PhUSE 2015

Paper PD11

The Rewarding (and Rocky) Road from SAS Applications Programmer to SAS Software Developer

Jeffrey Abolafia, Rho Inc., Chapel Hill, NC USA
Carrie Dundas-Lucca, Zencos Consulting, Cary, NC USA

ABSTRACT

There are many satisfying career paths available to SAS programmers within the pharmaceutical industry. A common progression begins with production programming, creating datasets, tables, figures, and listings, then moving to developing small utility programs and, later, developing tools for corporate-wide use. The skill set required at each stage of this path varies, often significantly. Failure by the programmer or management to satisfy the differing requirements of the production/developer roles can result in software that is difficult to maintain, poorly documented, and not properly validated.

This presentation identifies the different requirements at each phase of a SAS programmer's career in the pharmaceutical Industry. We first describe the standard career path of a SAS programmer in the pharmaceutical industry. Next, we describe the skill set and tools that are required to make the successful transition from production programmer to software developer. We place particular emphasis on how the developer mindset (project life cycle, validation, and workflow) differs from the production programming environment. Finally, we will highlight a few of the challenges SAS Software Development teams might have to overcome to achieve success.

INTRODUCTION

SAS Programmers perform a wide variety of activities within the pharmaceutical industry. These activities or roles tend to vary throughout the career span of a SAS programmer and have evolved over time. As in most professions, SAS programmers perform relatively simple tasks at the onset of their career. As they gain experience over time, SAS programmers take on increasingly complex assignments. Coincident with these career paths, the nature of the SAS programmer role is evolving in the pharmaceutical industry. Many SAS programmers are now expected to perform tasks associated with the clinical data scientist role. In the next section we examine the typical career path of a SAS programmer in the pharmaceutical industry and how it has changed over time. In particular, emphasis will be placed on one of the relatively new career growth options to SAS programmers: becoming a Software Developer.

The information presented in the following sections is derived from the authors' working in the pharmaceutical industry for over 25 years, in both academic and non-academic environments, and meeting the challenges posed by more than 50 sponsors.

THE CAREER PATH OF A SAS PROGRAMMER

Entry level SAS programmers (or statistical programmers in general) can broadly be characterized as "application programmers", focusing on project work. They usually start by performing less challenging tasks such as producing data listings. As their SAS programming expertise develops, a common progression is from listings to simple displays, then to complex displays, then simple dataset programming, and finally to creating more complex datasets. SAS programmers who are also statisticians

may have a different path: starting with performing statistical analysis on existing datasets and then moving on to dataset creation or validation.

As SAS programmers gain experience they usually recognize that repetitive tasks can be automated. As a result, the next step in their careers is usually developing small utility programs. These programs are characterized by performing a single task, are usually SAS macros, and are only used by a handful of internal staff. Examples of small utility programs include macros to rename variables, format variables, recode variables, or store a list of variables in a macro variable.

Over time, as their familiarity and expertise with the SAS language grows, SAS programmers continue to develop utilities with increasing functionality, complexity, and broader usage. As some point the simple utility evolves into a system of utilities or a software solution for a complex problem. These tools typically perform several tasks, are used throughout the company and are comprised of a system of related programs or macros. A corporate-wide reporting system or software to produce a submissions package are examples of more complex utilities.

Adding the software developer role to the application programmer role typically occurs slowly over time and is driven by need and increasing experience with statistical software. It also may not by a pre-planned decision. In fact, we propose that when this occurs, most application programmers do not even realize they are now also software developers. This can be problematic as the work processes and skill sets needed to perform these two distinctly different roles vary tremendously. In the next section we examine both of these roles and how they diverge, with a focus on several areas: workflow and work processes, validation and documentation, and the skill set and tools required to perform the role.

THE APPLICATION PROGRAMMER ROLE

WORKFLOW

Application programmers typically work on creating data sets, listings, displays, and statistical analysis. These outputs or deliverables are pretty well defined and rigorously spec'ed out. Specifications for these deliverables usually originate from project statisticians or programmers, or may even be written by the programmer performing the work. Specifications or requirements for dataset creation are generally produced in spreadsheet format, while specifications for analysis or displays are normally stored in spreadsheet format, as an annotated display, text, or some combination of the three methods. Additional documentation may also include the protocol and the Statistical Analysis Plan (SAP). Using these specifications as input, the programming for a given dataset or display is more often than not completed by a single programmer (a second programmer usually does the validation). The customer or recipient of the output is usually the project statistician or another programmer. Next the programming is validated by double independent programming, which is still the industry standard (validation will be discussed in detail below). Over the course of a study, dataset and display programs may require updates as more data becomes available or additional displays or analyses are requested. Upon study completion programs are archived and may be re-used for subsequent studies. However, programs normally do not require updates, maintenance or support beyond study completion.

VALIDATION AND DOCUMENTATION

For better or for worse, validation for datasets and displays in the pharmaceutical industry usually means double independent programming. In this workflow one programmer is given the role of production programmer and produces the deliverables used for production. A second programmer is designated as the validation programmer and is tasked with reproducing the work of the production programmer while being blinded to the code of the production programmer. The production and validation programmers work iteratively until their outputs match. Once the double independent programming is complete, a statistician typically reviews outputs for accuracy and consistency.

Over the past several years, CDISC[(1)] standards have evolved from nice to have to all but required. This has added an additional component to validation. CDISC submission databases must now be validated for compliance to the relevant standard such as SDTM or ADaM. This is usually accomplished using an off the shelf product such as OpenCDISC.

Documentation in the realm of production programming takes a variety of formats. These include: 1) specifications described above for datasets, displays, and analysis; 2) internal document contained within a program which contains specific information about the program and its history (this is also where the documentation for simple utilities is routinely stored); 3) a spreadsheet or tracker which may document anomalies, interactions between the production and validation programmer, and other relevant communication; 4) a signoff area which demonstrates programming, validation, and statistical review have been completed and 5) the define file and reviewer's guide (now expected by regulatory agencies) which provides information about CDISC-compliant submission databases.

SKILL SETS AND TOOLS

SAS programmers within the pharmaceutical industry come from a wide variety of backgrounds and possess a diverse set of skills. Not surprisingly, SAS programming is the one skill they share. In addition to SAS, many SAS programmers must now be proficient with R as its use is rapidly increasing in the pharmaceutical industry. SAS programmers should also be adept at using other statistical software packages, such as WinNonlin® for PK analysis, for more specialized applications. Outside the realm of statistical software, SAS programmers must be well versed with the Microsoft Office suite of products, how to integrate these products with SAS or R, one or more scripting languages, and the operating system in which they are programming. Finally, in order to produce CDISC-compliant deliverables, SAS programmers must be well versed in CDISC standards and competent in XML and XML related technologies as well as CDISC validation software such as OpenCDISC[(2)] .

THE SOFTWARE DEVELOPER ROLE

WORKFLOW

The workflow for a software developer differs from that of an application programmer in a number of key ways. First, software development projects are guided by the software development life cycle and the software development methodology, both of which can be specific to individual software development projects. In contrast to their previous role as an application programmer, a SAS programmer who grows into a software developer will participate in developing the software methodology for their projects. Thus providing guidance for the project rather than strictly receiving and processing directions from others.

The activities associated with a software development project are also very different from those of an application programmer and require a more interpersonal approach. Software developers often need to interact with their end-users in order to participate in the user requirements-gathering process. They will also need to interact on a very regular basis with their software development and validation team members. As the software is developed and proceeds into the informal and formal testing processes, the software developer will continue to interact with the project team members to resolve bugs, improve functionality, correct mistakes in the draft design, and so forth. The activities of a software developer require a significant amount of interaction with others and do not allow for a silo that characterizes the application programmer.

One of the most critical distinctions in the workflows between application programming and software development is the endpoint, or lack thereof. For application programmers, the end of a project is typically associated with the final delivery of the submission package to the sponsor and ultimately, the FDA. In software development, there really is no such thing as an endpoint since software applications require ongoing maintenance, updates to functionality, expansion in scope, and support through its lifetime. While the release of a software application might be considered a major achievement, it should not be considered the end of a project. The true end of a software development project is when the software is

decommissioned which might not happen for decades, if ever. The length of time a software developer might commit to a single development project is vastly different from that of an application programmer. In short, a software developer might develop a piece of software and support it for the remainder of their career!

VALIDATION AND DOCUMENTATION

Unlike the role of the application developer, the software developer plays an active role in developing the documentation set for their projects. Software developed or implemented in the pharmaceutical industry has to be analyzed for compliance with CFR Part 11 and depending on the outcome of that analysis, it might also need to be validated before it can be released for use. For all software development projects, it is good practice to document a handful of items such as user requirements, design specifications and/or functional requirements, test scripts, a traceability matrix, and incidents. For software that requires validation, the documents listed out above, plus a validation plan and report should also be developed. The software developer participates in writing or is the primary author of many of the documents necessary for a software development project because of their expertise on the subject.

Another key distinction between application programming and software development is validation. As described earlier, validation for an application programmer comes in the form of double independent programming. But in software development, validation is achieved through the documentation of user requirements, development of the software based on those requirements, and testing to confirm the software meets those requirements. An analysis of the risks the software introduces based on the CFR Part 11 requirements is also performed and all risks must be mitigated to an acceptable level before the software can be considered validated and be released for use. The activities associated with validation include a significant amount of interaction with the project team and documentation galore, as described in the preceding section. Once again, software development projects require SAS programmers to develop strengths outside of the SAS programming language. This can prove to be quite a challenge.

SKILL SETS AND TOOLS

While technical skills are an obvious requirement for both a SAS application programmer and a software developer, the latter must also possess a set of soft skills that assist them in interpersonal interactions and communications. The natural progression of an application programmer to software developer oftentimes takes place as the result of the SAS programmer's technical strengths and not necessarily because he or she possesses the soft skills necessary to succeed as a software developer. In the recommendations section of this paper, we will discuss how this disconnect can be addressed by organizations that grow their application programmers into software developers.

The role of a software developer differs from that of an application developer in a number of ways already described in this paper. One that took us a bit of time to pinpoint is the transition of the individual's role from that of *client to support*. In the role of application programmer, you are the user of software applications (aka, the client) that support you in your day-to-day work activities. Software developers function in a support capacity by developing software solutions that provide support to end-users in their day-to-day activities. The switch from client to support is a challenge in and of itself but it is even more difficult to achieve when trying to adjust from client to support for the very same set of work activities. As we've experienced, application programmers who grow into software developers oftentimes do so by creating software solutions for the very work they do as application programmers. When doing so, they typically serve in both client and support roles, which creates confusion and can be counterproductive. When the transition is made to a software development role, the former application programmer must take off their client "cap" and replace it with their support "cap" and function exclusively within that realm.

Below is a table that compares the two roles based on the key attributes described in the previous two sections of this paper.

Key Attribute	Application Programmer	Software Developer
Project Specifications	· Written by others · Delivered to programmer · Programmer might provide minimal feedback or input · Spreadsheet format	· Participates in collection and documentation activities
Validation approach	· Double independent programming · Outputs reviewed by statistician · Minimal documentation	· Testing software to confirm it meets user requirements · Risk analysis and mitigation · Extensive documentation
Project outputs	· Executable SAS programs	· Software
Project timeline	· Multiple months · Delivery of programs equates to the end of the project	· Multiple months · Delivery of software is only the beginning; software requires ongoing updates and support
Required skills	· SAS · CDISC standards · Other statistical programming languages and tools · Microsoft Office	· Ability to write/design software · Communication · Collaboration
Mindset	· FDA is my client · Team of one	· Users are my clients · Team member

CHALLENGES

As SAS programmers become more proficient over time, adding the software development role can be quite rewarding. However it does come with a series of challenges. The primary challenges that we have observed result from the distinctions we've described in the previous section between the application programmer role and the software developer role.

The software developer role requires a different mindset than the application programmer role. The following quote best describes this challenge: "It is not what *you* think they need". One of the greatest problems we have seen with SAS programmers making the transition is that they underestimate the need to continually talk with end users. Software developers have to reach out to their customers to gather requirements and find out what's working and what's not working. As a software developer you may also be one of the end users, but you are not the only end user. It is imperative that software developers engage with a representative set of all user types and not proceed with development until the proposed design will meet the needs of all user types.

A second issue within the realm of mindset is project duration (development life cycle). Deadlines for production programming are generally tight and project duration is relatively short. Most project related tasks such as dataset and display programming can be completed within a few days. In contrast, software development projects are long term projects consisting of several stages. There is a planning phase where the developer (and his/her team) gather user requirements, develop functional and/or design specifications, produce a project plan, and design a validation strategy. Phase two is the development phase, where code is written and reviewed. In phase three, validation and user acceptance testing is performed and the software is released. The final phase, maintenance, is best characterized by the saying "if you build it, you own it". It's also a concept foreign to application programmers. It can go on indefinitely and is where updates are made and support is provided.

Closely related to the mindset challenge are challenges related to communication and documentation. Simply put, software development requires a lot more communication and documentation than application programming. Application programmers typically receive structured specifications and interact with their project team members via email and an issue tracker. Communication is likely to be with other programmers and project statisticians, data managers, and project managers. Documentation is usually limited to comments and documentation contained within programs, specifications and other information needed to produce the define file, and assisting with the reviewer's guide. On the other hand, software developers have a wider communication stream, interacting with validation staff, a diverse set of customers, as well as other programmers and developers. Communication also is also continuous and more frequent as software developers are constantly interacting with end users/customers as well as validation personnel. The documentation required to successfully support software projects can be extensive and it is a necessity at each stage of the software life cycle development process. It includes gathering user requirements, system design specifications, a validation and testing plan, a project plan, documentation contained within programs, using a tool such as Jira to track bugs, requested features, and status, writing test scripts, and producing a guide for end users. Furthermore, documentation must be developed when updates are made to an existing system. We have found that getting used to the amount of documentation required is one of the most challenging aspects of making a successful transition from application programmer to software developer. We would also argue that most application programmers are surprised at the volume of documentation required to support a software development project and thereby ill-prepared to participate in developing it.

The area of validation also presents a series of challenges. Within the realm of production programming, validation usually means double independent programming. For software development, validation is not so cut and dry. The validation methodology can differ for each software development project. At the start of each project a developer will usually work with a validation specialist to develop a custom plan for the project. Developers must also learn to write user acceptance test scripts, support user acceptance testing, and document testing results. In addition, when updates/fixes are needed, developers must participate in deciding on the level of validation required.

In summary, in order to make a successful transition from production programmer to software developer, a programmer must get used to lengthier projects, programs/software that may require infinite support, enjoy (or at least get used to) more diverse and extensive communication, documentation, and validation. If you just want to primarily write code, transitioning to software development may not be a good fit.

RECOMMENDATIONS

In this section we provide recommendations for both application programmers and managers to facilitate a smooth transition to the software developer role.

Recommendation #1: See the Train Coming

As an application programmer's skill set grows over time, the utilities or software they develop typically becomes more complex and its usage becomes more widespread. So, the key is recognizing when statistical computing programming methodologies no longer apply and software development methodologies are needed. Two indicators that an application programmer is crossing the line are: 1) Functionality: the software is performing more than a single task and 2) User Base: the software is being used by individuals other than the programmer. We recommend checking with a validation specialist or others with CRF Part 11 knowledge to help determine the appropriate methodology to follow.

Recommendation #2: Provide training

Once a programmer or manager recognizes that an application programmer is starting to develop more complex software, it is incumbent on management to provide training on software development methodologies. Many of the techniques and tools (such as code versioning and revision control systems) employed by software developers are not commonly utilized by application programmers. In fact, many application programmers are unaware of them.

Recommendation #3: Make some new friends

Develop working relationships with staff in software development and validation. They can provide training, support, assistance, tools, and templates to facilitate a successful transition into software development.

Recommendation #4: Embrace collaboration and change

This one is primarily for management. Establish an environment that promotes collaboration across functional areas and relationship building with other groups. It has been our experience that validation and documentation related activities are not the favorite tasks for a majority of statistical programmers. Many statistical programmers see these types of activities as unnecessary paperwork and avoid them when possible. We strongly recommend that management create an atmosphere that promotes interaction with validation and quality assurance/control groups and that stresses the importance of the activities performed by these groups.

Recommendation #5: If you enjoy it, do it.

If you are an application programmer and really enjoy documentation and validation, it could be that developing software is your calling. While we have only come across a handful of application programmers with these characteristics, each one of them has turned out to be a successful software developer. On the other hand, if documentation and validation are tasks you don't enjoy, you are likely more valuable as a production programmer.

CONCLUSION

As statistical programmers develop their skill set over time, taking on the software developer role can be rewarding to both programmers and their organizations. However it is vital that both programmers and their managers realize how very different these two roles are and to recognize when an application programmer is starting to cross the line into the realm of software development. Failure to do so will often result in software that is poorly documented, hard to maintain, and does not meet the needs of its intended end users. On the other hand, if management and programmers recognize this transition early on, they can create an environment that facilitates success by: providing training in software development life cycle and methodology, the tools and skillsets required to be a successful software developer, and by maintaining an open and collaborative work environment.

REFERENCES

(1) "Standards and Implementations". Online at www.cdisc.org/standards-and-implementations

(2) "OpenCDISC". Online at www.opencdisc.org.

ACKNOWLEDGMENTS

I would like to thank my colleague Frank DiIorio for his valuable input during preparation of this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Jeffrey Abolafia
Rho, Inc.
Jeff_abolafia@rhoworld.com

Carrie Dundas-Lucca, MSIS
Zencos Consulting
cdundaslucca@zencos.com