HARRIS CORPORATION

WHITE PAPER SUBMISSION

NASA SPACE SCIENCE

The Role of Commercial Software in an Open Source World

Authors:

Zachary NORMAN
Harris Geospatial Solutions, Inc.,
303-544-4410,
znorman@harris.com

Daniel PLATT
Harris Geospatial Solutions, Inc.,
303-413-3928,
dplatt04@harris.com

Submitted: January 12, 2018

The Role of Commercial Software in an Open Source World

Zachary Norman, *Harris,* Daniel Platt, *Harris*

***Abstract*—One of the greatest achievements of the Information Age has been the ease-of-access to information, including Open Source Software (OSS) packages. With the rise of OSS and drive to reduce costs in today's world, there is a growing dichotomy between Commercial Off-the-shelf Software (COTS) and OSS alternatives. Because the pros and cons of COTS vs OSS solutions can be one of personal preference and have no clear-cut solution as to which is better, it is proposed that minimizing costs and understanding the difference between COTS and OSS be used as the metrics to help individuals and organizations make the choice that best fits their needs. In addition to this, there may be a middle ground where COTS and OSS are used in conjunction with one another and a broad, open source policy for NASA could cost more than creating open source solutions on top of existing COTS offerings.**

***Index Terms*—Commercial Software, Challenges, Comparison, Open Source Software**

I. Introduction and Background

TODAY we live in a world where many have grown accustomed to having instant access to information without the need to search through books or become experts on different subjects. The same is also true for software on our computers: there are open-source projects for nearly every type of application available today.

These Open Source Software (OSS) packages vary widely in application and complexity. Several prominent examples are Python, Node.js, R, and Linux/Unix distributions. Even this paper was generated using the OSS tools Tekmaker and MiKTeX which are easy to acquire and use. The use of OSS can increase productivity and provide a faster reaction time for organizations [1].

There are several distinguishing factors between OSS packages and Commercial Off-the-shelf Software (COTS). The major differences are that users of OSS will have access to the source code and consumers have the right to freely distribute the software [2]. Apart from these two elements of OSS, COTS software typically involves a business model based upon proprietary software that may entail license sales, product subscriptions, premium support, or consulting. The business model for COTS software varies by industry, but the primary focus is generally software sales and maintenance.

With easy access to open-source software and more pressure than ever to cut costs across businesses and sectors, individuals and organizations are facing the problem of how to reduce capital expenses while maintaining the same level of productivity. One simple way to do this is to remove the acquisition of COTS tools from budgets and replace the COTS tools with free, open-source alternatives.

For some organizations this may be feasible, but it can depend on a multitude of factors, many of which are related to cost. This paper will explore some of the many costs associated with open-source software to help make individuals and organizations make educated, informed decisions on the software that they choose to use.

II. Building OSS Solutions

Before diving into some of the challenges that can be encountered when developing OSS, we have identified two different development routes when creating OSS. These two scenarios

are: creating OSS from scratch or building OSS on top of an existing framework or solution.

The cost of creating a software package from scratch will far exceed that of simply making additions to an existing software solution. The majority of the costs will arise from the initial development in creating the framework to be used along with: testing, debugging, and documenting the solution. These are all important aspects of any software solution to make sure that it is rigorous and users can be successful with the developed tools.

The other option for OSS development is to build on top of an existing framework which can some from OSS or COTS software. An example of this would be creating libraries for Python or the Interactive Data Language (IDL) that are released to the public. This option is ideal because it allows users to focus more on developing algorithms and science instead of a framework for software development. Building OSS on top of an existing framework, whether COTS or OSS, also reduces risk as many bugs will likely have been discovered and fixed already which will reduce the initial investment. However, it is still important that the algorithm developers follow their due diligence to create tests and document their solution even when creating OSS on top of an existing platform.

A. Customer Base

Another important factor when developing OSS is what user group you want to cater to. The user group can be defined as the developer who may be creating the tool for themselves or the general public. It is important that all individuals have equal access to software solutions that have been developed. If these software solutions can only be interacted with programmatically and do not have a user interface (UI), then the number of individuals who can easily use the suite of tools will be greatly reduced. This is important to consider because, when it comes to software development, a UI can take a significant amount of development effort to handle the logic that a few lines of code can accomplish.

III. OSS Costs

OSS can initially be enticing because there is no capital expenditure to acquire the software. Unfortunately the costs for open source software are related to investing time and how much effort it takes to develop and maintain a solution. Additionally, depending on the type of OSS that is developed, there are going to be different costs. Note that some of these costs are also present when selecting and evaluating COTS software packages.

A. Software Development Skills

Time and effort will need to be spent educating developers on the best practices of software development including methodologies and concepts, tools, and coding practices. This applies mostly when developing an OSS solution from scratch without a preexisting framework build upon. This is true because OSS varies in quality and, if you want to encourage individuals to use your tools, then developed OSS needs to have high standards to ensure positive user experiences.

It should be noted that troubleshooting skills are also key for being successful when using OSS for development. Some problems that can be encountered are: library incompatibilities, installation issues, and software bugs. In order to move past these blocks it is important to have the personnel to support developers or that the developers have the skills to account for any issues that are encountered. This may require developers to learn new skills or to hire new personnel to help keep the software development going which will increase the amount of time, and therefore cost, of developing OSS.

B. Software Selection

Compared to COTS software, when choosing an OSS package there are likely going to be many sources to evaluate and compare

against one another. This can be challenging because there may not be time to properly evaluate a software option [3]. This can be further complicated if there are multiple forks, or separate versions, of the same OSS that each need to be evaluated [4]. The lack of time and too many options can then become even more problematic because the quality of open source software can vary from beginner exercises to paid development [5]. In practice, this can translate to bumps in the road during development when you may run into bugs or missing features.

An additional challenge when selecting OSS is that the maintainers may eventually drop support leading to an uncertain future [6]. If an unsupported OSS is critical to development, then additional resources may be needed to maintaining new code or efforts will need to be made to find a replacement. This may not be as much of a problem for larger OSS projects, but is still an important consideration in the selection process.

C. Software Support

A potential set of challenges when using OSS in practice can be software support. This includes: documentation, technical support, and feature requests.

In practice, one of the biggest differences between COTS and OSS is the documentation that has been developed. When purchasing COTS software, there is generally going to be a high level of documentation that has been developed to help guide the user. With OSS, this may not always be the case, or the documentation could be lacking examples to help beginners quickly get started. It should be noted that mature OSS does tend to have better quality documentation [3]. Although documentation can be a challenge, one of the benefits of OSS is that there can be an active user community that is willing to help. If you are using OSS without an active community or large user base, then it can be challenging to get assistance and can increase the cost to learn how to use the software and slow down the development process.

In practice, challenges can also arise for OSS installation and setup. Sometimes these issues can be non-trivial to troubleshoot and solve. Without a dedicated technical support team users can spend hours trying to get OSS up and running.

Another challenge that OSS users may encounter is customizing the original software. If a user needs an addition or new feature, then they can attempt to have the changes included in the project. This can be difficult at times to have the changes accepted and, if they are not accepted, then the user will have to maintain a forked version of the original OSS with the features that they need [8]. With this in mind, it is also important to select OSS that allows for customization.

One of the challenges with all software packages is that a given solution may not fit every need [1]. This is why it is important to select COTS or OSS that allows for customization or extension. This way software can be developed that fits the exact user need to help improve efficiency and quality.

IV. OSS in Practice

The software development cycle can be sped up significantly by using projects that have a solid foundation. This is even true for commercial entities that use many OSS tools for development and testing. We have had much experience using and succeeding with OSS such as Angular and Node.js for developing web applications. Through our use of these popular OSS tools, we encountered: incomplete documentation for some packages and components, libraries/packages that had bugs which required workarounds, and framework bugs for different operating systems (Linux and Windows).

It was expected that some issues would be encountered and, because the software is completely free, it is nothing to be upset or surprised about. One of the major differences

between COTS and OSS is that, with OSS, what you see is what you get. If you want a change then you are likely going have to make the change yourself. This is even more true when your project is time limited: you may not have the luxury of waiting for a feature to be added to OSS (or COTS software) before a deadline.

One challenge with web development that may not apply to other programs or OSS, is that the industry is changing very fast. This means that, when frameworks such as Angular release a new version, there may be compromises on what packages are ready to use with the updated framework. In these situations you can be at the mercy of OSS maintainers to update and test their code so that it works with the newest version of a framework.

V. Additional Considerations

A fundamental challenge to organizations that have been around for a while is that there are likely legacy code bases built upon existing COTS packages. If the goal of an organization is to migrate existing technology to OSS, it doesn't always make sense to re-write this code. There will likely be new bugs created and there may not be an OSS equivalent which can lead to a loss of features/functionality. For example, the ENvironemnt for Visualizing Images (ENVI) is a COTS product produced by Harris Geospatial Solutions for remote sensing and image processing applications. Although there are some open source alternatives, the existing OSS lacks the complete functionality and extensibility that ENVI provides out of the box. Using OSS with a lack of functionality will incur costs to develop the features that are needed and projects will move at a slower pace while users learn how to use new tools. Note that the OSS alternatives to COTS software packages do vary by industry and the example above will not always hold true.

VI. Conclusion

The choice between OSS and COTS software is not a simple one: there are many factors at play and the solution will likely change from organization to organization. Some of the factors that affect software development with OSS will also apply to COTS software as well. Because of this, it is important that decisions be made with thorough research to fully understand the implications from choosing one over the other. Through research and real world experience, we believe that you may also achieve a balanced solution through using COTS and OSS.

Instead of a black and white answer to COTS vs OSS, we propose that there can be a middle ground where appropriate. One of the challenges with some COTS software packages is that the base product is not extendable and this can be a significant reason to choose OSS. However, if a COTS product is easily extendable and can be made to fit user needs, then the thorough testing, documentation, support, and the initial starting point can be a great benefit that reduces the cost and time for creating a solution from scratch. In addition to this, COTS software can also integrate well with OSS which can allow developers to take advantage of OSS within COTS tools.

Here are two examples of software developed by Harris Geospatial Solutions that blend COTS and OSS together in a symbiotic relationship. For example, in the COTS programming language IDL, there is a bi-directional IDL-Python bridge which allows IDL users to ingest and execute Python code directly. This can be beneficial if an algorithm is already present in Python, a developer of IDL will not need to re-write the algorithm in a different language. In addition to this, other types of COTS software can be entirely built upon OSS and, when you have the COTS software, you also have access to all of the source code. An example of this is the Geospatial Services Framework (GSF) which is built on Node.js. When you have GSF installed, you can see each line of code and, if you have the proper skill sets, you can make changes to the software to customize it for your needs without the need

for feature requests or consulting help.

Although OSS can be enticing because it is free, there are still costs which will be mostly tied to investments in employee time. It can take a considerable amount of effort and technical skills to integrate OSS solutions together and create new tools from scratch. While COTS software requires an initial investment, it can drastically reduce the time to develop solutions and provide official support and quality documentation. The solutions built on top of COTS software can also be released as OSS that other users can access. If OSS is built on a COTS product that is widely used, then many individuals will still have access to the tools.

With so many factors to consider when choosing COTS or OSS, ultimately everything comes down to cost and properly evaluating options. Because there are so many solutions to this problem, it is recommended that a broad, open source policy for NASA not be chosen when there are some situations that COTS software can play a positive role in science and algorithm development.

REFERENCES

[1] Chuck Cohn. "Build vs. Buy: How to Know When You Should Build Custom Software Over Canned Solutions." Internet: https://www.forbes.com/sites/chuckcohn/2014/09/15/build-vs-buy-how-to-know-when-you-should-build-custom-software-over-canned-solutions, Sep. 15, 2014 [Jan. 9, 2018].

[2] "The Open Source Definition". Internet: https://opensource.org/osd, Mar. 22, 2007 [Jan. 8, 2018].

[3] Ayala, C., Hauge, O., Conradi, R., Franch, X., Li, J., and Velle, K.S.: Challenges of the Open Source Component Marketplace in the Industry. Proc. Fifth IFIP WG 2.13 International Conference on Open Source Systems, 2009.

[4] Bac, C., Berger, O., Deborde, V., and Hamet, B.: Why and how to contribute to libre software when you integrate them into an in-house application?, Proceedings of the First International Conference on Open Source Systems, 2005.

[5] Linux Foundation. "6 Reasons Why Open Source Software Lowers Development Costs." Internet: https://www.linuxfoundation.org/blog/6-reasons-why-open-source-software-lowers-development-costs/, Feb. 28, 2017 [Jan. 9, 2018].

[6] Bac, C., Berger, O., Deborde, V., and Hamet, B.: Why and how to contribute to libre software when you integrate them into an in-house application?, Proceedings of the First International Conference on Open Source Systems, 2005.

[7] Conlon, P., and Carew, P.: A Risk Driven Framework for Open Source Information Systems Development, First International Conference on Open Source Systems, 2005.

[8] Hauge, Ø., Sørensen, C.-F., and Røsdal, A.: Surveying Industrial Roles in Open Source Software Development: Open Source Development, Adoption and Innovation (2007).