



Photo Courtesy of the Maryland Governor's Office

# THE FORTRESS AND THE BAZAAR: OPEN-SOURCE AND DOD SOFTWARE

*David Lechner and Harold Kaiser*

This article discusses the application of the open-source software project model for the Department of Defense, a software systems development that uses the tenets first laid out in an original paper entitled “Cathedral and the Bazaar,” written by Eric S. Raymond in 1998 explores the implications as applied to defense weapons systems software, in addition to examining the attributes, problems, and benefits of open-source software.

**T**his work is based on a paper written by Eric S. Raymond entitled “Cathedral and the Bazaar,” which is widely considered the principal manifesto of the open-source software movement. We provide a brief overview of the Raymond paper, then attempt to explain the ideas of that document in the terms and methods of the defense-industrial establishment in America. The goal of promoting open-source in defense projects and procurement should be emphasis on lower costs, new functionality, and improved software reliability for the products used by our military.

## SUMMARY OF RAYMOND’S “CATHEDRAL AND THE BAZAAR” PAPER

In his paper, Raymond discusses the historical model for software development and compares it to building cathedrals, a slow and laborious effort with exacting methods carefully applied. Each brick and beam was carefully planned and the structure was erected with painstaking craftsmanship to meet the planned design and symmetry. The bazaar, by contrast, was often created ad-hoc and in an evolutionary fashion. The bazaar started with a few street vendors and was later built up by additional vendors and merchants, each staking out a piece of the market place as their own and

maintaining and adding to their stall until a full-blown Agora was in place. Clearly each approach served its purpose well, but the cathedral method is rigid and static. It needs the contribution of each part exactly as designed in order to stand, and has difficulty adapting to any other shape or location. The open-source movement is all about flexibility and evolutionary development.

Raymond went on to review nineteen axioms of open-source development by discussing his experience developing “Fetchmail,” a Linux application used to forward e-mail. These axioms are the basic tenets of the open-source community, and Raymond discussed them with detailed analysis and examples in the original paper. In brief form, these tenets are:

1. Every good work of software starts by scratching a developer’s personal itch.
2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
3. “Plan to throw one away; you will anyhow” (Brooks, 1975).
4. If you have the right attitude, interesting problems will find you.
5. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.
6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.
7. Release early. Release often. And listen to your customers.
8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.
9. Smart data structures and dumb code works a lot better than the other way around.
10. If you treat your beta-testers as if they are your most valuable resource, they will respond by becoming your most valuable resource.
11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.
12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.
13. Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.

14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.
15. When writing gateway software of any kind, take pains to disturb the data stream as little as possible—and *never* throw away information unless the recipient forces you to.
16. When your language is nowhere near Turing-complete, syntactic sugar can be your friend.
17. A security system is only as secure as its secret. Beware of pseudo-secrets.
18. To solve an interesting problem, start by finding a problem that is interesting to you.
19. Provided the development coordinator has a medium at least as good as the Internet and knows how to lead without coercion, many heads are inevitably better than one.

---

***Linux is one of the best known and most successful examples of open-source software, but a review of Sourceforge.net showed that there are now over 20,000 open-source software projects involving over 70,000 developers worldwide targeting many different operating systems.***

---

Raymond discusses these issues in detail, but it is his last item (#19) that is the essence of an open-source approach. It describes the advantages of multiple groups working on code to provide more innovation and faster debugging cycles, discussed primarily in terms of the Linux operating system. Linux is one of the best known and most successful examples of open-source software, but a review of Sourceforge.net showed that there are now over 20,000 open-source software projects involving over 70,000 developers worldwide targeting many different operating systems. After discussing some problems with open-source for defense use, this article will develop new axioms, discuss advantages for Department of Defense (DoD) use, and develop some practical methods for the DoD to promote open-source solutions.

## WHAT IS “OPEN-SOURCE?”

The principal trait of an Open-Source (OS) project is simply stated in the name, but it involves more than just open-source code. “Open-source” projects are collaborative, rapidly updated and rapidly released, and pushed by highly motivated volunteers working with a diversity of interests, skills, and hardware sets. Some people work on open-source projects and are paid for it. Some projects move more slowly. Some only target one OS platform, and some many. This article will focus on how a generic open-source approach could be applied to DoD projects.

## DEFENSE PROJECT SOFTWARE DEVELOPMENT: AN HISTORICAL OVERVIEW

Since the earliest use of computers, all the rigidity of process and formalized methods that the military systems engineers could muster have been applied to software projects, making them extremely rigid and well documented (e.g., DoD Standard-1467 and MIL-STD-2168). Several arguments for this are listed below, but we discuss how they do not stand up well.

### THE CURRENT DOD SOFTWARE METHODS ILLUSTRATE AND EMBODY THE CATHEDRAL BUILDING METHODS

The DoD processes evolved due to several factors, some of which can win instant sympathy from the reader as being not only reasonably justified, but critically important. Some examples are:

- The system-fired missiles and munitions needed every safeguard possible to prevent accidental firing or targeting.
- The lives of the soldiers depended on the project. They deserved all the care and testing that could be applied in order to produce the best possible product.
- The taxpayers’ dollars were at work; reducing the possibility of mistakes or bugs was paramount.
- National security depended on the software program being perfect. To protect the country, we could spare no possible pain to make it so.

Typical military software projects in the 1970s and 1980s required formally documented and carefully detailed designs, involving “waterfall” builds with incremental steps for adding functions and exhaustive testing at each step. The code had to be well documented, with interface design documents, requirements definition documents, system design documents, database design documents, program development folders, and test plans and reports. Usually the code was documented to transition it to a

government organization. Once the “cathedral” was completely built, the government team was to “move in” and take on ownership and maintenance.

Systems that actually fired munitions were even more carefully tested. The live-fire testing and certification processes for missile and munitions-type projects are still very extensive. These testing and certification processes often evolved in a reactionary way to resolve problems associated with the previous fielding of poorly designed systems and were intended to provide a valuable check on the work of vendors developing weapons systems that are dangerous by design. In fact, the nature of military software is critically enabling in terms of achieving functional performance. This made the DoD software project much more than a cathedral in style. It became a “fortress,” and its details had to be constructed and managed as any good military fortress was: in secrecy.

### **THE CLOAK OF NATIONAL SECURITY SCREENS DOD SOFTWARE CODE FROM CO-DEVELOPERS SO WELL THAT RESOLVING PROBLEMS IS DIFFICULT AND TIME CONSUMING**

Since the original developing group is the only group to get effective exposure to the source code, there are actually no co-developers. Thus a very biased group of developers (i.e., the proud “parents” of the “baby”) can only see the problem from their singular perspective. This keeps the code from getting wider peer review, more thorough testing, unbiased discussion, or more diverse usage. In order to prevent the dissemination of software source code to threatening nations, some form of continued security around the software investment made within the weapons systems is essential. To say that no other vendor or R&D group is patriotic enough to protect the software, however, is ludicrous.

### **CONTRACTORS AND LABORATORIES ARE MOTIVATED TO WITHHOLD THE SOFTWARE SOURCE CODE AT ALL COSTS TO PROTECT THEIR PROPRIETARY OR BUSINESS INTERESTS**

This motivation is due in large part to the desire of the DoD companies to justify sole-source contracts or establish a favored business position. The possession and working knowledge of the applications software is one of the best ways to maintain a privileged “insider” position on a program, normally securing years of business revenue when successful. The developer is normally funded to fix software bugs, revise the user interfaces, provide training and logistics support, and continue integrating hardware and resolving hardware obsolescence problems. This can amount to considerable business revenue that continues long after the system is delivered. One classic approach to maintaining the best business position for all this work is to minimize disclosure of the software source code.

Normally, software documentation, carefully crafted in true cathedral or fortress style, is eventually delivered in order for the Government to exercise their rights to full ownership. This typically involves sending the boxes of paper and computer disks to a Government engineering laboratory. These laboratories, often lacking the equipment or expertise to really use the source code, are often unable to do more than lock up

the code in a secure vault, ultimately to be discarded many years later. Sometimes the Government itself acts to protect the developer's monopoly by limiting distribution of the code. The rationale is that the code has been paid for and now works, so that paying any other vendor to work on it is a waste of scarce resources and unnecessary. This logic fails in several aspects, failing to recognize key issues:

1. Secondary benefits arise from the distribution of knowledge to other projects, disciplines, or programs as code is re-written and re-used in ways never expected. Other program managers (PMs) can decide how best to utilize the code as their own program needs and budget dictates.
2. Maintenance benefits occur with open-source as other organizations or developers review and utilize the code and discover and remove bugs. The larger number of developers will have different and new perspectives and suggestions for improvement. The original owners of the code can then take advantage.
3. The laziness (or greed) of other activities or vendors gives incentives to leverage existing work as much as possible if given the access. Open-source code re-use can work to significantly lower new project costs and improve schedule performance.
4. Some companies will invest their own funds to upgrade existing code under Internal Research and Development (IRAD).
5. Sometimes Government PMs may invest funding to build on the existing work in a new way with other vendors.
6. The policy does not recognize the public ownership already established by using the taxpayers' funding, and that the application is already owned collectively by the public and other companies, not by a department or a single vendor.
7. The value to other projects can only be judged by their developing teams.

### COMMON SOFTWARE WAS HISTORICALLY NOT "OPEN-SOURCE"

Commonality was often championed as a cost savings, but actually promoted a business position. Commonality was resisted by others to defend their business positions.

Historically, commonality tended to be championed by the vendors that owned and benefited from selling the common product or solution. The goal was simple: take over the business content of the other "non-common" vendors. In reality, there are differences in mission, environment, and interfaces that exist from ship-to-ship or use-to-use. Adapting existing software to a new platform or environment takes familiarity with both the new environment and the software.

Typically, an incumbent organization is more familiar with the mission and platform, while the “common” software team knows the new software code. Both of the organizations have good claims to being able to efficiently adapt the code to the new mission. The interfaces and mission needs involve more complex integration issues, whereas the new code is probably well documented and evolutionary. The benefits of a having several groups using the software code and working out its bugs are fairly certain, but the ability of a single organization to support multiple customers with different missions is questionable.

### **SINGLE-MAINTENANCE ENTITIES WERE JUSTIFIED TO CONTAIN COSTS, BUT THE CODE SUFFERED FROM A REDUCED KNOWLEDGE BASE**

Raymond says several times that multiple sets of eyes can find software bugs faster. The original development team is unlikely to even recognize some problems. They are emotionally biased towards the design they created. The Fast Fourier Transform, as an example, can be coded hundreds of ways, but all are essentially the same algorithm. Some of these methods are significantly faster than others and some depend more on the use of a specific machine. Many groups use the original algorithm, while others were intrigued and developed new implementations. This would not have happened if a group had declared a single method as “best” and “common” (or worse yet, kept it a secret!).

The DoD offices are often challenged to fund a single software maintenance team. Long-term software maintenance can be enormously expensive. The DoD Users provide long lists of complaints and suggestions, but no licensing fees or support service payments. Would the software maintenance costs be higher if the code was supported by multiple groups? With only one support agent each, correction comes slower and with greater expense. Problems that take one group months to find may be quickly found by other organizations. Collaborative debugging versus isolationist debugging is what the open-source community claims will make projects more successful. Only experimentation can decide if the collaborative support is less expensive, but if it is, the net savings could be directed into new functionality and improvements.

### **FUNDED SOFTWARE MAINTENANCE TEAMS ARE BIASED, YET HISTORICALLY ALSO MANAGE AND RECOMMEND THE UPGRADE EFFORTS**

Financial neutrality is important to keep upgrade and maintenance decisions unbiased. An active community of co-developers can help keep decisions unbiased. The potential benefits of proposed changes accrue to all of the developers. This axiom does not preclude the use of available expertise, including the original developer. Allowing an overly strong voice by a developer that is funded to maintain the code, however, can lead to long-term bias in the methods and types of changes being worked on.

## PROPOSALS MUST OFTEN BE WRITTEN QUICKLY AND WITHOUT CRITICAL INFORMATION ON THE PROJECT'S SOFTWARE

The DoD customers often undervalue the effort it takes to understand software programs and complex systems deployed today, and companies (when solicited) are given little information to review before being asked to provide their best ideas. Informed proposals come from informed teams, not out of the clear blue sky.

Distributed knowledge through open-source code could provide a detailed understanding of the state of the practice in solving a systems problem. This understanding allows others, when invited, to submit their best ideas. Some project managers work hard to keep potential bidders informed and others do not. Without perspective on what the project is doing and what the current problems are, it is hard to offer improvements.

---

***Distributed knowledge through open-source code provides a detailed understanding of the state of the practice in solving a systems problem.***

---

Even the customer's views are limited by their perspective, and a Request for Proposal (RFP) author may not see that a current design is lacking or problematical. Not recognizing the need of an informed review can cause a negative reply or a "left-field" response, which wastes the bidder's proposal budget and the reviewer's time.

Allowing informed and innovative proposals recognizes the potential value provided by new solutions. An open-source environment would allow developers to propose improvements based on their detailed and informed view of the source code, its problems, and their own strengths and experience.

## THE DOD OPERATIONAL TEST AND EVALUATION (OT&E) PROCESS ENCOURAGES A RIGID "ONE-SHOT" CATHEDRAL APPROACH

The DoD currently uses formal testing methods that encourage programs to avoid frequent small tests in favor of single large test events, even though the OT&E offices encourage frequent interaction with a program. Most programs, however, will subconsciously try to "get everything right" before putting the glaring spotlight of formal OT&E on their system. Not wanting to risk negative reviews when the system is in its infancy, they avoid frequent incremental testing in favor of fewer and larger events that are scripted in nature. This is a very important problem, since corrections are less expensive when feedback is earlier and more incremental. These practices violate the "release early" and "release often" axioms.

## THE PROPOSED OPEN-SOURCE MODEL FOR DEFENSE SOFTWARE PROJECTS

So far, we have reviewed the problems that exist with DoD software projects, and concerns that must be addressed regarding national security considerations. Listed below is a new set of axioms with at least one vision on how open-source software methods could be applied to defense systems software projects:

1. The code should be freely licensed to all willing co-developers.
2. The distribution must be controlled due to security conditions.
3. The distribution must limit technology transfer to international entities.
4. The distribution should not limit or constrain the platform (OS) use.
5. The soldiers, sailors, and pilots that comprise the user communities should be provided at least a partial distribution of the source code (especially for the user interfaces), as they are co-developers.
6. There should still be strict configuration management of installed software baselines.
7. Changes should not normally affect original functionality unless bugs are being fixed.
8. Commercial (generic) platform options for running the software are preferred.
9. A single lead activity or vendor should still be charged and funded to integrate proposed changes into a CM-controlled distribution and provide associated integration, stress testing, and documentation and logistics (e.g., operator manuals, training, etc.) updates for the system, even if open-source methods are used to evolve the software code.
10. The OT&E process must encourage more incremental testing and feedback.

Under the proposed open-source model, the software application code would be provided to any or all registered university labs, Government labs, and vendors that are willing to take the proper safeguards and precautions to protect the national security aspects involved. Then they could either invest their own resources or propose to a sponsoring agency to fund work using the code.

The originating office may alternately establish their software agent as the control point for incorporating changes developed by others and re-distributing the code, as is done on Linux OS, or might allow a completely ad-hoc distribution method as used on many smaller projects. All changes made to the application's code, however, should be

provided to the originating team and all other registered development groups. They can review it and incorporate it at their discretion. Other co-developers may see a potential improvement and would have the option of implementing it on their projects or of proposing sponsored project work to any of several potential funding sponsors.

This proposed arrangement does not preclude a sponsoring program office from developing an entirely new application. The existence of a basic version of the software code for military applications, however, gives them an example to leverage fully (if possible), borrow from (if reasonable), or move on past (if necessary).

---

***A critical aspect of bringing open-source methods to DoD use is using voluntary processes and avoiding the pitfall of establishing a new DoD office chartered with managing and mandating open-source practices.***

---

A critical aspect of bringing open-source methods to DoD use is using voluntary processes and avoiding the pitfall of establishing a new DoD office chartered with managing and mandating open-source practices. A voluntary method that encourages participation would encourage participants to listen to their users and customers. Mandated methods tend to turn into monopolies with all of their inherent problems. Establishing a central registry of code and its originators, however, might be very beneficial.

## EXAMPLES OF OPEN-SOURCE AT WORK WITHIN THE DOD

The Global Command and Control System, Maritime (GCCS-M) has been a recent and significant example of the open-source model at work within the DoD. All Navy ships, airplanes, and units coordinate their tactical command and control movement using this major DoD application. It was developed over many years and at tremendous expense. Before 1998, only one vendor had full access to and working knowledge of the source code. Then the Commander of the Space and Naval Warfare Systems Command, Rear Admiral John A. Gauss, made a decision with staffing coordination to open up the source code by providing it to several other software development activities. This decision was contested initially by the vendor, but the “opening” was implemented about a year later. After being made “open,” over a dozen organizations paid a single-time developmental license (used to fund initial set-up and technical support) to the Navy and have obtained a copy of the program and all of its build packages. This was not truly a “free” deal, as the license fee is charged for the initial support. It is, however, truly “open,” since all of the source code is provided. Reportedly, the vendor was

initially suspicious of the Navy's intentions, but since making GCCS-M open-source, positive benefits have included identification of potential improvements, expanded use of the software, and increased opportunity for the vendor.

Another interesting example of software commonality and openness is the Core Architecture Data Model (CADM) being developed in Extensible Markup Language (XML) for all of the DoD services by the Defense Information Systems Agency (DISA). A common data model allows software programs to integrate in new ways and avoid having multiple definitions of the same item. Developers that leverage the CADM can utilize an off-the-shelf XML "parser" module to provide the interface services for XML type data, thus eliminating the development of huge amounts of source code (data handling and error checking is a large part of a program). This is an extremely powerful philosophy for the DoD, placing emphasis on a common view of the underlying data that the code manipulates instead of a common code set that may be hard to port.

---

***If even a small percentage of the software maintenance costs were reduced, the DoD cost savings would be significant.***

---

## DOD BENEFITS FROM USING OPEN-SOURCE METHODS

A "DoD-Foundation Class Library" containing commonly used algorithms and application modules would be openly shared and maintained amongst programs. It would provide tremendous benefits, promoting software functions from successful programs while allowing improved debugging of errors or elements missing from others. Successful attributes of a DoD open-source approach would include:

- More application codes being re-used, lowering non-recurring costs.
- The cost of software maintenance and improvement is lower, with faster debug cycles and more groups familiar with the source code and technically capable of working with it.
- Open-source software allows greater flexibility in choosing hardware solutions, since the single-source provider was historically biased.

- The better parts will get re-used and leveraged as much as possible (selective evolution), while the “buggy” or poorer parts will get replaced or fixed. This is a genetic evolution model of “survival of the fittest.”

Examples of the types of software modules that could be shared amongst programs with great benefit to the DoD as a whole are sonar and radar signal processing, image processing, track management, phased-array beamforming, graphical user libraries, scene rendering, and data fusion. If even a small percentage of the software maintenance costs were reduced, the DoD cost savings would be significant.

### PROBLEMS WITH AN OPEN-SOURCE MODEL FOR DEFENSE SYSTEMS

There are some potential problems with the use of open-source software in a defense system application, such as increased cost to national security. The problems must be balanced against the gains for each particular program and measured against the potential benefits. Some of the arguments are:

- *This is expensive; co-developers are not really free.* This is an important issue. A legion of co-developers capable of working on complex software systems is expensive. This is certainly true if a small project is viewed individually. When viewed generically, many company, laboratory, and university efforts are all relevant to big projects and are already funded. When a software module is used by four or five programs, however, the cost-per-program for the life-cycle support decrease (in aggregate) compared to duplicative efforts funded by multiple programs. The cost of debugging should become lower through repeated porting and broadened team support.
- *The platform can be a big problem.* The software may need a specific type of system on which to run. This problem has few simple solutions. Open-source methods and porting for other project use will tend to eliminate platform-specific nuances and allow deployment on less expensive commercial-equivalent systems as time goes by.
- *National Security is threatened through widespread code release.* In reality, National Security is threatened more by expensive and bug-ridden software. Technology transfer and secure vaulting and transfer rules can protect the national interests. The groups and companies participating are already developing military systems.
- *Nothing is really gained; the algorithms are published in journals to publicize the state of the art, and the coding of the algorithm is trivial.* This argument does not recognize the complexity of the software involved and the potential benefits and budget savings through use of a previously debugged version of the algorithm.

- *Competitors will take advantage of “open-source” to develop preferred business positions.* It is a bold move to register and provide software modules to competitors, some of whom may be large and aggressive. There have been notable examples of companies that obtained existing software from a competitor and leveraged it to their advantage. The use of a feedback system and “registry” web site should provide a means of identifying those companies that were obtaining open-source code but not contributing to the “open” efforts or identifying how they use the code. This imbalance could be used to justify a ban on a group from receiving further code developed by others. This would put financial pressure on such companies, since their proposals would become more expensive and receive less favorable financial and technical reviews.
- *The GPL License is confusing. What does it mean for the DoD?* This article does not attempt to cover the differences in the various open-source licenses, which is an important topic that needs review and comprehensive study. A specialized version of the GPL may need to be developed for DoD use.

## SHARPENING THE COMPETITIVE EDGE WITH OPEN-SOURCE

One significant reason that corporations tightly protect their software source code is to maintain a singular business position through collecting intellectual property (IP). Despite the fact that the taxpayer often funded the code development, the old adage “possession is nine tenths of the law” applies. Senior business managers can be understandably horrified at the prospect of releasing the source code for a project to potential competitors. Certainly the DoD sector is no different in this concern than the private sector, yet companies such as SGI and IBM are now pursuing open-source strategies for significant portions of their software IP base. Strong factors can help to give open-source policies more business importance for the military sector than for the private sector, such as:

- Regardless of who else gets the source code, the developing agent will always have stronger bragging rights and credentials when trying to develop new business. Most customers will prefer to deal with the originating agent, since they are clearly a sharp group. This may, however, take some reminders.
- Wider use of one’s software on other programs can open doors to those programs and provide new customers. Getting into a new program or project has always been very difficult in the military business. The advantage of having one’s software modules already in use on that program would be a major selling point in claiming capability in supporting the needs of that customer. This implies that some method of tracking the use of that code is available. Registration for updates should be sufficient.
- The companies that support DoD open-source solutions can claim enormous credibility with their customers, the soldiers, and the taxpayers. This is most

important in areas of basic research, where continued funding is dependent on demonstrated original work and program transitions.

- The money saved by avoiding repetitious software is a savings in personal and corporate taxes. In comparison to a single individual's personal taxes the savings may represent a lifetime of earnings.
- The money saved in software redevelopment could be used more productively on new functionality. Today's military systems are better than ever, but more capability could often be utilized to face the complex and dense multi-threat environments. Developing that functionality is more challenging than basic building block code.
- Open-source policies make good business sense on their own financial merit, since they can lower the cost of developing and supporting software (cost centers or revenue drains) while leveraging non-employee (i.e., free) people working to improve the systems your company is selling. This happens without major costs to your organization, usually with only minor assistance, to promote your original IP content to new customers.

Some of these benefits are intangible. The bottom-line financial incentive is necessary to sway senior management. Benefits should start to accrue with even modest use of open-source policies and should be monitored and measured if possible.

## HOW THE DOD CAN PROMOTE OPEN-SOURCE

There are several ways that the DoD can encourage the use of open-source software. Of particular interest are voluntary methods, since mandated policy efforts, such as the Tactical-Digital Standards or Military Standard (MIL-STD-1499), have historically proven ineffective. Some voluntary methods that we have identified are:

1. Request notification and disclosure if any open-source module or code is planned or in use. Take that notice back to originating companies. This could be required in proposals, design reviews, and in delivered code and documentation. This notification would help to initiate feedback from the Government back to the original developers.
2. Include open-source data in proposal requirements, and make it an evaluated criteria, and offer award criteria "points" to those with positive policies.
3. Provide pre-approved release of source code to other domestic DoD vendors. This puts the process in a streamlined mode of operations and prevents bureaucracy. This also allows the developing company to track those companies that are using their software and code and to send them future updates or releases. This allows them to boast about how their work is benefiting other projects.

4. Establish a central registry or web site for companies and researchers to access and review what software modules are available to leverage. This would allow companies or customers to check a vendor's support for open-source through their willingness to "register" software modules.
5. Actively identify and register DoD laboratory-developed software modules as available to all reasonable DoD vendors. Since the DoD labs are already public entities, those labs can be directly tasked and funded to take a leadership position in providing open-source software modules.
6. Be patient with companies trying to develop an open-source methodology. In some cases a business interest may delay the release of code, and this may be unavoidable.
7. Identify "in possession" software already delivered to the Government and register it as available. Most development or R&D programs include extensive requirements for delivering code and documentation.
8. Encourage a more incremental and open OT&E system with more frequent feedback and reviews. This should help achieve a more build/test/build methodology.

## HOW DOD VENDORS CAN INITIATE OPEN-SOURCE POLICIES

There are ways that a company can promote open-source practices with small steps and assist the development of a larger but fair business environment. These steps start small (though still brave) and then grow in scope:

1. Offer limited release of source code to laboratories and universities. These are groups less likely to compete directly for business, and the release creates something like "free IRAD" work for the company.
2. Post public web site notification that select modules are available, either on the company's own web site or on a registry. Initially, the modules could be older or less-vital ones, or perhaps prototypes that need help.
3. Release executable library software modules as dynamic link libraries with application programmer interface (API) information. This is a transitional step, since the source is not really available, and a form of protection is still offered to company intellectual property. This release to other vendors, however, allows some of the benefits to accrue, while still retaining IP content.
4. Fully release application and source code for application programs. This would require review of security requirements and customer coordination. It is the boldest

of the steps we can identify, but provides the most benefits to both the developer and the DoD community as a whole.

## SUMMARY

Open-source projects are collaborative, rapidly updated and rapidly released, and pushed by highly motivated volunteers working with a diversity of interests, skills, and hardware sets. Benefits of open-source for the DoD include greater reliability, lower software development and maintenance costs, and more rapid evolution. Steps that the DoD can take to promote OS include making laboratory code available, creating a code catalog or registry, and making it a proposal evaluation and award criteria. To achieve these benefits the movement to open-source policies must be made jointly by the industry vendors and DoD customers and led by the DoD's senior management.



**David Lechner** has 20 years of experience in Department of Defense systems development and acquisition. His civil service career included NAVAIR(REWSON), NAVELEX (PMW143), GSA (FEDSIM), and NAVSEA (PMS425). He holds a B.S. in Electrical Engineering from Carnegie Mellon University, an M.E.Ad. from George Washington University, and an M.S. in Computational Physics from George Mason University. He spent 5 years with DRS Electronics Systems on combat systems programs and currently works for GeoLogics Co.

E-mail address: [Lechnerd@wdn.com](mailto:Lechnerd@wdn.com)



**Harold Kaiser** is the Executive Director for Radar Programs, with 30 years' experience at DRS Technologies, Inc, responsible for managing various sonar and radar engineering development and manufacturing programs. He holds a degree from RCA Institute of Technology, and received a diploma from the Defense Acquisition University (formerly DSMC) Program Management Course. Kaiser earned Program Management Institute, PMP status in 1992 and is currently preparing for renewed PMP certification.

E-mail address: [Kaiser@drs-c3.com](mailto:Kaiser@drs-c3.com)

## AUTHOR BIOGRAPHY

## REFERENCES

- Bezroukov, Nikolai. (1999, October). Open-source software development as a special type of academic research (A critique of vulgar Raymondism). *First Monday*, 4(10). Retrieved on November 21, 2005, from [http://www.firstmonday.org/issues/issue4\\_10/bezroukov/index.html](http://www.firstmonday.org/issues/issue4_10/bezroukov/index.html).
- Brooks, Fred P. (1975). *The mythical man month*. Boston: Addison-Wesley.
- Raymond, Eric S. (1998, March). The cathedral and the bazaar. *First Monday*, 3(3). Retrieved on November 21, 2005, from [http://www.firstmonday.org/issues/issue3\\_3/raymond/index.html](http://www.firstmonday.org/issues/issue3_3/raymond/index.html).