PharmaSUG 2021 - Paper QT-161

"Who wrote this!?" Challenges of replacing Black Box software with more dynamic supportable processes utilizing Open-Source Languages

Frank Menius, Dennis Sweitzer, Terek Peterson, and Monali Khanna, YPrime

ABSTRACT

Good News - the software program that was written and put into production years ago still works. Challenge is dealing with the "Black Box" whereas input is entered, and output extracted correctly but no one truly understands the processes and transformations that take place inside the box. For validated applications, a company may be able to operate like this for a long time but what if a bug is located or the needed output format changes? These applications then must be enhanced or replaced. This paper is about the experience gathered in the effort to replace a SAS macro program utilized in data transfers with a more dynamic and supportable program in R. We will focus both on the technical challenges of converting a solution from one language to another and the inherent challenges of change management. We will share steps and lessons learned that can expedite a similar process.

INTRODUCTION

We have all been there. We have inherited the maintenance of a block of code, a macro, a software package that was developed by someone else. Perhaps this person moved on to another role, or maybe is no longer with the organization. Perhaps this software was developed by a vender or contractor who is no longer supporting it. This package might be used for a vital business function and for months or years it has just performed as expected without the need to be looked at. Specified input is provided and expected output is returned; a validated black box. Now, however, something breaks, output needs have changed, other interfacing packages have been updated, or clients are asking for more functionality. The end result is this software needs to be updated, repaired, or replaced. Thus, we often find ourselves asking, “Who wrote this!?”, as we delve into the challenges of replacing Black Box software with more dynamic supportable processes utilizing open-source languages.

WHY OPEN-SOURCE?

Before we get into the challenges, lets answer the question of why we chose open-source technology like R and Python instead of traditional proprietary options like SAS. There are many parts to this answer, but it culminates in the need to be able to continue to support and develop a software solution throughout the lifetime of its use and not just in its initial inception and validation.

Part of supporting a software package is having individuals that can support it. College graduates entering the work force today are much more likely to have skills and training in open-source technologies than traditional solutions. Even statisticians and statistical programmers are more likely to have used R on campus than SAS. As the innate skills of the workforce transitions it is more efficient to adapt to utilize those new skills than retrain the workforce to meet the old ones. Training to use open-source technologies are also more available and less costly than that of more traditional options. With cost efficient training available a company is able to easily expand skills among its current workforce rather than seek out new specially trained candidates. This even opens up the possibility for cross role functionality.

Open-source technologies are also dynamic and adaptive with a diverse set of developers constantly making changes and updates. New needs or functionalities are often quickly addressed. There is also a large community of collaborative users who regularly exchange information meaning that solutions to development problems are often a quick search away. The dynamic nature of open-source technology does mean that there is a need for structured documentation and qualification of chosen applications. This topic has been thoroughly detailed by Andy Nicholls et al. in “A Risk-Based Approach for Assessing R Package Accuracy within A Validated Infrastructure” and will not be a focus here.

CHALLENGE 1, FIGURING OUT WHAT IT DOES?

If you have been given a piece of software to update, fix, or replace, the first step is likely finding out what it is supposed to do and how it does that. If you have been divinely blessed and are lucky the piece of code you are looking at has well detailed documentation, is thoroughly commented throughout, and the original author is in close proximity. The rest of us however have no documentation, the comments consist of a one-line header, and the author is missing or the vendor out of business.

LOCATING DOCUMENTATION

As organizations grow and mature their documentation needs and capabilities also increase. If the system you inherited has some age you might find that any documentation on the system may not meet your companies' current standards or quality. This documentation may also be in another format or archived as documentation systems are changed. Finding reliable documentation for any given piece of code or application can be a difficult task. It may require tracking down original authors or reaching out to venders who could have collaborated in creating supporting documentation. It likely will mean consulting with quality control departments and searching though documentation archives looking for a validation test plan etc. If the piece of code developed organically you may find that these pieces of documentation just do not exist.

CREATING DOCUMENTATION

If you are unable to find adequate documentation to what and how the piece of software functions, or how it was validated and tested you will likely be presented with the task of creating some documentation yourself. To do this it is often necessary to do the time-consuming work of running a section of code piece by piece and examining logs and outputs files to make note of what each and every line of code does. In short you are adding in the comments you wish had been left for you. It is important to also document what is being done and not only how it is being done. Comment how something is being done in by commenting the code, but document what it does elsewhere. It may be useful to have a centralized location for documenting changes for every variable or output. It may also be helpful to see how the software package is used by business units at your organization. You may be familiar with one aspect of use for example but not aware that another business unit uses the same package for a slightly different function or has a different input file. It will be important to document how all business units interact with the software to ensure that the needs of all stakeholders continue to be supported by the fix, update, or replacement. A tool to help track down all those stake holders may be archived run logs, that may indicate user ID's and dates and times of use. You may also be able to track down and compare output files used by different parts of the organization.

CHALLENGE 2, WHAT DO WE NEED IT TO?

Once you have documented what the code does step by step as well as what the inputs and outputs are it is time to examine how you wish to fix, update, or replace it. To start with it will be important to clearly define the requirements, not only those that caused the need for the fix, update, or replacement but also the existing requirements from all stakeholders. Remember any changes may have unforeseen consequences if all requirements are not accounted for.

IS IT BETTER TO FIX, UPDATE, OR REPLACE?

As you documented the function of the code you may have found areas that were not needed, were inefficient, or poorly coded. You may have even found unknown errors in the code or hot fixes and work arounds that were never reconciled. It is now time to develop a plan to fix, update, or replace the code. Fixing or updating the code may be simpler but requires the ability to continue to maintain it in the future. Replacing the code may be more time consuming in development, testing, and validation but it may have inherent benefits. By replacing the code you may be able to use a more supportable platform that more team members have the skill to maintain. Replacement many also make the package more dynamic and updateable in the future. If your investigation into the code found it was inefficient or error prone replacement may be the ideal solution. In addition, older code may be bloated with multiple updates and outdated functions. Replacing it may mean that you end up with a slimmed down version crafted to perform your needs without the bloat.

IS IT JUST A TRANSLATION?

If you are replacing the piece of software by putting it into another language, should it be just a translation from one language or another? I would argue no. Not only is this an opportunity to make the code more efficient and purpose driven but it may also be an opportunity to reimagine the entire process around its function. For example, we had a SAS code that consumed a .csv to produce and export file. During investigation we found that SAS have some inherent issues with reading in .csv in regard to guessing metadata Through this process we decided that it would be more beneficial to reach out directly to an existing SQL database instead. These types of decisions can be made as you work to improve the overall process and not just the software application.

CHALLENGE 3, TRANSLATING FROM ONE LANGUAGE TO THE OTHER

When you do move to replace a code package with one from another language there are some pitfalls to avoid. Functionality may be the same or different between the languages. For example, SAS and R both make assumptions about reading in .csv files but their assumptions are different. SAS and R both store dates as integers, but one starts counting in the 1960s and the other starts counting in the 1970s. An additional consideration to make is that not all languages handle the same datatypes and have different data structures. This may mean that intermittent steps used in the first language may not work for the second. This is why it was important to establish what the code was doing and not just focus on the how as the how will not be the same.

CHALLENGE 4, TESTING, VALIDATION, AND DOCUMENTATION

As discovered trying to locate and recreate the documentation for the original code it is fundamentally important at this point to make sure your documentation is clear, organized, and complete. Put all the comments into the code that you wish you had inherited. Remember the goal is that this package is dynamic, and supportable and will likely outlive your role in its upkeep.

In addition to code comments, it is important to also make sure that you have clearly documented the functional requirements of the package. This will be important in the testing and validation phase as these requirements are what you will be building a test script for and testing against. It will be important to work with your quality control department to ensure that you have clear, thorough, and repeatable testing procedures with documented objective evidence. Again, the hardest part of this process for us was locating or creating documentation of the original code, therefore it can be argued that documentation is the most important task going forward.

CONCLUSION

Fixing, updating, or replacing Blackbox software applications can be challenging and time consuming. There are challenges around documentation, understanding, translation, and validation. The road may lead to using more dynamic and supportable software application and a reimagining of the overall business practice. It could also involve the use of new workforces and a new technology. Fundamentally though all effort should be made to eliminate the black box through complete and detailed documentation. The last thing you want at the end of the process is to have created another black box without adequate documentation that someone else will have to decipher in the future.

REFERENCES

Nichols, Andy; Bargo, Paulo R.; Sims, John. “A Risk-Based Approach for Assessing R Package Accuracy within A Validated Infrastructure.” R Validation Hub. January 23, 2020. Available at https://www.pharmar.org/white-paper/

CONTACT INFORMATION

Monali Khanna
YPrime
9 Great Valley Parkway
Malvern, PA 19355
mkhanna@yprime.com
https://www.yprime.com/
https://www.linkedin.com/in/monali-k-a259a5178/

Frank Menius
YPrime
3301 Benson Dr #300
Raleigh, NC 27609
fmenius@yprime.com
https://www.yprime.com/
https://www.linkedin.com/in/frank-menius/

Terek Peterson
YPrime
9 Great Valley Parkway
Malvern, PA 19355
tpeterson@yprime.com
https://www.yprime.com/
https://www.linkedin.com/in/terekpeterson/

Dennis Sweitzer
YPrime
9 Great Valley Parkway
Malvern, PA 19355
dweitzer@yprime.com
https://www.yprime.com/
https://www.linkedin.com/in/dennissweitzer/