Benefits of Open Source Software in Defense Environments

Daniel Russo
Consorzio Interuniversitario Nazionale per l'Informatica (CINI)
dr.daniel.russo@gmail.com

Abstract

Even though the use of Open Source Software (OSS) might seem paradoxical in Defense environments, this has been proven to be wrong. The use of OSS does not harm security, on the contrary it enhances it. Even with some drawbacks, OSS is highly reliable and maintained by a huge software community, thus decreasing implementation costs and increasing reliability. Moreover it allows military software engineers to move away from proprietary applications and single vendor contracts. Furthermore it decreases the cost of long term development and lifecycle management, besides avoiding vendor's lock in. Nevertheless deploying open source software deserves an appropriate organization of its lifecycle and maintenance, which has a relevant impact on the project's budget that cannot be overseen. In this paper we will describe some major trends in OSS in Defense environments. The community for OSS has a pivotal role, since it is the core development unit. With Agile and the newest DevOps methodologies governmental officials could leverage OSS capabilities, decreasing the Design (or Technical) Debt. Software for Defense purposes could perform better, increase the number of the releases, enhance coordination through the different IT Departments (and the community) and increase release automation, decreasing the probability of errors.

1. Introduction

This paper claims five main issues about the benefits of the use of OSS in Defense environments: Cost, Innovation, Delivery, Security and Agility.

Some of these issues appear pretty straightforward, like the cost issue. In fact the use of OSS decreases (or eliminates) contractor's monopoly regarding the exclusivity of the required capabilities. Furthermore innovation is fostered, since Defense officials have more time to focus on their core business and not to code (which is done by the community). Similarly, also the delivery is improved, because military software engineers can focus on changes and integration of existing software capabilities, instead of having to redevelop entire software systems. Time management is optimized to deliver new capabilities which relate more with the core business. It has not to seem strange that also the security is enhanced through a peer reviewed and reactive community. Furthermore a broad access to the source code enables software security even after the release. Likewise, software development is agile, since there is no copyright infringement (since the OSS is licensed by the Defense agency or the Government) so it is easy for all departments to use clones (piece of codes) to set up their own systems, without any copyright infringement. It is like a public library, where any official can just pick what he needs to build his system.

DevOps is a growing software engineering methodology to assure a better integration of the Development, Quality Assurance and Operations Department, in order to optimize the quality and the number of releases. This is, for instance, crucial in a dynamic environment where features have to change very fast (like a battle environment). Furthermore, optimizing the releases decreases also the Design Debt, which is very typical of Scrum methodologies.

After an explanation of the (ii) background, this papers points out that the OSS community can lead to enhance several benefits for software development (iii) and software optimization through the newest methodologies (iv). In conclusion (v) we suggest further research in reference to the European defense software development system.

2. Background

The debate about the use of Open Source Software (OSS) in military programs is not new in literature [1]. Passively managed closed government programs, developed mainly with a waterfall methodology, are

less flexible and less efficient than open source systems. Thus, the Department of Defense (DoD) is trying to shift through a more open Software Development Approach [2]. Nevertheless there are two main problems that arise: IPRs and National Security. Usually, the government does not have the intellectual rights to make it more open, having, maybe, only purpose rights but not unlimited rights. Furthermore the government wants to maintain a National Security advantage by classify it, in order to not permit to others to see/use it.

The real point behind these issues is the real trade off, i.e. are the benefits greater than the drawbacks? The answer to this question cannot be univocal; it definitely depends from the organization and settings of the software design. For example, we see an excellent respond of using open architectures for modelling and simulation (M&S) at NATO level [3]. Especially for developmental environments, where interoperability plays a major role (e.g. NATO) open architecture seems to be a good solution in order to create a fully respondent and effective environment (i.e. M&S). Clearly we have to distinguish the level of confidentiality and sensibility of every environment, but especially for interoperability open architecture seems to have a good respond. We find in literature many case studies where openly-shared systems address main issue of interoperability [4]. Using Commercial-Off-The-Shelf products (COTS), in defense environment has relevant cost savings effects through economies of scale, this especially for non-critical systems, such Special Test Equipment (STE) for testing military avionics equipment. Therefore, the potential cost savings due to COTS usage is proportionately greater in STE than in the higher volume avionics systems that are tested [5]. A second major benefit of using COTS products is that test system development schedule cycle time is greatly reduced [5]. Even though the realization of flexible and robust systems without a supporting architecture is difficult, requires significant system rework, and precludes the concept of a product line, still the use of COTS seems to be worthy [6]. The use of Agile development system is probably of the most appropriate organization of software development lifecycle and maintenance, which has a relevant impact on the project's budget. Especially in defense environment "sustainability" is a major issue [7]. Successful software sustainment consists of more than modifying and updating source code [8]. It also depends on the experience of the sustainment organization, the skills of the sustainment team, the adaptability of the customer, and the operational domain of the team. Thus, software maintenance as well as operations should be considered part of software sustainment [9]. We know, actually that the majority of defense system lifecycle costs are incurred after the system has entered operations. Operations and sustainment costs can easily reach 60% to 80% of a weapon system's total lifecycle costs, depending upon the type of system and the duration of its employment [10]. DevOps, blurring the lines between software development and operations teams, pushing continuous integration even earlier in a product/system lifecycle, seems a promising potential option for use in IT systems and weapon and logistics support systems [11].

3. Benefits of OTD

As it is known, the Open Technology Development (OTD) at the DoD has become reality and it is used to develop military software. Software developers of the community (i.e. not governmental officials) and governmental developers, develop and manage collaboratively software in a decentralized way [12]. Thus, OTD is grounded on open standards and interfaces, open source software and designs. Furthermore collaborative and distributed online tools and technological agility enhances OTD.

These practices are proven and in use in the commercial world. Likewise non-military environments, the DoD is pushing for open standards and interfaces that allow software to evolve in a complex development environment. Therefore, using, improving, and developing open source software might minimize redundancy in the development and maintenance process, fostering the agile development of software. Not surprisingly the DoD uses open source software also for critical applications, considered as structural part of military infrastructure, especially in four broad areas: (i) Security, (ii) Infrastructure Support, (iii) Software Development, and (iv) Research [2]. Collaborative and distributed online tools are now widely used for software development. The private sector also often strives to avoid being locked into a single vendor or technology and instead tries to keep its technological options open (i.e. using OSS). Removing such OSS tools (e.g., OpenBSD) would mean to harm crucial infrastructure components on which network (i) Security relies on. Furthermore it would also limit DoD access to the use of powerful OSS analysis and detection applications that hostile groups could use to help stage cyberattacks, as the general expertise in it. Finally, the established ability of OSS applications to be updated rapidly in response to new types of cyberattack would be harmed. This is in part because

DoD groups use the same analysis and network intrusion applications that hostile groups could use to stage cyberattacks. The uniquely OSS ability to change infrastructure source code rapidly in response to new modes of cyberattack has been proven to be very effective [2]. Therefore OSS has been proven to be reliable for many DoD's application, also in critical and sensitive ones, like cyberattacks. Interestingly, from an IPR perspective, the GPL (the most common used license in the DoD) turns out to be surprisingly well suited to use in security environments. This is because of the existing and well-defined abilities to protect and control release of confidential information. The established awareness largely removes the risk of premature release of GPL source code by developers but, at the same time, developers make an effective use of the autonomy of decision typical of the GPL license. The (ii) Infrastructure Support depends on OSS, since OSS applications rely on the ability of the DoD to support web and Internet-based applications. (iii) Software Development relies on the OSS community to grab from the large pool of software developers, also with specific skills in different programming languages, directly outgrowths of the Internet. Finally, (iv) Research benefits from OSS's little support costs. The unique ability of OSS to support sharing of research results in the form of executable software is particularly valuable for the DoD.

Like in any commercial - OTD environment, it is the software community that has the proper access to source code and designs documents across the company interacts with the company itself. This creates a decentralized development environment, leveraging existing software assets. Not surprisingly, OTD methodologies that have been used from OSS to open standards architectures have their most successful implementations from the direct interaction with the end-user community. The only way to make an OTD successful is, thus, the merging interest and inputs of both developers and users.

Briefly, we will now highlight the most controversial issues of OSS in security and defense environments.

3.1. Security issues

A recent qualitative analysis showed that among software security professionals, OSS is perceived as a powerful defense tool against attackers [13]. Interestingly, no major issues were emphasized in the close vs. open source comparison of the two software paradigms in the context of vulnerability and risk management. One relevant professional affirmed that it is "...impossible to say which one is more secured and has less vulnerability, look just at Borland [closed source software] example that had a backdoor password for many years. It was discovered only recently" [13]. Among professionals there is the common belief that the time to reach and fix the vulnerability in open source security software is much faster than closed source software, therefore more efficient. Nevertheless it is also worthy to say that in their opinion most of the closed source software is generally better tested and contains less bugs and vulnerabilities. What it is interesting to notice is that having full access to the source allows an independent assessment of the exposure of a system, like any peer review system. Also the risk associated with using the system makes patching bugs easier and more likely and drives software developers to spend more effort on the quality of their code, in order to be not blamed by the community in which they have freely chosen to engage, with an easy software quality assessment [14].

Even if we cannot affirm that OSS is more valuable than close systems, we can, at least state that it has, for many applications, the same dignity.

3.2. Cost issues

Even if OSS is free, this does not mean that is has no cost. We can for sure argue that since it is breaking vendor's monopoly, it lowers lock-in costs. More in detail we can say that there are some cases where the use of OSS is cost beneficial, like in stable slow-growth environments with a large number of software installations the low purchasing and maintenance cost of OSS can result in savings and thereby increased profitability [15]. More in general we can state that adopting OSS lowers cost and increase operating efficiencies. Studies, like Spinellis *et al*. point out that cost optimization is the main reason why large organizations switch from close to open software [15]. Some academics also argue that, since the market will provide companies with a closed system which is most suitable for their needs in terms of flexibility, technological sophistication, or ability to adapt software to their specific needs, the major benefit of OSS is the price [16].

3.3. Innovation issues

Open Source is basically a huge library where everyone could pick what he needs. Pieces of code from different programs can be assembled without having to invent a new system from scratch or break IPRs. Developers can rapidly assemble and modify

existing systems and components, focusing their time and effort writing the code that takes standing capabilities to a higher degree, or combines already existing components into one integrated system. Thus, programmers need to focus on changes and integration of new and critical software capabilities. This form of software reuse is called in literature software cloning [17].

3.3.1. Cloning. There is a relevant debate in literature about the advantages and disadvantages of cloning. In table 1 we figure out the most relevant issues.

Table 1.

Advantages	Disadvantages
Clones are useful if different customers share similar requirements [18].	High maintenance costs [19].
Some programming languages encourage the use of tem plates, which result in software cloning [18].	Propagation of bugs: if a clone contains an error, it will spread rapidly over other parts of the program [20].
The use of clones can respond, sometimes, to efficiency requirements in the development [21].	Cloning discourages the use of refactoring, leading to a bad design of the system [22].
Using clones reduces the time required to develop a program [23].	Using clones increases the size of the code, leading to a less efficient system [24].

Advantages and disadvantages of code cloning

4. Sustainable Software Development

Open source is not a *panacea*. There is and will always be the need for software engineers to code, test, deploy and operate. Open source represent a valuable tool to e.g. overcome single vendor's lock-in and improve network security, among others. In a survey of over 400 business executives conducted by the IBM Institute for Business Value (IBV) it came out that even if software development and delivery are felt as "critical" by software houses, only 25 percent believe their teams leverage development and delivery effectively [25]. IBM called this, "execution gap, which is basically the difference between the need to develop and deliver software effectively and the ability to do so. IBM concludes that this gap is causing missed business opportunities for the vast majority of companies. This image is useful to understand the centrality of a sustainable software development. Organizations that use OSS cannot override such issues. Therefore it is important to understand the main topics of software development in OSS integration.

Software development methodologies appear to be more complex and mixed than just straightforward techniques, as the 8th *Annual Survey on the State of Agile* suggests with 55% of prevalence of Scrum in Agile [26]. Many implementations in execution appear to be hybrids of Agile methods, with some traditional methodologies, such as Waterfall. Such a hybrid is usually called Water-Scrum-Fall, known also as a flexible development approach that includes both waterfall and Agile development principles [27]. The point is that organizations, usually, utilize Scrum software development techniques but employ traditional waterfall methodologies for non-development issues (e.g. planning and budgeting).

What often happens is a poor software design, caused by different factors, like business pressure, lack of deep system understanding by both developers and business, lack of documentation or collaboration. The technical (or design) debt can also be described as the gap between the current state of a software system and an idealized state in which the system is perfectly successful in his environment [28].

Sustainable software development methodologies can narrow this gap. This is possible through a tight relationship and cooperation between customers and producer but also between the development and the operation department of a company. Agile methodologies give some important answers to the technical debt issue as also DevOps. Implicitly a DevOps approach applies agile and lean thinking principles to all stakeholders in an organization who develop and operate. It balances development and operation concepts with the aim of changing cultural mindsets and leveraging technology more efficiently [29].

5. Conclusions

In this paper we pointed out some major issues of OSS integration within a Defense Environment. We also highlighted that any integration cannot disregard from sustainable software development.

Future research could take into consideration some case studies within European Armies of OSS integration. Even if we have some research about the DoD, studying OSS implementation outside the US could interesting to confront if the US represent a specialty or they are in line with other Defense Organizations.

Acknowledgments

I would like to thank Prof. Paolo Ciancarini for his helpful remarks and continuous inspiration.

References

[1] Scott, J., Wheeler, D. A., Lucas, M. and Herz, J.C., "Software is a Renewable Military Resource", *DACS*, 14(1) 2011, pp. 4-7.

[2] MITRE Corporation, *Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense*, 2003.

[3] Hassaine, F., Abdellaoui, N., Yavas, A., Hubbard, P., Vallerand, A.L., "Effectiveness of JSAF as an Open Architecture, Open Source Synthetic Environment in Defence Experimentation". In *Transforming Training and Experimentation through Modelling and Simulation*, 11, 2006, pp. 11-1 – 11-6.

[4] Sawilla, R.E., Wiemer, D.J., "Automated computer network defence technology demonstration project (ARMOUR TDP): Concept of operations, architecture, and integration framework", *Technologies for Homeland Security (HST), 2011 IEEE International Conference on*, 2011, pp. 167 – 172.

[5] Pizzica, S., "Open Systems Architecture solutions for military avionics testing", *IEEE Aerospace And Electronic Systems Magazine*, 2001, 16(8), pp.4-9.

[6] Henry, M. Vachula, G., Prince, G.B., Pehowich, J., Rittenbach, T., Satake, H., Hegedus, J., "A comparison of open architecture standards for the development of complex military systems: GRA, FACE, SCA NeXT (4.0)", *Proceedings - IEEE Military Communications Conference MILCOM*, 2012, pp. 1-9.

[7] Defense Acquisition University, *Sustainment Archived References*, 2013: https://acc.dau.mil/CommunityBrowser.aspx?id=677043. Accessed on 10.04.2015.

[8] Lapham, M. A., Woody, C, "Sustaining Software-Intensive Systems (CMU/SEI-2006-TN-007)", *Software Engineering Institute, Carnegie Mellon University*, 2006: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7865. Accessed on 10.04.2015.

[9] Defense Acquisition University. *Integrated Product Element Guidebook.* DAU, November 2011: https://acc.dau.mil/CommunityBrowser.aspx?id=496319. Accessed on 10.04.2015.

[10] Taylor, M., Murphy, J. "Colt. OK, We Bought This Thing, but Can We Afford to Operate and Sustain It?" *Defense AT&L: Product Support Issue*, March–April 2012, pp. 17-21.

[11] Regan, C., Lapham, M. A., Wrubel, E., Beck, S., Bandor, M., "Agile Methods in Air Force Sustainment: Status and Outlook (CMU/SEI-2014-TN-009)", *Software Engineering Institute, Carnegie Mellon University*, 2014.

[12] Herz, J.C., Lucas, M. and Scott, J., "Open Technology Development Roadmap Plan", April 2006, *Office of the Under Secretary of Defense for Acquisition*.

[13] Silic, M., "Dual-use open source security software in organizations e Dilemma: Help or hinder?", *Computers & Security*, 39, 2013, pp. 386-395.

[14] Hoepman, J. H., Jacobs, B., "Increased Security Through Open Source", *Communications of the ACM*, 50 (1), 2007, pp. 79-83.

[15] Spinellis, D., Giannikas, V., "Organizational adoption of open source software", *The Journal of Systems and Software*, 85, 2012, pp. 666– 682.

[16] Attewell, P., "Technology diffusion and organizational learning: the case of business computing", *Organization Science*, 1997, 3, pp. 1–19.

[17] Rattan, D., Bathia, R. Singh, M. "Software clone detection: A systematic review", *Information and Software Technology*. 55, 2013, pp. 1165-1199.

[18] Kim, M., Bergman, L., Lau, T., Notkin, D., "An Ethnographic study of copy and paste programming practices in OOPL" *Proceedings of 3rd International ACM-IEEE Symposium on Empirical Software Engineering (ISESE'04)*, 2004, Redondo Beach, CA, USA, pp. 83-92.

[19] Monden, A., Nakae, D., Kamiya, T., Sato, S., Matsumoto, K., "Software quality analysis by code clones in industrial legacy software", *Proceedings of 8th IEEE International Symposium on Software Metrics (MET-RICS02)*, 2002, Ottawa, Canada, pp. 87-94.

[20] Johnson, J.H., "Substring matching for clone detection and change tracking", *Proceedings of the 10th International Conference on Software Maintenance*, 1994, Victoria, British Columbia, Canada, pp. 120-126.

[21] Kapser, C.J., Godfrey M.W., "Cloning considered harmful considered harmful: patterns of cloning in software", *Empirical Software Engineering*, 13, (6), 2008, pp. 645-692.

[22] Lavoie, T., Eilers-Smith, M., Merlo, E., "Challenging cloning related problems with GPU-based algorithms" *Proceedings of 4th International Workshop on Software Clones*,2010, Cape Town, SA, 25-32.

[23] Kapser, C.J., Godfrey M.W., "Supporting the analysis of clones in software systems: a case study", *Journal of Software Maintenance and Evolution: Research and Practice*,18, (2), 2006, pp. 61-82.

[24] Koschke, R., "Frontiers of software clone management", *Proceedings of Frontiers of Software Maintenance (FoSM08)*,2008, Beijing, China, pp. 119-128.

[25] IBM, "The software edge: How effective software development and delivery drives competitive advantage", *IBM Institute for Business Value*, 2013: http://www-935.ibm.com/services/us/gbs/thoughtleadership/softwareedge . Accesed on 18.04.2015.

[26] VersionOne, "8th Annual State of Agile Survey", *VersionOne, Inc.*, 2014: http://stateofagile.versionone.com/. Accessed on 15.01.2015.

[27] West, D., Gilpin, M., Grant, T., Anderson, A., "Water-Scrum-Fall is the Reality of Agile for Most Organizations Today: Manage the Water-Scrum and Scrum-Fall Boundaries to Increase Agility", 2011, *Forrester Research*.

[28] Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., Maccormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N., "Managing technical debt in software-reliant systems", *FSE/SDP Workshop on the Future of Software Engineering Research, FoSER* 2010, pp.47–51.

[29] Swartout, P., "Continuous Delivery and DevOps: A Quick Start Guide", *Packt Publishing*, 2012.