




Managing the Operational and Security Exposure of Free and Open Source Software

Black Duck Software White Paper



The fastest growing, most agile software companies are built on free and open source software (FOSS): consider Facebook with its 500 million users, Twitter, Amazon, Android, YouTube, Google and more. Companies that use free and open source software for internal use or companies which use FOSS components in the creation of customer-facing goods and services understand the benefits of innovation, and use FOSS as a way to balance risk and innovation while managing cost. Benefits are clear: with the industry-standard cost per line of code (LoC) at \$10 - \$20 and with an average Fortune 1000 production/re-use of 50,000 LoC per component, the use of multi-source development practices with FOSS can save an organization in the starting range of \$500K to \$1M per project.

FOSS is pervasive in today's software lifecycle and supply chain. Gartner estimates that by 2013 open source technology will be included in 85 percent of all commercial software packages, and by 2016 will be included in mission-critical software packages within 99 percent of global enterprises¹. It's clear that free and open source is about opportunity, and yes, about risk, complexity, and management challenges as well.

In our two-part series of white papers Black Duck examines key considerations and risk factors in the context of free and open source software use. In this first paper, *Managing the Operational and Security Exposure of Free and Open Source Software*, we address the operational and security risks every development manager must take into account when creating software in a multi-source development environment, how they are often underestimated in the use of FOSS, and how they can be addressed with best practices and tools. In the second paper, *"IP Risks and Myths: The Real Exposure Issues with Free and Open Source Software"* we look at the intellectual property (IP) and legal exposure that may be encountered when FOSS software makes its way into the development lifecycle, whether for internal use or to be distributed in products.

We welcome your thoughts and comments.

Introduction

FOSS today: mainstream and with a positive ROI as long as operational issues are managed

Enterprise IT organizations spend tens to hundreds of millions of dollars each year on software development. These organizations, under pressure to contain or cut costs while delivering new business and service innovation in less time, increasingly turn to the use of free and open source software as a solution. The trend is so established that Gartner analyst Mark Driver has declared the use of open source mainstream:

"As OSS has matured, it has expanded its role within mainstream IT organizations to take on increasingly complex and mission-critical challenges. In 2010, Gartner estimates that open source was included in at least 75% of Global 2000 enterprises."²

As FOSS use proliferates, companies need to evaluate FOSS offerings as competitive alternatives to home-grown and third-party-software and make selections based on technical excellence, including quality, reliability and TCO merits. Despite the blessing of top-tier analysts, however, the mention of 'open source' can still confound and concern executives with financial and operational responsibilities. Operational and security issues must be considered, and a strategy, policies and processes put in place to make FOSS's use successful. Among the most pressing operational and security issues is the potential for technical failure, which can be compounded by lack of transparency about where code was sourced.

A day in the life of a development manager

A global Fortune 100 company developing enterprise software with a cadre of 5,000 globally dispersed developers, outsourced development teams, and development teams from recently acquired firms faced a challenge - how to integrate and manage FOSS components brought into the development stream by individual developers. The components weren't

¹ Driver, Mark What Every IT Practitioner Needs to Know About OSS; ©2010 Gartner, Inc. & its Affiliates.

² Various, Gartner Predicts 2011: Open-Source Software, the Power Behind the Throne. ©2010 Gartner, Inc. and/or its affiliates.

captured in an enterprise software 'catalog.' There was no list of licenses and dependencies, versions, patches, or point of origin. Documentation was spotty. In many cases it wasn't apparent if the FOSS had actually been implemented in the company's products, or who was maintaining code updates on what had been integrated.

The company set out to institute manual processes to track code sources and identify free and open source within its development environment, but time and staffing needed to manage the job slowed development. The manual processes also proved prone to human error.

Then the company went on an acquisition spree to expand its product portfolio. After completing several deals it discovered some acquired software containing open source components had escaped its manual processes. Export issues caused by the use of code containing encryption algorithms raised costs and caused release delays due to the need to remediate and to file documentation with government agencies.

The company was preparing to ship a major upgrade to a core product and running final pre-ship tests when a component had a failure. Tracking the failure back, the development team was able to isolate the problem to a subsystem built from home-grown and FOSS components. The FOSS code hadn't been brought up to its latest version, creating

incompatibilities with the core product's libraries and drivers. Total delay in shipping the finished product: six weeks. Total cost? Damage to their brand? Don't ask. Clearly it was time to find a better solution. The company began searching for an automated tool to manage, control and catalog FOSS components in its enterprise products.

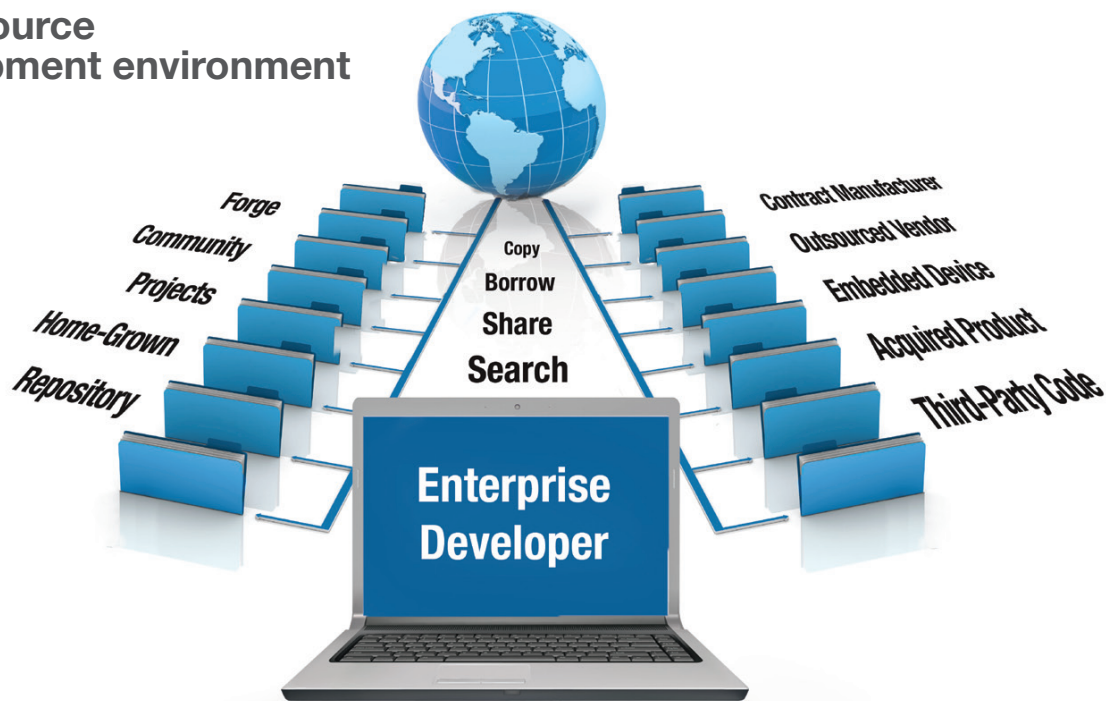
In the following pages, we will examine technical failure and security issues that development managers and export control managers face when creating products from multi-source code using FOSS.

Operational exposure via technical failure

In a multi-source development environment, or an IT environment in which code of uncertain provenance is used, technical and security failures such as that described above are a threat to operations. These failures can stem from a number of sources:

- Not knowing what's in your code
- Code quality and reliability
- Downtime
- Incompatibilities and version proliferation
- Unpredictable performance and availability

Multi-source development environment





What's in your code?

Development teams know they can accelerate the development process and speed innovation using FOSS and third party code in their products. However, the use of FOSS introduces new challenges not present with commercially-sourced code. A number of products are on the market to help companies proactively scan codebases to identify unknown code, code origin and license information, as well as automatically check compliance and identify security information. The best tools support both the front end of the development process, where developers acquire components, and monitor components in use, as well as the back end of the process when code needs to be validated before it is deployed.

Quality and Reliability

Software acquired from ISVs is typically bound by service level agreements (SLAs). While not a guarantee of the quality and reliability of software and components, SLAs are the operational glue that holds many enterprise IT organizations together. With FOSS, however, SLAs need to be enabled with process and change control internally, oftentimes with the help of third-party support organizations.

Helping to ensure code quality and reliability, many top FOSS projects have vibrant developer communities or corporate hosts (e.g., Android & Google) which provide the “many eyes” paradigm that can support superior quality and reliability, which combined with advice, support and patches from community members, helps to mitigate the possibility of service interruptions and failures.

Technical failures - downtime

Downtime resulting from poor software quality and reliability is costly and needs to be managed. When working with FOSS, a first step to avoiding this problem is knowing where the code came from, the community/developers behind it, and which version is in use. While your developers may know how to integrate FOSS code into your development stream, they may not have the technical background to find and fix root cause problems that lead to failure and downtime. Here, as in all things FOSS, knowing what's in your code - and managing it throughout the application development lifecycle - is key.

Managing FOSS to minimize technical failures

Clearly, with SLAs hard to come by and downtime caused by technical failure a reality, managing FOSS is critical to success. Enterprises must manage the acquisition and introduction of FOSS with an effective governance process to stay ahead of issues and ensure ROI.

FOSS makes its way into development streams in a number of ways. It can be acquired by internal developers, who most likely use search engines, forges and communities to find and assess the best code to solve a development challenge from hundreds of thousands of projects on the Net. It may be acquired as code embedded in software from outsourced development partners, or embedded in acquired hardware. It may come to a company during a merger or acquisition. While you may have a practice of acquiring FOSS code only from trusted sources, there's still a chance that you haven't identified all FOSS in your environment.

Establishing an acquisition process and policy is the first line of defense against technical and security vulnerabilities. A FOSS acquisition process has the added benefit of creating a control point for managing legal risk - a topic which requires its own discussion.

Incompatibilities

Incompatibilities often emerge from two directions: using code from incompatible source code distributions, and version proliferation. Android has both issues: it is a fork from the Linux kernel, with hundreds of components in a constant state of flux. Thousands of developers working for handset and tablet manufacturers, application developers, and systems integrators modify the code daily on multiple source trees, creating opportunities for technical/operational incompatibility.

Unpredictable performance and availability

Not all code is created equal. In the FOSS world this reality has additional impact: with the exception of major distributions with large communities and commercially-hosted projects, there may be no individual or organization committed to ensuring adequate performance, uptime and availability. It may be unclear who last modified a component, documentation may be sparse and development staff may be unfamiliar with the fundamentals of a given component. In the Android example cited earlier, the multitude of components, code trees and the sheer size of the developer community contribute to varying levels of performance and availability in different Android distributions.

In all cases cited earlier, an organization's operational exposure from the unmanaged use of FOSS is clear. Managing FOSS use requires a comprehensive governance program.

Operational exposure through security issues

Code defects exist in every piece of software, and some affect security. The industry average is one defect per 1,000 lines of code (LoC). Android, an open source project, is purported to have .47 defects per 1,000 lines of code, making it arguably more secure than closed-source code.³ Great news, and an argument for the quality of FOSS, but diligence is warranted.

Security is made more complex by mixing FOSS with other code. In addition to the obvious risk of not managing integration well, it's the development organization's responsibility to track security vulnerabilities and establish a process for remediating security issues. (Many FOSS infrastructure platforms have a security process, as do foundations such as the Apache Foundation, e.g., see <http://www.apache.org/security/committers.html>.)

Fortunately, with the right automation tools, the tracking of security vulnerabilities in FOSS components, and the process of alerting and tracking where affected components are used can be automated saving significant time, money and minimizing risks.

Governance

Enterprises should plan to establish a governance program that encompasses all third-party code, whether from a commercial vendor, a FOSS project, or an outsourced supplier. FOSS is different - many hands do make a great FOSS project - but this richness adds complexity. Build in governance to avoid technical exposure while benefiting from the richness and abundance. An effective governance program has four main elements: strategy, policy, process and technology.

- Business strategy - spells out the objectives for use of FOSS.
- Policy - sets the rules for evaluating, approving, using and releasing FOSS code and participating in communities.
- Process – the way policy is reliably realized on a day-to-day basis.
- Technology – enables automation of the governance program to ensure compliance. Allows development organizations to “design in” and minimize overhead.

Among ongoing considerations in managing the use of FOSS to avoid technical issues, recognize that your development organization bears the full burden of support, documentation and maintenance if code

was acquired from little-known or inactive projects - after all, there are hundreds of thousands of projects out there, and not all are active or supported by a community. Create a process and assign development resources to continually track for posts and updates to catch bugs and security vulnerabilities before they become technical issues.

“Given the ubiquity of OSS in the typical IT enterprise, it becomes paramount to establish a firm set of corporate rules that allow IT organizations to leverage the benefits of OSS (innovation, cost, and others), while minimizing the associated risks.”⁴

A clearly articulated FOSS strategy combined with policy, enhanced processes and automation technology (which includes a tracking repository to record security vulnerabilities, manage license obligations, version control issues and compliance with export regulations) make for world-class governance. With proper governance, the introduction of an external component into a project can be managed by an integrated set of business processes that puts software developers in partnership with purchasing, IT, security, legal, product management, and any other internal groups that have a stake in the policy administration of software development.

Solution? Discover and Manage FOSS

A prudent first step towards managing FOSS is to understand how much you have and where it's used. Many development organizations, large and small, have never had policies or guidelines about FOSS use, which didn't prevent developers from using it. Quite the contrary. The lack of cost and the freedom to acquire software without going through a procurement process has been one of the attractions. Fortunately with technology it's possible to automatically scan and audit a code base to discover what's in it. Audits aren't a one-time thing for a development shop that uses FOSS, though; it's wise to audit code on an on-going basis to ensure compliance with the company's risk management policies and tolerance for technical issues. Once a preliminary audit has identified FOSS-in-place, consider four processes to keep things under control:

- **Code acquisition and cataloguing process**
 - Captures all relevant attributes of FOSS components
 - Assesses compliance with company and development policies
 - Assesses any other risks, e.g. security vulnerabilities
 - Captures results as a digital 'fingerprint' that enables easy identification of FOSS components in subsequent scans.

³ http://www.theregister.co.uk/2010/11/02/android_security/

⁴Mark Driver, Lead OSS Analyst, Gartner Group, November 2010.

- **Code update process**
 - Checks for code origin, quality, and new IP introduction
- **Release process**
 - Verifies all FOSS in use
 - Audits the code for license, export and security issues
 - Remediates issues that arise
- **Compliance process**
 - Document the steps necessary for license compliance
 - Prepare supply chain compliance kits.

Why operational controls matter: the ROI discussion

Given the potential for cost avoidance using FOSS in a mixed-source environment, governance of FOSS use has direct implications for return on investment throughout the application development lifecycle. As Gartner's Mark Driver has observed,

“The lack of effective governance programs has already led to a negative ROI impact in at least 10% of OSS adopters in 2010, but the growing presence and use of OSS technology in increasingly mission-critical deployments will drive negative ROI to a 50% outcome among adopters by 2015.”⁵

ROI depends on programmatic governance and management of FOSS use. Additionally, automation of governance processes improves ROI. Consider the key cost drivers of manual code management:

- Amount of code
- Number of open source components used.

Cost of Manual Methods

Black Duck Software research has shown that it costs approximately \$450/mb to manually analyze code for policy violations (assuming the code is 20% source and 80% binary). Assuming the average component contains 350,000 lines of code, reviewing regular changes and updates can cost \$6,600 for manual code scan and analysis. The cost of manually validating code is the largest expense in a manual code review process. Add to this the need to manage components - which can run \$1,200 per year - and the \$7,800 price tag per component,

per year becomes a significant cost. If an organization maintains a library of 200 FOSS components, the cost rises to \$1.5 million a year for manual code management. The impact on ROI, thus, is significant. The good news is that with proper automation technology the cost can be as little as 1/10 the cost of a manual process.⁶

Open source: abundance, complexity, innovation and responsibility

Using FOSS in a software development project or product can have many benefits when managed properly - faster time to solution, lower cost, potentially lower redistribution costs and lower code maintenance costs are a few. Yet FOSS also introduces complexity. The richness and abundance of FOSS - many hundreds of thousands of projects are tracked in the Black Duck KnowledgeBase - can bring complexity and operational challenges such as managing code and version consistency across projects and product lines, managing support for external code elements, and managing security issues.

As FOSS use increases in Enterprise IT, benefits and challenges must be balanced by management and governance. The flexibility, innovation and cost advantages of open source can be managed to extract maximum ROI from development efforts. While many IT organizations may be unprepared for the challenges of managing and monitoring the use of open source code, FOSS use can be governed with relative ease, much as any third-party software is managed.

But FOSS should be treated differently in a few critical areas:

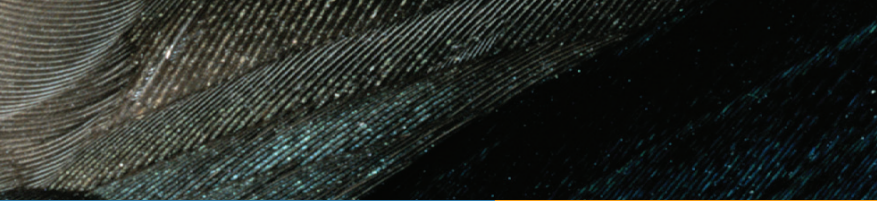
- FOSS comes “as is” with a disclaimer, not a warrantee
- Projects may not have a structured approach to QA
- It is the downloader's responsibility to keep track of patches and releases
- SLA support may not be available from a project, but it may be available from independent providers
- There is no legal or IP indemnification (as is customary with commercial software.)

With FOSS, as with all software, planning and execution are vital to avoid operational, technical and security failures. Enterprise IT users of open source must:

- Know what open source software is being used and its essential attributes
- Know where each open source component is being used and that its use is appropriate
- Know who is responsible for the maintenance of each component
- Know that compliance with open source license obligations is met.

⁵ Mark Driver, *ibid.*

⁶ Black Duck Software whitepaper, *The Business Case for Automating Open Source Code Management*





About Black Duck Software

Black Duck Software is the leading provider of products and services for automating the management, governance and secure use of free and open source software, at enterprise scale, in a multi-source development process. Black Duck® enables companies to shorten time-to-solution and reduce development costs while mitigating the management, compliance and security challenges associated with free and open source software. Black Duck Software powers Koders.com, the industry's leading code search engine for open source, Ohloh.net, the largest community for and free public directory of open source, and The Olliance Group, the leading open source business and strategy consulting firm. Among the 500 largest software companies in the world, according to Softwagemag, the company is headquartered near Boston and has offices in San Mateo, California, London, Paris, Frankfurt, Hong Kong, Tokyo and Beijing. For more information, visit www.blackducksoftware.com.

© 2011 Black Duck®, Know Your Code®, Ohloh®, SpikeSource®, Spike®, and the Black Duck logo are registered trademarks of Black Duck Software, Inc. in the United States and/or other jurisdictions. Koders™ is a trademark of Black Duck Software, Inc. All other trademarks are the property of their respective holders.



Contact

To learn more, please contact:
sales@blackducksoftware.com or
call +1 781.810.5100

Additional information is available
at Black Duck's web site:
www.blackducksoftware.com