS programs licensed to the government using the GPL. Normally the government only expects to get the usual commercial rights to commercial software, and not "unlimited rights". So if the software displays a license in a way that can't be legally disabled (as required by the GPL), there is no problem, because this is an ordinary commercial software license term. The same would be true if you used Microsoft Windows; you aren't normally permitted to disable the rights-display functions of Microsoft Windows either.

In contrast, the government normally gets "unlimited rights" only when it pays for development of that software, in full or in part. Software developed by government funding would typically be termed "noncommercial software", and thus falls under different rules. The government *does* have the right to take software it has unlimited rights to, and link it with GPL software. After all, the government can use unlimited rights software in any way it wishes.

Once the government has unlimited rights, it can release that software to the public in any it wishes - including by using the GPL. This is not a contradiction; it's quite common for different organizations to have different rights to the same software. The program available to the public may improve over time, through contributions not paid for by the U.S. government. In that case, the U.S. government can choose to use the version to which it has unlimited rights, or it can use the publicly-available commercial version available to the government through that version's commercial license (the GPL in this case).

Q: How can I evaluate OSS options?

OSS options should be evaluated in principle the same way you would evaluate any option, considering need, cost, and so on. In some cases, the sources of information for OSS differ.

Be sure to consider total cost of ownership (TCO), not just initial download costs. Even if OSS has no cost to download, there is still a cost for OSS due to installation, support, and so on (whether done in-house or through external organizations). Be sure to consider such costs over a period of time (typically the lifetime of the system including its upgrades), and use the same period when evaluating alternatives; otherwise, one-time costs (such as costs to transition from an existing proprietary system) can lead to erroneous conclusions. Include upgrade/maintenance costs, including indirect costs (such as hardware replacement if necessary to run updated software), in the TCO.

By definition, open source software provides more rights to users than proprietary software (at least in terms of use, modification, and distribution). That said, other factors may be more important for a given circumstance.

The DoD does not have a single required process for evaluating OSS. The following externally-developed evaluation processes or tips may be of use:

- **How to Evaluate Open Source Software / Free Software (OSS/FS) Programs**
- **Navica's Open Source Maturity Model (OSMM)**
- **Capgemini's Open Source Maturity Model (OSMM)**
- **Top Tips For Selecting Open Source Software**
- **Business Readiness Rating™ (BRR)**
- **QSOS**

Q: How can I migrate to OSS?

Migrating from an existing system to an OSS approach requires addressing the same issues that any migration involves.

The IDA Open Source Migration Guidelines recommend:

- before starting have a clear understanding of the reasons to migrate;
- ensure that there is active support for the change from IT staff and users;
- make sure that there is a champion for change – the higher up in the organisation the better;
- build up expertise and relationships with the OSS movement;
- start with non critical systems;
- ensure that each step in the migration is manageable.

It also suggests that the following questions need to be addressed:

- how to ensure the interoperability of systems;

- how to ensure the interoperability of systems;
- how to support mobile users;
- how to securely identify remote users;
- how to build systems that are manageable.
- ensure that security is designed in from the start and not tacked on as an after thought.

It also recommends ensuring "that decisions made now, even if they do not relate directly to a migration, should not further tie an Administration to proprietary file formats and protocols". It also notes that OSS is a disruptive technology, in particular, that it is "a move away from a product to a service based industry".

Q: How can I get support for OSS that already exists?

You can support OSS either through a commercial organization, or you can self-support OSS; in either case, you can use community support as an aid.

Commercial support can either be through companies with specialize in OSS support (in general or for specific products), or through contractors who specialize in supporting customers and provide the OSS support as part of a larger service. Examples of the former include Red Hat, Novell, HP, Sun, IBM, DMSolutions, SourceLabs, OpenLogic, Carahsoft, and Mozilla.

Some have found that community support can be very helpful. The 1997 InfoWorld “Best Technical Support” award was won by the "Linux User Community". However, you should examine past experience and your intended uses before depending on this as a primary mechanism for support.

Q: How do GOTS, Proprietary COTS, and OSS COTS compare?

Government Off-the-Shelf (GOTS), proprietary commercial off-the-shelf (COTS), and OSS COTS are all methods to enable reuse of software across multiple projects. Thus, they are all strategies for sharing the development and maintenance costs of software, potentially reducing its cost.

GOTS is especially appropriate when the software *must not* be released to the public (e.g., it is classified) or when licenses forbid more extensive sharing (e.g., the government only has government-purpose rights to the software). If the software is not released to the public at all and it provides a direct military advantage, then the U.S. military (and its allies) may obtain a distinct military advantage (note that such software would normally be classified). Unlike proprietary COTS, GOTS has the advantage that the government has the right to change the software whenever the government chooses to do so. Unfortunately, the government must pay for *all* development and maintenance costs of GOTS; since these can be substantial, GOTS runs the risk of becoming obsolescent when the government cannot afford those costs. Also, since there are a limited number of users, there is limited opportunity to gain from user innovation - which again can lead to obsolescence. Even where there is GOTS/classified software, such software is typically only a *portion* of the entire system, with other components implemented through COTS components.

Proprietary COTS is especially appropriate when there is an existing proprietary COTS product that meets the need. Proprietary COTS tend to be lower cost than GOTS, since the cost of development and maintenance is typically shared among a larger number of users (who typically pay to receive licenses to use the product). Unfortunately, this typically trades off flexibility; the government typically does not have the right to modify the software, so it often cannot fix serious security problems, add arbitrary improvements, or make the software work on platforms of its choosing. If the supplier attains a monopoly or it is difficult to switch from the supplier, the costs may skyrocket. What is more, the supplier may choose to abandon the product; software escrow can reduce these risks somewhat, but in these cases it becomes GOTS with its attendant costs.

OSS COTS is especially appropriate when there is an existing OSS COTS product that meets the need, or one can be developed and supported by a wide range of users/co-developers. OSS COTS tends to be lower cost than GOTS, in part for the same reasons as proprietary COTS: its costs are shared among more users. It also often has lower total cost-of-ownership than proprietary COTS, since acquiring it initially is often free or low-cost, and all other support activities (training, installation, modification, etc.) can be competed. Its flexibility is as high as GOTS, since it can be arbitrarily modified. However, note that this cost discussion only applies if there are many users; if no user/co-developer community is built up, then it can be as costly as GOTS.

Q: What are the risks of failing to consider the use of OSS components or approaches?

For the DoD, the risks of failing to consider the use of OSS where appropriate are of increased cost, increased schedule, and/or reduced performance (including reduced innovation or security) to the DoD due to the failure to use the commercial software that best meets the needs (when that is the case). It also risks reduced flexibility (including against cyberattack), since OSS permits arbitrary later modification by users in ways that some other license approaches do not. In addition, ignoring OSS would not be lawful; U.S. law specifically requires consideration of commercial software (including extant OSS, regardless of exactly which license it uses), and specifically instructs departments to pass this requirements down to contractors and their suppliers.

DoD contractors who always ignore components because they are OSS, or because they have a particular OSS license they don't prefer, risk losing projects to more competitive bidders. If that competitor's use of OSS results in an advantage to the DoD (such as lower cost, faster schedule, increased performance, or other factors such as increased flexibility), contractors should expect that the DoD will choose the better bid. This does not mean that existing OSS elements should always be chosen, but they should be considered.

Q: Is there a large risk that widely-used OSS unlawfully includes proprietary software (in violation of copyright)?

No; this is a low-probability risk for widely-used OSS programs. A primary reason that this is low-probability is the publicity of the OSS source code itself (which almost invariably includes information about those who made specific changes). Any company can easily review OSS to look for proprietary code that should not be there; there are even OSS tools that can find common code. A company that found any of its proprietary software in an OSS project can in most cases quickly determine who unlawfully submitted that code and sue for infringement.

In addition, widely-used licenses and OSS projects often include additional mechanisms to counter this risk. The GPL and LGPL licenses specifically recommend that "You should also get your employer (if you work as a programmer) or school, if any, to sign a 'copyright disclaimer' for the program, if necessary.", and point to additional information. Many projects, particularly the large number of projects managed by the Free Software Foundation (FSF), ask for an employer's disclaimer from the contributor's employer in a number of circumstances. The Linux kernel project requires that a person proposing a change add a "Signed-off-by" tag, attesting that the "patch, to the best of his or her knowledge, can legally be merged into the mainline and distributed under the terms of (the license)."

In practice, OSS projects tend to be remarkably clean of such issues. For example, **Code Analysis of the Linux Wireless Team's ath5k Driver** found no license problems.

When considering any software (OSS or proprietary), look for evidence that the risk of unlawful release is low. Factors that greatly reduce this risk include:

- Widespread availability and use of the software (which increases the likelihood of detection)
- Configuration management systems that record the identity of individual contributors (which acts as a deterrent)
- Licenses or development policies that warn against the unlawful inclusion of material, or require people to specifically assert that they are acting lawfully (which reduce the risk of unintentional infringement)
- Lack of evidence of infrigement (e.g., an Internet search for project name + "copyright infringement" turns up nothing). Parties are innocent until proven guilty, so if there *is*

such a charge, investigate the charges' merits instead of presuming guilt.

Q: Is there a large risk to DoD contractors that widely-used OSS violates enforceable software patents?

Typically not, though the risk varies depending on their contract and specific circumstance. Note, however, that this risk has little to do with OSS, but is instead rooted in the risks of U.S. patent infringement for *all* software, and the patent indemnification clauses in their contract.

It is difficult for software developers (OSS or not) to be confident that they have avoided software patent infringement in the United States, for a variety of reasons. Software might not infringe on a patent when it was released, yet the same software may later infringe on a patent if the patent was granted after the software's release. Many software developers find software patents difficult to understand, making it difficult for them to determine if a given patent even applies to a given program. Patent examiners have relatively little time to review each patent, and do not have effective access to most prior art in software, which may lead them to grant patents for previously-published inventions or "obvious" inventions. The U.S. has granted a large number of software patents, making it difficult and costly to examine all of them. Recent rulings have strengthened the requirement for "non-obviousness", which probably renders unenforceable some already-granted software patents, but at this time it is difficult to determine which ones are affected. As a result, it is difficult to develop software and be confident that it does not violate enforceable patents. The DoD has not expressed a position on whether or not software should be patented, but it *is* interested in ensuring that software that effectively supports its missions can be developed in a cost-effective, timely, and legal manner.

U.S. government contractors (including those in the DoD) are often indemnified from patent infringement by the U.S. government as part of their contract. This greatly reduces contractors' risks, enabling them to get work done (given this complex environment). They can obtain this by receiving certain authorization clauses in their contracts. **FAR 52.227-1 (Authorization and Consent)**, as prescribed by **FAR 27.201-2(a)(1)**, inserts the clause that the "Government authorizes and consents to all use and manufacturer... of any invention (covered by) U.S. patent". The related FAR 52.227-2 (Notice and Assistance Regarding Patent and Copyright Infringement), as prescribed by FAR 27.201-2(b), requires the contractor to report to the Contracting Officer each notice or claim of patent/copyright infrigement in reasonable written detail. Specific patents can also be authorized using clause FAR 52.227-5 or via listed exceptions of FAR 52.227-3. See also **DFARS subpart 227.70--infringement claims, licenses, and assignments** and **28 USC 1498**.

As noted in **DFARS 27.201-1**, "Pursuant to 28 U.S.C. 1498, the exclusive remedy for patent or copyright infringement by or on behalf of the Government is a suit for monetary damages against the Government in the Court of Federal Claims. There is no injunctive relief available, and there is no direct cause of action against a contractor that is infringing a patent or copyright with the authorization or consent of the Government (e.g., while performing a contract)."

There are other ways to reduce the risk of software patent infringement (in the U.S.) as well:

- Some protocols and formats have been specifically devised and reviewed to avoid patents; using them is more likely to avoid problems.
- Prior art invalidates patents. Patents expire after 20 years, so any idea ("invention") implemented in software publicly available for more than 20 years should not, in theory, be patentable. Once an invention is released to the public, the inventor has only one year to file for a patent, so any new ideas in some software must have a patent filed within one year by that inventor, or (in theory) they cannot be patented. See **Prior Art and Its Uses: A Primer, by Theodore C. McCullough**
- OSS can often be purchased (directly, or as a support contract), and such purchases often include some sort of indemnification.
- Various organizations have been formed to reduce patent risks for OSS. The **Open Invention Network** (OIN^{SM}) may in some cases provide some additional protection. OIN purchases patent rights; patents owned by OIN are available royalty-free to any company, institution or individual that agrees not to assert its patents against the "Linux System" (which includes a large set of OSS projects). **The Linux Foundations' Patent Commons** forum is a neutral forum where patent pledges and other commitments can be readily accessed and easily understood.

Q: How can I avoid failure to comply with an OSS license? What are good practices for use of OSS in a larger system?

The following are good practices:

- Educate all software developers that they must comply with all valid licenses - including both proprietary *and* open source software licenses. Explain the basic terms of the most common OSS licenses to them.
- Before including *any* software in a larger system (be it proprietary *or* OSS), review its license to ensure that the license will not impede anticipated uses.
- When including externally-developed software in a larger system (e.g., as a library), make it clearly separable from the other components and easy to update. Commercial software (both proprietary and OSS) is occasionally updated to fix errors (including security vulnerabilities), and your system should be designed so that it is relatively easy to accept these updates.
- Document from where and when any external software was acquired, as well as the license conditions, so that future users and maintainers can easily comply with the license terms.

Releasing software as OSS

Q: Has the U.S. government released OSS projects or improvements?

Yes, both entirely new programs and improvements of existing OSS. There are far too many examples to list; a few examples are:

- **Security-Enhanced Linux (SELinux)**
- **OpenVista**
- **Expect**
- **EZRO**
- Evergreen (by the State of Georgia),
- OpenSSL (this improvement was a Common Criteria evaluation)
- Bind implementation of DNSSEC
- GNAT Ada compiler
- BSD TCP/IP suite

Q: What are the risks of the government *not* releasing software as OSS?

If the government modifies existing OSS, but fails to release those improvements back to the main OSS project, it risks:

- Greatly increased costs, due to the effort of self-maintaining its own version

- Greatly increased costs, due to the effort of self-maintaining its own version
- Inability to use improvements (including security patches and innovations) by others, where it uses a "non-standard" version instead of the version being actively maintained

Similarly, the government develops runs the following risks when it develops new software but does not release it as OSS, it risks:

- Greatly increased cost, due to having to bear the *entire* burden of development costs
- Inability to use improvements (including security patches and innovations) by others, since they do not have the opportunity to aid in its development
- The development and release of a competing OSS project. In this case, the government has the unenviable choice of (1) spending possibly large sums to switch to the OSS project (which would typically have a radically different interface and goals), or (2) continuing to use the government-unique custom solution, leaving the U.S. systems far less capable that others' (including our adversaries)
- Questions about why the government - who represents "the people" - is not releasing software that they paid for back to "the people".

Clearly, classified software cannot be released back to the public as open source software. However, often software can be split into various components, some of which are classified and some of which are not, and it is to these *unclassified* portions that this text addresses.

Q: What are the risks of the government releasing software as OSS?

The key risk is the revelation of information that should not be released to the public. Classified software should *already* be marked as such, of course. This risk is mitigated by reviewing software (in particualr, for classification and export control issues) before public release.

Q: Can government employees develop software and release it under an open source license?

Not under typical open source software licenses based on copyright, but there is an alternative with the same practical effect.

Software developed by US federal government employees (including military personnel) as part of their official duties is not subject to copyright protection and is considered “public domain” (see 17 USC § 105). Public domain software can be used by anyone for any purpose, and cannot be released under a copyright license (including typical open source software licenses).

However, software written entirely by federal government employees as part of their official duties *can* be released as “public domain” software. This is not under a copyright license, it is *absence* of a license. By some definitions this is technically not an open source license (because no license is needed), but “public domain” software can be legally used, modified, and combined with other software without restriction. Thus, “public domain” software provides recipients all of the rights that open source software must provide. An example of such software is **Expect**, which was developed and released by NIST.

Government employees may also modify existing open source software. If some portion of the software was developed by persons who are not US government employees, then the software can be released under copyright license. (See next question.)

(See also **GPL FAQ, Question "Can the US Government release a program under the GNU GPL?"**)

Q: Can government employees contribute code to open source software projects?

Yes, but the following considerations apply:

As stated above, software developed by government employees as part of their official duties is not subject to copyright protection in the United States. If a government employee enhances or modifies a (copyrighted) open source software program, the resulting work is a “joint work” (see 17 USC § 101) which is partially copyrighted and partially public domain. The resulting joint work as a whole is protected by the copyrights of the non-government authors and may be released according to the terms of the original open-source license.

However, the public domain portions may be extracted from such a joint work and used by anyone for any purpose. For computer software, modern version control and source code comparison tools typically make it easy to isolate the contributions of individual authors (via "blame" or "annote" functions).

(See also **Free Software Foundation License List, Public Domain**)

(See also **GPL FAQ, Question "Can the US Government release improvements to a GPL-covered program?"**)

Q: Can contractors develop software for the government and then release it under an open source license?

In many cases, yes, but this depends on the specific contract and circumstances. Under the "default" DFARS and FAR rules and processes, the contractor often keeps and exercise the rights of a copyright holder, which enables them to release that software as open source software (as long as other laws and regulations are met).

For DoD contractors, if the standard DFARS contract clauses are used (in particular DFARS 252.227-7014) then the contractor who developed the software retains the copyright to the software and has the right to release it to others, even if the software was developed exclusively with government funds. In some cases a DoD contractor may be required to transfer copyright to the government for works produced under contract (see DFARS 252.227-7020). If this is the case, then the contractor cannot release the software as OSS without permission, because the contractor doesn't own the copyright.

Contractors for other federal agencies may have a different process to use, but after going through a process they can often release such software as open source software. If the contract includes the typical FAR 52.227-14 (Rights in data - general) clause, without any special alternatives or additions, then the contractor must make a written request for permission to assert copyright in works containing data first produced under the contract. As described in FAR 27.404-3, a contracting officer would generally grant such a request. Certain FAR clause alternatives (such as FAR 52.227-17) require the contractor to assign the copyright to the government. Again, if this is the case, then the contractor cannot release the software as OSS without permission, because the contractor doesn't own the copyright.

There are many alternative clauses in the FAR and DFARS, and specific contracts can (and often do) have different agreements on who has which rights to software developed under a government contract. The FAR and DFARS specifically permit different agreements to be struck (within certain boundaries). Thus, if there is an existing contract, you *must* check the contract to determine the specific situation; the text above merely describes common cases.

Contractors must still abide with all other laws before being allowed to release anything to the public. Obviously, contractors cannot release anything (including software) to the public if it is classified. The release of the software may be restricted by the International Traffic in Arms Regulation or Export Administration Regulation. The release may also be limited by patent and trademark law.

Q: Can the government release software under an open source license if it was developed by contractors under government

contract?

In many cases, yes, but this depends on the specific contract and circumstances. Under the usual "default" rules, the answer is "yes" if it was developed for the DoD under the DFARS. Under the "default" rules, the answer is typically "no" if it was developed for under the default FAR rules (used by many other federal agencies) unless the contract transferred the copyright to the government or was modified in some way to permit it.

If the contractor was required to transfer copyright to the government for works produced under contract (e.g., because the FAR 52.227-17 or DFARS 252.227-7020 clauses apply to it), then the government can release the software as open source software, because the government owns the copyright.

Under the DFARS, which is typically used for DoD contracts, the government can release software as open source software once it receives "unlimited rights" to that software. DFARS 252.227-7014(a)(15) defines "unlimited rights" as "rights to use, modify, reproduce, release, perform, display, or disclose computer software or computer software documentation in whole or in part, in any manner and for any purpose whatsoever, and to have or authorize others to do so". As noted in "Technical Data and Computer Software: A Guide to Rights and Responsibilities Under Federal Contracts, Grants and Cooperative Agreements" by the Council on Governmental Relations (CAGR), "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." In short, once the government has unlimited rights, it has essentially the same rights as a copyright holder, and can then 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.

If the standard DFARS contract clauses are used (see DFARS 252.227-7014), then unless other arrangements are made, the government has unlimited rights to a software component when (1) it pays entirely for the development of it (see DFARS 252.227-7014(b)(1)(i)), or (2) it is five years after contract signature if it partly paid for its development (see DFARS 252.227-7014(b)(2)). Before award, a contractor may identify the components that will have more restrictive rights (e.g., so the government can prefer proposals that give the government more rights), and under limited conditions the list can be modified later (e.g., for error correction). Where possible, software developed partly by government funds should broken into a set of smaller components at the "lowest practicable level" so the rules can be applied separately to each one. Note, however, that this may be negotiated; if the government agrees to only receive lesser rights (such as government-purpose rights or restricted rights) then the government does *not* have the rights necessary to release that software as open source software.

The rules for many other U.S. departments may be very different. Contracts under the federal government FAR, but not the DFARS, often use clause FAR 52.227-14 (Rights in Data - General). If all defaults are accepted, and no additional alternatives/amendments are added, by default the government does not receive the right to distribute to the public software it paid to develop; see FAR 52.227-14(c)(1)(iii). (This is actually a special case; the government normally *does* have the right to public release of copyrighted works it paid to develop.)

There are many alternative clauses in the FAR and DFARS, and specific contracts can (and often do) have different agreements on who has which rights to software developed under a government contract. The FAR and DFARS specifically permit different agreements to be struck (within certain boundaries). Thus, if there is an existing contract, you must check the contract to determine the specific situation; the text above merely describes common cases.

If the intent of a contract is to develop software to be released as open source software, it is best to expressly include release as OSS as part of the contract. This makes the expectations clear to all parties, which may be especially important as personnel change.

Other laws must still be obeyed. Classified information may not be released to the public without special authorization to do so. The release of the software may be restricted by the International Traffic in Arms Regulation, or Export Administration Regulation. The release may also be limited by patent and trademark law.

Q: Does releasing software under an OSS license count as commercialization?

In most cases, yes. 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, *even if* it was originally developed using public funds.

This does not mean that organizations will automatically arise to help develop/support it. Whether or not this will occur depends on factors such as the number of potential users (more potential users makes this more likely), the existence of competing OSS programs (which may out-compete the newly released component), and how difficult it is to install/use. Thus, components that have the potential to (eventually) support many users are more likely to succeed. Similarly, delaying a component's OSS release too long may doom it, if another OSS component is released first. If the OSS is intended for use on Linux/Unix systems, **follow standard source installation release practices** so that it is easier for users to install.

Q: What license should the government or contractor choose/select when releasing open source software?

It depends on the goals for the project, however, here are some guidelines:

- **Public domain where required by law.** You *must* release it as "public domain" (when releasing it at all) if it was developed by a US government employee as part of their official duties. Otherwise, choose some existing OSS license, since all existing licenses add some legal protections from lawsuits. (The "MIT license" is similar to public domain release, but with some legal protection from lawsuits.)
- **Release modifications under same license.** If it is a modification of an existing project, or a plug-in to it, release it under the project's original license (and possibly other licenses). This way, the software can be incorporated in the existing project, saving time and money in support.
- **Consider anticipated uses.** If it must work with other components, or is anticipated to work with other components, ensure that the license will permit those anticipated uses. In particular, will it be directly linked with proprietary or classified code?
- **Make sure it's really OSS.** Choose a license that has passed legal reviews and is clearly accepted as an OSS license. Choose a license that is recognized as an **Open Source Software license by the Open Source Initiative (OSI)**, a **Free Software license by the Free Software Foundation (FSF)**, *and* is acceptable to widely-used Linux distributions (such as being a **"good" license for Fedora**).
- **Use a widely-used existing license.** Choose a widely-used existing license; do *not* create a new license. This eliminates future incompatibility and encourages future contributions by others. **Bruce Perens noted back in 1999**, "Do not write a new license if it is possible to use (a common existing license)... The propagation of many different and incompatible licenses works to the detriment of Open Source software because fragments of one program cannot be used in another program with an incompatible license." **Many view OSS license proliferation as a problem**; **Serdar Yegulalp's 2008 "Open Source Licensing Implosion" (InformationWeek)** noted that not only are there too many OSS licenses, but that the "consequences for blithely creating new ones are finally becoming concrete... the vast majority of open source products out there use a small handful of licenses... Now that open source is becoming (gasp) a mainstream phenomenon, using one of the less-common licenses or coming up with one of your own works against you more often than not". As an aid, the **Open Source Initiative (OSI) maintains a list of "Licenses that are popular and widely used or with strong communities"**. Another useful source is the list of **licenses accepted by the Google code hosting service**. See the licenses listed in the FAQ question "What are the major types of open source software licenses?".
- **Choose a GPL-compatible license.** The GNU General Public License (GPL) is the most common OSS license; while you do not need to use the GPL, it is often unwise to choose a license incompatible with the majority of OSS. Thus, avoid releasing software under only the original ("4-clause") BSD license (which has been replaced by the

"new" or "revised" 3-clause licence), the "Academic Free License" (AFL), the now-abandoned "Common Public License" 1.0 (CPL), the "Open Software License" (OSL), or the "Mozilla Public License" (MPL).

- **Choose a license that best meets your goals.** Choosing between the various options - particularly between permissive, weakly protective, and strongly protective options - is perhaps the most difficult, because this selection depends on your goals, and there are many opinions on which licenses are most appropriate for different circumstances. A "permissive" license permits arbitrary use of the program, including making proprietary versions of it. A "protective" license "protects" the software from becoming proprietary, and instead enforces a "share and share alike" approach between parties. A "weakly-protective" license is a compromise between the two, preventing the covered library from becoming proprietary yet permitting it to be embedded in larger proprietary works. If the goal is maximize the use of a technology or standard in a variety of different applications/implementations, including proprietary ones, permissive licenses may be especially useful. However, if the goal is to encourage longevity and cost savings through a commonly-maintained library or application, protective licenses may have some advantages, because they encourage developers to contribute their improvements back into a single common project. In many cases, weakly protective licenses are used for common libraries, while strongly protective licenses are used for applications. Common licenses for each type are:
  - Permissive: MIT, BSD-new, Apache 2.0
  - Weakly protective: LGPL (version 2 or 3)
  - Strongly protective: GPL (version 2 or 3)

Licenses that meet all the criteria above include the **MIT license**, **revised BSD license**, the **Apache 2.0 license** (though Apache 2.0 is only compatible with GPL version 3 not GPL version 2), the **GNU Lesser General Public License (LGPL)** versions 2.1 or 3, and the **GNU General Public License (GPL)** versions 2 or 3.

In some cases, it may be wise to release software under multiple licenses (e.g., "LGPL version 2.1 and version 3", "GPL version 2 and 3"), so that users can then pick which license they will use. This can increase the number of potential users.

Q: How should I create an open source software project?

First, get approval to publicly release the software. One way to deal with potential export control issues is to make this request in the same way as approving public release of other data/documentation.

If it is an improvement to an existing project, release it to the main OSS project, in whatever format they prefer changes. Many prefer "unified diff patches", generated by "diff -u" or similar commands. Most projects prefer to receive a set of smaller changes, so that they can review each change for correctness.

If it is a new project, be sure to remove "barriers to entry" for others to contribute to the project:

- Use a common OSS license well-known to be OSS (GPL, LGPL, MIT/X, BSD-new, Apache 2.0) – don't write your own license
- Establish project website. Typically this will include source code version management system, a mailing list, and an issue tracker.
- Document the project's purpose, scope, and major decisions - users must be able to quickly determine if this project might meet their needs.
- Use typical OSS infrastructure, tools, etc. Requiring the use of very unusual development tools may impede development, unless those tools provide a noticeable advantage.
- Maximize portability, and avoid requiring proprietary languages/libraries unnecessarily. The more potential users, the more potential developers.
- The released version *Must run*. Small-but-running is better than big-and-not.
- Establish vetting process(es) before government will use updated versions (testing, etc.)
- Determine if there will be a government-paid lead.

Some documents that may help include:

- **"Producing Open Source Software: How to Run a Successful Free Software Project" by Karl Fogel**
- **Free Software Project Management HOWTO**
- **Software Release Practice HOWTO**
- **Recognizing and Avoiding Common Open Source Community Pitfalls**

Q: In what form should I release open source software?

OSS should be released using conventional formats that make it easy to install (for end-users) and easy to update (for potential co-developers). These formats may, but need not, be the same.

If you are releasing OSS source code for Unix-like systems (including Linux and MacOS), you should follow the usual conventions for doing so as described below:

- **Releasing Free/Libre/Open Source Software (FLOSS) for Source Installation**
- **GNU Coding Standards, especially on the release process**
- **Software Release Practice HOWTO**

Q: Where can I release open source software that are new projects to the public?

You may use existing industry OSS project hosting services such as **SourceForge**, **Savannah**, **Tigris**, **Google code**, **Apache Software Foundation** or **Microsoft CodePlex**. Each hosting service tends to be focused on particular kinds of projects, so prefer a hosting service that well-matches the project. Using industry OSS project hosting services makes it easier to collaborate with other parties outside the U.S. DoD or U.S. government.

DISA's **Forge.mil** is "a family of services provided to support the DoD's technology development community. The system currently enables the collaborative development and use of open source and DoD community source software. These initial software development capabilities are growing to support the full system life-cycle and enable continuous collaboration among all stakeholders including developers, testers, certifiers, operators, and users." It uses a variant of the software used by SourceForge.

If the project is likely to become large, or must perform filtering for public release, it may be better to establish its own website. Note that many of the largest commercially-supported OSS projects have their own sites.

Community Sites about OSS

Q: Where do OSS developers congregate and what conferences should I go to?

An outside DoD/IC discussion list can be found at: **Military - Open Source Software**.

The DoD CIO does not endorse any specific event or conference. That said, there have been a few conferences specifically focused on OSS in the government or military context, at which DoD CIO personnel have presented information on DoD policy and OSS. For example, in August 2009, there was a Military-OSS working group meeting in Atlanta, Georgia, info here **Mil-OSS**. In November 2009, The Government Open Source Conference (**GOSCON**) will be held in Washington, DC.

Atlanta, Georgia, info here **MIPOSS**. In November 2009, The Government Open Source Conference (**GOSCON**) will be held in Washington, DC.