GAMP® CONSIDERATIONS

When Relying on Open-Source Software

By James Canterbury and Petch Ashida Druar

This article aims to refresh information on open-source software (OSS) within regulated computerized systems that was first discussed in an article in May-June 2010 *Pharmaceutical Engineering*®. The adoption of OSS advanced since then, and the article explores the importance of recognizing when an organization is relying on OSS and the benefits and risks this brings from a GAMP® 5 perspective.

Reliance on OSS has become prolific across today's information technology (IT) environments. Whether it is the use of well-known operating platforms like Linux or statistical analysis tools such as R or leveraging available JavaScript libraries to build custom applications, OSS has permeated most enterprises, including pharmaceutical/biopharmaceutical companies. When relying on OSS within a regulated computerized system, it is important to understand the method in which that software is developed and maintained so that critical thinking can be applied when determining the level of risk and mitigation strategies.

In the May-June 2010 issue of *Pharmaceutical Engineering*®, the article "Guide for Using Open Source Software (OSS) in Regulated Industries Based on GAMP" detailed the various support models for maintaining a GxP environment where OSS is used [1]. OSS is sometimes referred to as free/libre/open-source software (FLOSS) or free and open source software (FOSS), which attempts to distinguish between the values behind developing OSS and the licensing models for distributing it [2]. While important to understand, the primary concern from a GxP perspective is the development and maintenance of this software, and we will simply refer to it as OSS in this article.

This article aims to refresh *Pharmaceutical Engineering*® readers on the topic and build upon the foundation set in the 2010 article by highlighting several areas that have advanced since the publication of that article. Specifically, we will cover the importance of recognizing when an organization is relying on OSS and the benefits and risks this brings from a GAMP Category 5 perspective (see Figure 1). The large majority of OSS today would be classified as GAMP Category 1 software (i.e., embedded software components, libraries, development tools, and operating systems).

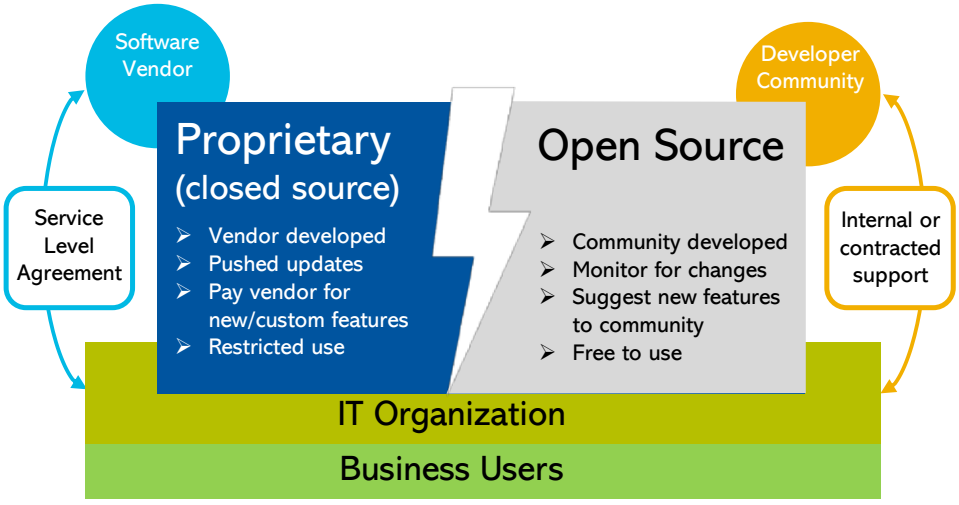
Like other infrastructure components, the inherent level of GxP risk is low; however, with increasingly connected systems and the rise in cybersecurity attacks (which often exploit vulnerabilities in GAMP Category 1 software to gain unauthorized access to networks and system resources), it is increasingly important for the GxP practitioner to have a solid understanding of what they are relying on and to plan their risk-based validation approach accordingly.

When we look toward the future, there is a strong trend for smaller fit-for-purpose applications that often run on broader, decentralized networks. In a GxP environment, these specialized systems could be GAMP Category 4 or 5 software and would carry a higher risk. Examples range from applications for managing clinical trials to post-market surveillance. These types of applications rely extensively on OSS, especially if they run on public networks.

OSS CHANGES

While a lot has recently changed in IT, the principles of GAMP set forth in the 2010 article still hold true for most companies that leverage OSS. However, there have been significant developments in the way communities organize to develop and maintain OSS. It is this collaborative development process and the freedom for anyone to access the source code to study, use, or modify it as they see fit that we must consider when using it to meet regulatory requirements.

Figure 1: Comparison of closed- vs. open-source software.



One driver for the increased adoption of OSS is its availability and reusability: developers find it easier and faster to build from a component they already know works. A software package is a collection of components that developers pull together to deliver the functionality that users need. By referencing predeveloped components, developers can develop faster and be more innovative.

For example, a few years ago, if you were building an in-house application using JavaScript and your users wanted the ability to left justify their comments in a text box (i.e., align them with the left margin), you would probably use the then-popular "left-pad" package available from the package manager company NPM (www.npmjs.com/package/left-pad) by simply including "$ npm install left-pad" in your build. Now your home-built, possibly proprietary, software is reliant on an open-source package. (Note: As of this writing, left-pad has been deprecated, but is still a relevant example).

In 2020, the Synopsys Cybersecurity Research Center (CyRC) published their annual Open Source Security and Risk Analysis report (OSSRA) [3] and found that of the 1,253 applications audited, 99% contained open-source components. In fact, as pointed out in a 2019 TechCrunch article [4], it is actually software developers, employed by companies, who often discover and integrate OSS components into their current projects. The article states,

> *Once 'infected' by open-source software, these projects work their way through the development cycles of organizations from design, to prototyping, to development, to integration and testing, to staging, and finally to production. By the time the open-source software gets to production, it is rarely, if ever, displaced.*

These references to components are often multiple layers deep, i.e., where one component refers to a library that is made up of other components that refer to libraries.

It is similar to the old anecdote of infinite regress where it was postulated that our world rested on the back of a giant turtle. When challenged to describe what the turtle stood on, the answer was an even larger turtle, with the ultimate conclusion that it was turtles all the way down. With open-source components and reference libraries, it is likewise "turtles all the way down" [5].

Software companies, realizing that this is inevitable, have begun to embrace the use of OSS. A review of the "commits" (the term used when an update to code is posted) between 2011 and 2020 shows that just behind software companies dedicated to open-source development (such as RedHat and Liferay), are familiar names such as Google and Microsoft [6]. These same corporate entities often provide the grants that support the foundations that manage the code base of large open-source projects. Even SAP, considered a highly proprietary software, has an "open source program office" as part of the Linux Foundation [7].

It is no longer a question of if your organization uses OSS; it is a question of "do you understand where it is being used?" The level of oversight and control over these software components have typically been low and should be given closer examination, especially by regulated companies.

OSS allows developers to innovate faster and deliver software with features that capitalize on the collective thinking and experience of hundreds of thousands of developers worldwide. This generally leads to more secure software, more frequent updates, and enhanced modernizations, but to reap these benefits, you need to keep it up to date.

Cultural movements aside, it is undeniable that OSS has become more prevalent, and it extends far beyond installing a Linux operating system on your server or using the Libre Office Suite because you are looking for some free software. In fact, while companies will still cite cost as a driver for choosing OSS, many are realizing that this is not the primary factor; and, as the 2010 article pointed out, "free" software is rarely free.

The decision to use OSS is not always just about cost; it can also be strategic. Because OSS does not come from a proprietary software provider, many companies select OSS so they have the option to switch to different software when needed. A 2020 survey by Tidelift showed 40% of respondents stated "avoiding vendor lock-in" as a primary driver for choosing OSS [8].

EMERGING TECHNOLOGY AND THE ROLE OF OSS GOVERNANCE

One emerging draw to OSS is the awakening of distributed and decentralized systems that operate to form a shared network under a common set of rules. These systems are commonly known as blockchains, but blockchain is only one form of this rapidly evolving class of technology.

The heart of these shared networks are their protocols, or the core code that dictates the rules by which the network functions. A public blockchain is owned by all the members who participate in it, and it is open for anyone to join; therefore, the protocol is necessarily open source. This is not a new concept; we have been living with open-source protocols for years, but they have become entrenched in our everyday lives, even if we do not realize it.

For example, if you are reading this article online, you are leveraging the TCP/IP protocol to make sure your request to view this article in your browser made it to the right computer. The difference is that TCP/IP was created in 1973 long before there was a large internet user base, and it became established as the de facto protocol for transmission as the internet we know today was being built. Changes to TCP/IP are today governed by the Internet Engineering Task Force, a nonprofit, but arguably centralized, authority for the protocol. In a public blockchain, anyone (you do not even have to be a participant) can propose changes, and if the majority of the participants accept the change, it becomes part of the code base. This has drastic implications on how we think about system governance.

While public blockchains may take open-source governance to the extreme, most emerging technologies make heavy use of this development method, even if they later lock down their algorithms in proprietary software. For example, some of the most robust frameworks and tool sets for machine learning (ML) algorithms, which often lead to artificial intelligence (AI) applications, are built using open-source tool kits such as TensorFlow (the open-source deep learning libraries supported by Google) and the Scikit-learn library of classification, regression, and dimensionality reduction algorithms [9].

Some of these same libraries are leveraged in more established software such as R, which is a free software environment for statistical computing and graphics [10]. Even if a company is not using the R runtime environment, they are likely using it as a plug-in in their statistical reporting or visualization applications (which may be proprietary). R is governed by The R Foundation, a group of volunteers who help decide which features are needed and how to fix any bugs that are reported by the user community, which is very broad. In 2021, there were three significant version releases, each addressing multiple issues and adding/changing features, some of which your organization may rely on for making important statistical-based decisions.

In the preceding examples, there is a mix of governance models. One is used for distributed software, such as blockchain, where you may be leveraging a network in which you cannot control the changes. And another is used for locally installed applications (such as R) where you may not know that your implementation has become outdated. And in between, you have ML algorithms, where the program itself may determine when it is best to update.

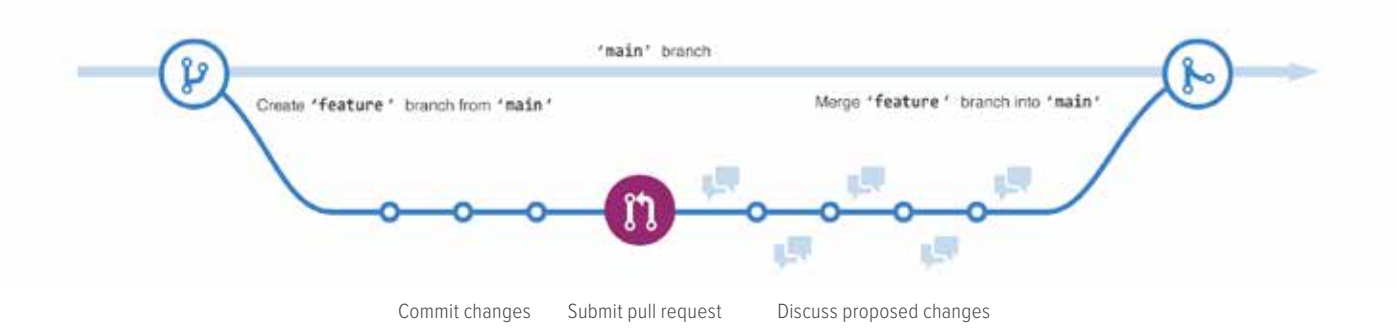
A BRIEF GLIMPSE INTO THE OSS MINDSET

It is often easiest to think of processes in an analog context. In a post on Opensource.com, Bryan Behrenshausen of Red Hat described OSS like baking a loaf of bread and sharing it with a friend [11]. But instead of just giving them the bread, you give them the recipe as well. This way, if they want to check the ingredients, they can see exactly what went into the bread, and if they noticed something did not taste quite right, they could let you know or even suggest how to fix the recipe. Or better yet, if they wanted to modify the recipe to suit their own taste, they are welcome to do so and can even share their version with others. Open source lets you blur the line between chefs, who create something new, and cooks, who follow instructions, by letting the cook talk directly to, or even become, the chef.

This communication between everyone is what fuels the "open-source community" (see Figure 2). Online collaboration tools have merged with social media to create a very responsive and adaptive approach to software development. A great example of this is GitHub, which is the most popular code-hosting platform in the world. It works by allowing a developer to create a new repository (or "repo") for a project they are working on. The repo can contain anything: folders, files, images, datasets. But most important, it should contain a README file that explains what the project is about.

This initial creation becomes the main branch of their project and is considered the definitive branch, or the source code of the project. If you, or anyone else for that matter, want to make a change, you create a branch off the main branch by creating a copy of it at that point in time. You then make your edits to the copied branch and commit your changes. If you think your changes are worthy of being incorporated into the main branch, you open a "pull request" for someone to review and pull your contribution into the main branch. This is where the collaboration begins; as soon as you submit a pull request, a side-by-side comparison of your code is created against the main branch and a

Figure 2: Visualization of collaborative development process [12].



discussion is started. Here, various developers who are interested in your project will comment on your updates (depending on the size of the community, this can take some time). If a consensus is reached that the changes should be accepted, someone with access rights to the main branch can merge the pull request to change the main branch with your updates. At this point, you can delete your branch because the main has now been updated (don't worry: GitHub keeps an extensive history of all branches, pulls, and merges). You can even require a certain number of reviews from other developers prior to allowing a branch to be merged.

Once the merge request has been completed, a new version of the main branch is available for anyone to download and use. If your project has been widely distributed, an announcement is generally made about the new version available. Occasionally a consensus cannot be reached about whether an update should be merged not. In this case a "fork" of the main branch lives on as a separate version. From an end-user standpoint, this is important because you may have to decide if you want to stay with the main branch or go with the fork. If over time, one becomes more popular than the other, or if you simply do not apply updates as new pull requests are merged, you may wind up with outdated and unsupported code. Worse yet, if you do not apply updates that addressed security weaknesses, you will be left with vulnerabilities in your applications.

What is fascinating about this process is that anyone—literally anyone with access to the internet—can view an open-source project and request changes to it. You do not have to be a developer, or even understand the source code: you can follow the community and suggest features and use cases that you think would be particularly good to include. If enough people agree with you, your request can be picked up by a developer and included in the next pull request.

This results in a strange form of user requirements, especially if you are a GxP practitioner used to seeing formal user requirements or design specifications. In an open-source community, these requirements may be captured in snippets of online dialogs or in README documents. As the previously referenced TechCrunch article puts it,

> *The community also ends up effectively being the 'product manager' for these projects. It asks for enhancements and improvements; it points out the shortcomings of the software. The feature requests are not in a product requirements document, but on GitHub, comments threads, and Hacker News. And, if an open-source project diligently responds to the community, it will shape itself to the features and capabilities that developers want [4].*

Most source code updates, especially those that are considered "components," such as the left-pad example, are handled with package managers, which let the developers bundle up their source code and push it out to anyone who is using it. Generally, developers consider it best practice to regularly install all updates before working in their environment so that they can make sure they have the latest version. Because features are added and updates are made frequently, this normally works well...until it does not. There is always the risk that the component you have been relying on might suddenly not be available. This can cause a developer's new build to fail and create disruptions while you scramble to find a replacement component.

Take the left-pad example we have been using throughout this article. In 2016, the developer who wrote and supported that code was not happy with the decisions made by management at NPM, Inc., the company that maintains the npm registry. In a fit of rage, he deleted all of his projects on NPM, including left-pad—which, according to an article on Ars Technica, "ended with JavaScript developers around the world crying out in frustration as hundreds of projects suddenly stopped working—their code failing because of broken dependencies on modules" [13].

In true open-source fashion, the community was able to rally around this and replace the repo with comparable code in about 2 hours and the software builds were able to continue. But the point is that dependencies on that one piece of code had become prolific and, in this case, a single developer was able to affect hundreds of projects with one action. It should be noted that this example is extremely rare and most OSS today repositories have redundancy built in to avoid this.

IMPACT ON PHARMA

Other than being an interesting glimpse into the world of open=source development, why does this matter for a pharmaceutical/biopharmaceutical companies? In a GxP environment, we rely on software day in and day out to perform as designed. It is always best practice to keep your company's code base running on the latest release (not the beta version, but the latest stable release). This helps ensure that any security flaws have been addressed and keeps your software compatible with future releases.

However, this comes at the risk of the code suddenly not operating as it used to (because open source can change) or it could lead to disruptions if the components being updated are no longer supported. Just like with any patch management, a good amount of due diligence needs to be taken when applying updates. But unlike commercial software, there is not always a vendor (or even documentation) to walk you through each update.

As the 2010 *Pharmaceutical Engineering*® article implied, either your IT becomes part of the open-source community, contributing to future releases, reporting bugs, and understanding the updates at a granular level, or you hire a third party to do this on your behalf. Whichever path is taken, the pharmaceutical manufacturer is responsible for maintaining the compliant and validated state of their GxP computer systems. And so GAMP plays an important role not only in the initial verification of software, but also in the ongoing verification of the environment as it is patched and updated. In the case of leveraging software as a service or vendor-hosted applications, it is important to understand their software development life cycle (SDLC) process for keeping up with the latest releases; it is often difficult (and risky) to apply a critical security patch if the codebase is already several versions behind.

RISKS AND CONSIDERATIONS FOR RELYING ON OSS IN REGULATED ENVIRONMENTS

In summary, the technical and project risks from 2010 still exist today. However, the use of OSS by pharmaceutical/biopharmaceutical manufacturers has become much more mainstream, and the level of complexity in dependencies has increased. When evaluating the overall risk to regulated systems, it is important to think like a developer. The diligence required to maintain an effective current state needs to be built into your overall IT culture. Relying on a third-party integrator to do this may alleviate some of the operational stresses, but it does not displace the risks involved. And to apply critical thinking to evaluate those risks, you need to understand what you are relying on.

This list summarizes items to consider and provides examples of good practice:

- **Understand what software you are relying on.** Even if you are purchasing commercial software, it likely has components of OSS incorporated into it. It is becoming more common to request a software bill of materials (SBOM) when evaluating new commercial software or validating in-house developed systems. Perform a risk assessment of the specific functions you are relying on.
- **Create an OSS catalog.** Build an inventory of the OSS functionality that is in use within your IT environment to help define a pragmatic governance model and to better understand where you may have risks.
- **Have confidence in the size and sustainability of the OSS community.** Newer software may be more nimble and have better features, but if the community does not have staying power, you may not have support for your system in the future.
- **Look for the use of development standards and good documentation.** Just because the source code is available for the public to review does not mean it is always developed well. Reading the documentation is usually a good indicator of the quality of the software development cycle.
- **Know what version you are using.** If you are using a local distribution of the software, you must verify that your copy does, in fact, match the version you are intending to install. Oftentimes, OSS will have several implementation options designed to accommodate a broad range of users and platforms. You also need to make sure you are downloading from a reputable source (preferably directly from the repo) and take steps to ensure the code was not altered along the way (this is often done with a checksum).
- **Understand the governance model.** Be comfortable with the governance model used by IT, or the service provider, for updates and patches. If you are implementing (or connecting to) a decentralized and distributed software (such as a public blockchain), make sure that you are comfortable with the governance model for that network and have a plan in place should that network become compromised (i.e., run your own archive node so that in the worst-case scenario, you can retrieve the transactions you have posted on-chain).
- **Keep up to date.** Make sure your SDLC process (for both you and your software vendors) requires regular patches and updates. Vulnerabilities are often exposed in software using outdated versions of OSS libraries. Unless you are compiling the program yourself, this is not always apparent.
- **Participate.** Open source works best when there is a broad community, so the best way to get new features that will make your business better is to ask for them. This requires getting involved in the forums and chats. Having this connectivity become part of your IT culture will help ensure that you stay in front of any major changes/disruptions. Regulated companies may consider putting procedures in place for employee contributions to OSS communities, to protect the regulated company from unintended risk to their intellectual property rights or conflict with business objectives.

CONCLUSION

The nature of developing software will continue to evolve as consumers ask for smaller fit-for-purpose applications and software providers push out more frequent updates to keep in front of vulnerabilities. In some cases, code is now being designed to operate privately on public networks, leading us into a world of trusted

algorithms, zero knowledge proofs, and formal verification—many of these advancements are developed under an OSS license. It's likely that reliance on OSS will continue to grow; therefore, it is beneficial to have a strategy in place for relying on OSS within GxP systems.

References

1. Kaufmann, M., M. Ciolkowski, A. Hengstberger, T. Jostes, E. Kruschitz, T. Makait, K. H. Menges, S. Münch, and M. Soto. "Guide for Using Open Source Software (OSS) in Regulated Industries Based on GAMP." *Pharmaceutical Engineering* 30, no. 3 (2010). https://ispe.org/pharmaceutical-engineering/may-june-2010
2. Peterson, S. K. "What's the Difference between Open Source Software and Free Software." Opensource.com. 7 November 2017. https://opensource.com/article/17/11/open-source-or-free-software
3. Synopsys Cybersecurity Research Center. *2020 Open Source Security and Risk Analysis Report.* Synopsis, Inc. 2020. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html
4. Volpi, M. "How Open-Source Software Took Over the World." 12 January 2019. TechCrunch. https://techcrunch.com/2019/01/12/how-open-source-software-took-over-the-world/
5. Jackson, M. "Best Practices." GitGuardian. 5 November 2021. https://blog.gitguardian.com/supply-chain-attack-6-steps-to-harden-your-supply-chain/
6. Oberhaus, D. "The Internet Was Built on the Free Labor of Open Source Developers. Is That Sustainable?" Vice. 14 February 2019. https://www.vice.com/en/article/43zak3/the-internet-was-built-on-the-free-labor-of-open-source-developers-is-that-sustainable
7. Baker, P. *"SAP: One of Open Source's Best Kept Secrets."* Linux Foundation. 20 December 2020. www.linuxfoundation.org/blog/2019/01/sap-one-of-open-sources-best-kept-secrets
8. Grams, C. "4 Reasons Businesses Adopted Open Source in 2020." Opensource.com. 22 December 2020. https://opensource.com/article/20/12/open-source-survey
9. Gavrilova, Y. "18 Machine Learning Tools That You Can't Go Without." Serokell. 24 June 2020. https://serokell.io/blog/popular-machine-learning-tools
10. The R Foundation. https://www.r-project.org/
11. Behrenshausen, B. "An Open Source Analogy: Open Source Is Like Sharing a Recipe." 1 June 2012. Open Source.https://opensource.com/life/12/6/open-source-like-sharing-recipe
12. Github guides (n.d.). GitHub. https://guides.github.com/activities/hello-world/
13. Gallagher, S. "Rage-Quit: Coder Unpublished 17 Lines of JavaScript and `Broke the Internet.'" 24 March 2016. Arstechnica.com. https://arstechnica.com/information-technology/2016/03/rage-quit-coder-unpublished-17-lines-of-javascript-and-broke-the-internet/
14. Stasticts and Data.*Top Companies Contributed to Open Source – 2011/2021.* https://statisticsanddata.org/data/top-companies-contributing-to-open-source-2011-2020/

About the authors

James Canterbury is a Principal in EY's Blockchain Consulting practice, where he focuses on applying decentralized networks to improve transparency and trust within supply chains. His background is in implementing pharmaceutical product quality and compliance systems and helping his clients operate within regulated IT environments. James is an advocate for standards within emerging technology and is a contributor to several open-source development projects. He holds a BS in industrial engineering from Penn State University and is a Certified Information Systems Auditor. James is currently Co-Chair of ISPE's Global GAMP Steering Committee and leads the GAMP blockchain special interest group. He has been an ISPE member since 2015.

Petch Ashida Druar is Manager for Computer Systems Quality Assurance in R&D Global Quality Assurance at GlaxoSmithKline, based in North Carolina. She joined the IT group in 2002, working primarily on computer systems validation projects supporting GCP and GLP business processes. Since moving to CSQA in 2012, Petch has audited external suppliers and internal computer systems for compliance with regulatory expectations for GxP computer systems. Petch received her BS in electronic and electrical engineering from the University of Birmingham, UK, and her MS and engineer's degree in engineering economic systems and operations research from Stanford University. She has been a member of ISPE since 2015 and is a contributing author to the ISPE *GAMP® Good Practice Guide: Data Integrity by Design*.