PharmaSUG 2019 - Paper SI-190

Integrating programming workflow into computing environments: A closer look

Tyagrajan Swaminathan, Sridhar Vijendra, Ephicacy Lifescience Analytics

ABSTRACT

System-level workflows are well known to improve process compliance, increase traceability and enhance audit trails in various domains including document management. It is time programming workflows are made integral to the domain of statistical programming for clinical trials to enhance productivity, reduce management overhead, ease tracking of large programming deliverables produced by globally dispersed teams and consequently, reduce time-to-market for every drug. System-level workflows are mapped based on the corresponding real-world processes that they model but conventional statistical computing environments (SCEs) that most programmers are used to hardly allow for automatic record keeping of processes taking place within the environment. What is recorded in a disparate programming status tracker has scope for human error since there is a possible lag and/or a gap between what is done in the programming environment and what is mentioned in the status tracker. This is where modern-day web or cloud-based SCEs make a difference, by bringing in integrated customizable workflows to streamline statistical programming activities. There is a need for workflows that integrate with our programming environment to keep track of programming tasks and record their completion without convoluted user actions. But, is it possible to exactly mirror the real-world processes within such programming environments? Are human-error-prone processes suitable for machine-compatible checks or will we be left to force-fit workflows into systems that do not want to co-operate?

INTRODUCTION

What really is a Statistical Computing Environment (SCE)? Our programming teams are all too familiar with the complex folder structures we use in our respective organizations. There is always an attempt to standardize the folder structure across projects and studies within an organization to allow for easier navigation. Almost always, SAS® is the tool that is integrated into such a data server and a version control system is tied to it to allow programs to be created, edited, and archived. Such systems come with their own complexities and problems, as they require a bunch of security settings and folder permissions that usually call for the expertise of an IT administration team. Such systems are what can be referred to as “conventional SCEs” since they provide an environment for creation and archival of study artefacts (tables, listings and figures (TLFs) and datasets) and study related documentation but they fall short because they don’t share the important attribute of having a detailed and easily accessible audit trail, which is the hallmark of an SCE [1, 2]. Audit trails can track everything from file edit time stamps to deliverable timelines and go as far as tracking user access history. Audit trails are very much necessary to ensure process compliance and help trace back instances of lack of compliance that can be used later to improve processes and possibly upgrade the functionalities within the SCE. Time stamps of file edits are routinely captured in all conventional systems, but any instances of file views or dataset access cannot be captured. The simplest example would be that of a QC programmer in a double programming activity peeking at the main side programmer’s code inadvertently or intentionally.

In addition, conventional SCEs do not have the capacity to integrate project management activities within their functionality and tracking of project status and the actual tasks being carried out in the system end up as disjoint activities. In order to understand this better, let us consider the way documents are handled in document management systems. Almost every document in a clinical trial goes through multiple reviews and edits by various team members until a final version is signed off and distributed for reference. This process from creation to review to re-editing (incorporating review comments) to a final approved version is often captured in document management systems in the form of a document management workflow which follows a specific standard operating procedure (SOP) set by the organization. In this process, the artefact (the document) changes states from *draft*, to *in-review*, and then finally to *approved,* as it moves from one team member/department to another. A final sign off for the approval is then archived in the Trial Master File. Hence, all documents go through a *document workflow*.

Depending on the organization, each document could have a specific workflow such as a *Protocol document workflow* for the study protocol, and so on. Document management workflows are now a routine part of document management systems, such as SharePoint. Why don't SCEs have such workflows for programming activities?

Modern day SCEs, cloud-based and otherwise, are making this possible in various ways [1, 2, 3, 4, 5]. In conventional SCEs, programming activities are tracked with the use of the infamous programming status tracker. There is a disconnect however, in what happens in the real world versus what is captured in the tracker. The DM dataset may be made ready-for-QC one morning but may not be marked as ready-for-QC in the tracker until the following morning because the programming status tracker was locked by somebody else for editing and the programmer forgets to follow up. An instant message sent by the programmer to his QC counterpart about the dataset being ready for QC is obviously not captured in the SCE. In another case, a deliverable may be sent out on Friday evening and marked in the timeline document only on Monday morning with an earlier date. This may be acceptable in most cases, with such discrepancies dismissed during an audit as *findings* but there is a way to make this all foolproof and seamless for the team. Modern day SCEs now offer integrated programming workflows that can improve traceability and reproducibility and maintain a detailed audit trail without radically affecting the way we work.

Modern SCEs are largely based on user-friendly well-engineered interfaces where users are not required to memorize cryptic commands to make things work. One of the ways to streamline project management for statistical deliverables is to have workflows in the SCE that can be checked off by team members as and when steps in an activity are completed. This could be a programming activity such as creation and QC of a dataset or it could be a project management activity such as completion of a large deliverable with various components put together by a distributed team. Having such a feature in an SCE would mean that a programmer would not need to open and update the dreaded status tracker after every dataset or table is completed. This would also mean that a project manager or team lead could simply refresh their screen every few hours to see how the work on a deliverable was progressing instead of repeatedly asking for updates from the team. And, elaborate status reports can be generated quickly with a click of a button.

However, like with all large systems with advanced functionalities, such SCEs need a significant amount of setup to be done in the back-end before such features can be used by programming teams on a routine basis. It is possible to hard-wire workflows into such systems but hard-wiring is not a one-size-fits-all solution even within the same organization. Hard-wiring of workflows makes workflows unusable if organizations want to tweak their processes on a regular basis using learnings along the way. Obviously, this calls for **customizable** workflows in an SCE, that allow for flexibility in adjusting the workflow as per the organizational SOP for each activity. Once they are put in place, they can be used by programming teams for routine programming activities. And as and when necessary, these template workflows can be tweaked to cater to project-specific processes. Undeniably, workflows can improve process compliance and enhance record-keeping by recording every action for every workflow created for every artefact. Custom creation of workflows is very much needed but just how much customization is necessary? How well can you fine-tune a customizable workflow? Will programming workflows change the way we work in the real-world? Will they change them for better or for worse? Will such workflows make documentation-averse programmers fall in line or more out?

This paper examines in detail the highlights and challenges of working with customizable workflows and what it takes to set them up and use them in day-to-day tasks. The mapping from a process to a workflow is carried out for a sample scenario typical of statistical programming activities. The focus is on understanding the nitty-gritty of workflows as applied to managing typical programming activities in an SCE that allows for integrated customizable workflows.

CREATING A WORKFLOW FOR A PROCESS

How is a workflow created and used in an SCE? How does a process become a workflow? How is it integrated into the activities that are carried out in the system? Creation of a workflow can be illustrated with a sample process such as the dataset creation and QC process. Almost every dataset in a study, including SDTMs and ADaMs, go through a creation and QC programming process, followed often by a

review from a senior team member. One way to use a programming workflow here is to consider that every dataset that is created should go through the workflow in the system. This would require a **workflow template** to be created so that instances of the workflow can be used for each dataset that is programmed. These individual instances will be referred to as **work items** and they typically correspond one-on-one to the entries in the programming status tracker. For instance, the dataset DM will need a work item, with the main side assigned to programmer A and QC side assigned to programmer B. CM will be another work item, with main side being done by A and QC side being programmed by C, and so on. The terms workflow template and work item are defined below.

WORKFLOW TEMPLATE

A workflow template corresponds to one process as defined in the programming SOP. An example for this could be the process of creation and QC of SDTMs. The workflow template created for this process would be the blueprint of a set of work items, all of which follow the same process as configured in the template. In our example, it would be a work item each for every SDTM in the study. A system might have only a handful of templates, one for each type of programming activity. These templates are used repeatedly, in the work items.

WORK ITEMS

Work items are instances of the workflow template. A system will have many hundreds of work items, one for each artefact created in every study, possibly for every deliverable. This is illustrated in Figure 1.

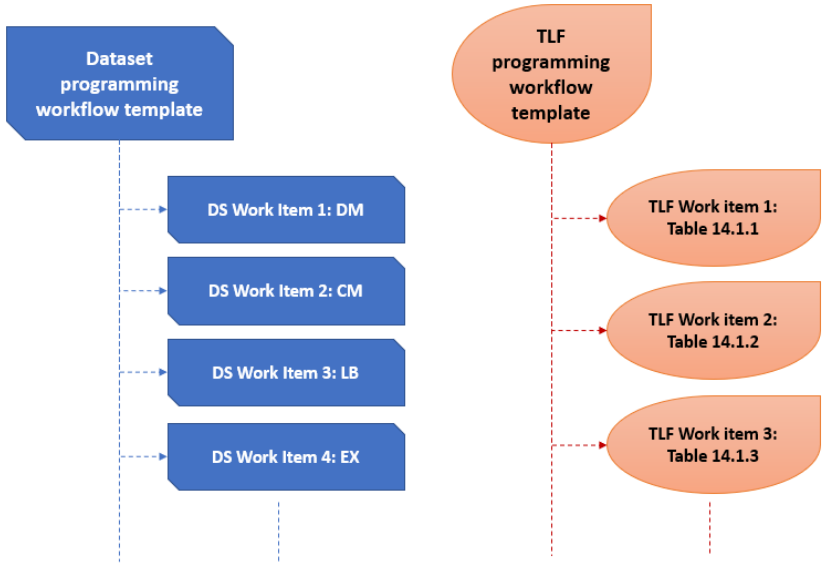

Figure 1. Workflow templates and work items

COMPONENTS OF A WORKFLOW

Before we proceed to dissect a sample programming process to map and create a workflow for it, let us become familiar with some additional nomenclature related to workflows. This nomenclature is neither universal nor standardized but will simply be used to explain the concepts in this paper:

STEPS

A step is the basic building block of a workflow and can be a sub-process within a process. In the most simplified version of a dataset creation and review process shown in Figure 2, step 1 of the workflow would involve creation of a dataset (assuming no double programming involved) and step 2 would involve review of a dataset by a statistician. Steps in a workflow may be sequential or parallel. As will be described later in this paper, the dataset programming and QC activity happens to be a distinctly parallel step while most review steps are sequential.

ASSIGNING STEPS TO USERS

Similar to how an entry is made for each programmer assigned to an artefact in the programming status tracker, users are assigned to work items in the SCE. Specifically, users are assigned to specific steps in the workflow in an SCE. This is illustrated in the example shown in Figure 2.

STEP STATES

A step can exist in one or more states. Once a work item is created, a diagram of states can be shown for all the steps in a work item at any point in time. For example, when a dataset creation process is in step 1 with the state of *Programming in progress*, the dataset review step is in the state of *Not started*.

STEP RESETS

A step can be reset to its initial state with a step reset. For instance, a step state can be reset from a *Completed* to a *Not started* state by a specific trigger. This trigger will most probably come from another step from which a feedback exists.

STEP NOTIFICATIONS

When a step is completed, and the workflow moves to a subsequent step, a notification is triggered to the user assigned to the next step. For instance, in the example shown in Figure 2, when the programmer creating the dataset completes their programming, a notification is sent to the user assigned to the subsequent step that the dataset is ready for review. Notifications can be sent in various forms, including email or a dialog pop-up within the SCE.

STEP REGISTRATIONS

Completion of a step in a workflow can be registered (recorded) in the system either manually or automatically. This is the essential part of the audit trail in the system. Manual step registrations are simpler but involve additional user action. Configuring the SCE for automatic step registration necessitates the overhead of the setup involved and is also prone to inadvertent step completions.

STEP OVERRIDES

Override of a step means that a user marks a step as complete or as skipped without performing the action required of the step. Override configuration will need to be done prior to step assignment. An example of a step override is that of a dataset review that is assigned to a lead programmer but is overridden by the statistician, because they find it unnecessary for certain reasons. The override is also captured by the SCE.

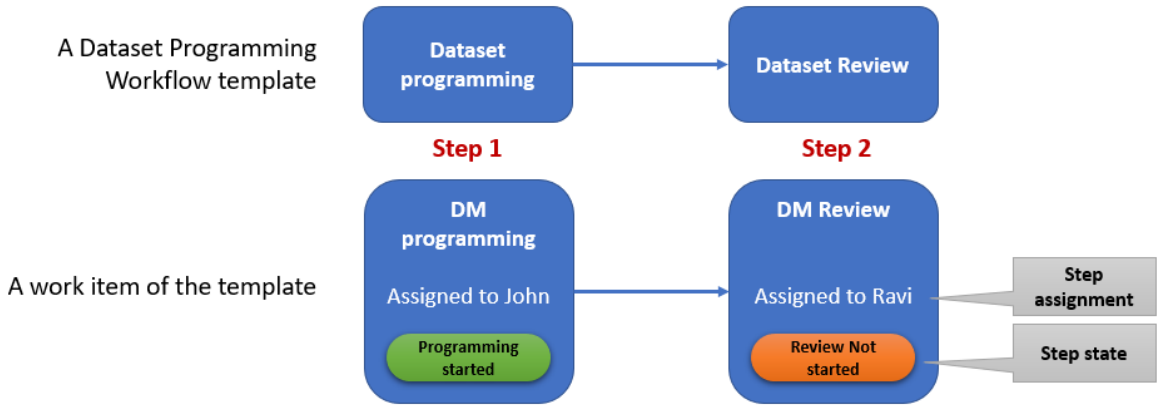

Figure 2: Components of a workflow

THE TLF PROGRAMMING WORKFLOW

Let us consider the typical scenario of a workflow to create, validate and review a TLF artefact, something that happens in our teams, day in and day out. Figure 3 illustrates this process. Additional details such as review of mocks before start of programming, code review after programming, log checks, version control check-ins, peer review of cosmetics, etc. have been left out of this process for the sake of simplicity.

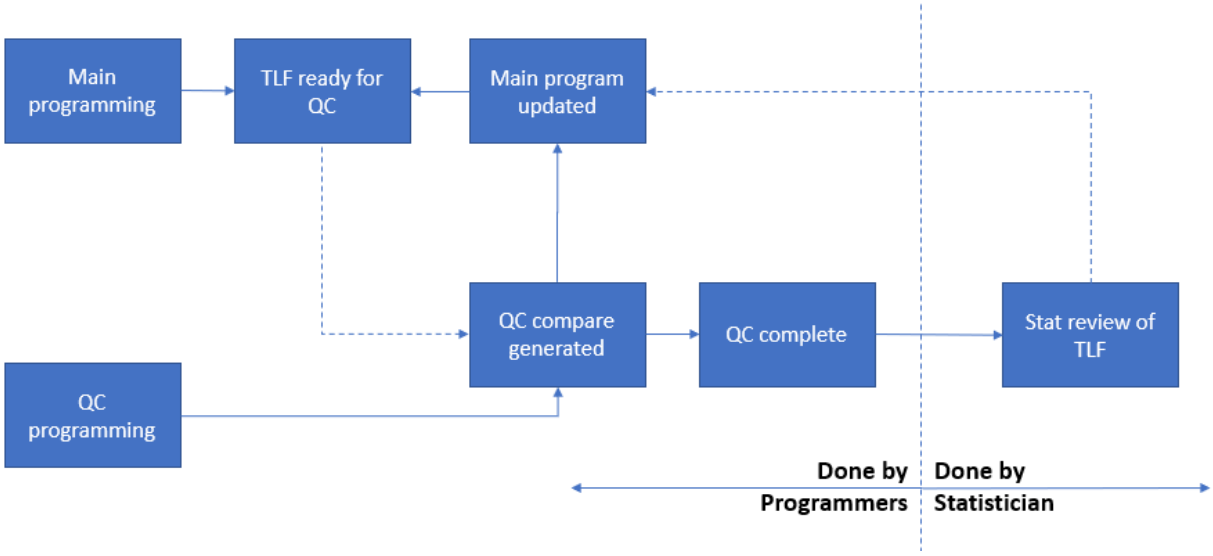

Figure 3: A typical TLF programming process

What would be the best way to convert this process into a workflow in an SCE that will allow work items to be created for each TLF artefact that is programmed for every deliverable? To ease the process of creating and customizing the TLF workflow, here are a few premises:

- The TLF mock shells are ready for programming. It is apparent that if the TLF mock shell creation and review must be a part of this workflow, then it will be another step that comes before TLF programming
- Minor processes such as review of mocks, code review, log checks and cosmetic checks albeit important, are not going to be included in the workflow for now.
- For the purpose of customization, it is assumed that there are only 2 types of steps in the SCE:
    - a simple sequential step that can be assigned to a single user
    - a parallel step that can be assigned to 2 users.

  A sequential step cannot be assigned to 2 users at the same time because if a step needs 2 users to complete it, it clearly indicates that there is a sub-process involved that the 2 accountable users need to complete one after the other or at the same time. Conversely, a parallel step cannot be assigned to a single user since 2 parallel activities assigned to the same user can be clubbed in a single step.

WORKFLOW VERSION 1

Let us consider the most simplified workflow derived from the above process. In this simplified workflow which we will refer to as Version 1, the first step is double programming and the second step is TLF review as shown in Figure 4. Now the question arises as to who should be assigned the first step – the main programmer or the QC programmer? And as per the premise mentioned above, one step cannot be assigned to 2 users. Hence this version of the workflow does not align itself to the actual process that is shown in Figure 3 and does not capture many of the states of the typical QC cycle.

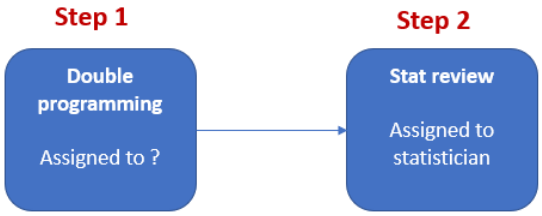

Figure 4: TLF Workflow Version 1

WORKFLOW VERSION 2

The next logical step would be to split the first activity of double programming into 2 steps so that it can be assigned to 2 separate users. This version is shown in Figure 5. This would mean that main programming is started first and when that is complete, the QC programmer gets a step notification to start programming. But this presents a problem, since very often, QC programming can be done in parallel with main programming.

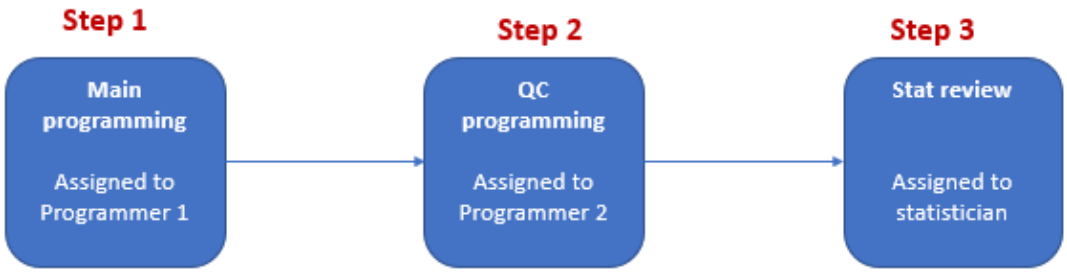

Figure 5: TLF Workflow Version 2

WORKFLOW VERSION 3

It is easy to see that if we quickly change the first 2 steps of version 2 in Figure 5 to a parallel step as shown in Figure 6, main programming and QC programming can be assigned to different users and more importantly, both these activities can be started independent of the other.

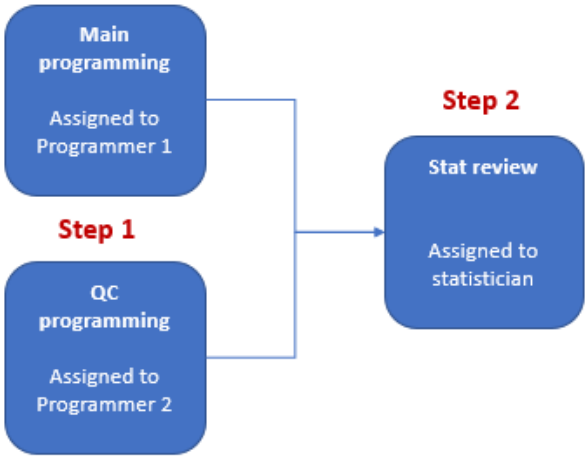

Figure 6: TLF Workflow Version 3

The challenge however is about how the QC programmer is intimated about the TLF output being ready for QC? There is a step notification when main programming is completed, but the notification goes to the statistician for initiating review. When does the QC programmer know when to start QC?

WORKFLOW VERSION 4

Version 3 of the workflow shown in Figure 6 does not address the issue of the QC programmer getting the necessary trigger to start the QC process, hence we can try adding another step to the workflow between the parallel step and the review step. This results in the workflow shown in Figure 7. When the main programmer completes the TLF programming and generates the output, the step notification goes to the QC programmer (programmer 2) assigned to the step named *QC process* (Step 2). The QC programmer can then complete the QC process and once the TLF is QC passed, then the step notification is sent to the statistician to begin review of the TLF. Looks like we are set. But what happens if the QC programmer needs to communicate any comments to the main programmer and the main programmer needs to update the TLF output? This must be done offline, away from the SCE, since the QC interactions cannot be captured in this workflow. But can we add more steps and capture this process?

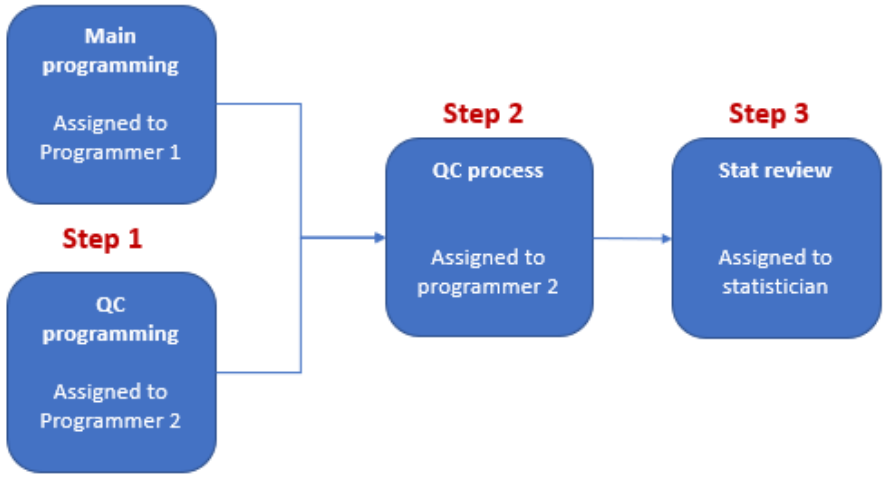

Figure 7: TLF Workflow Version 4

WORKFLOW VERSION 5

Version 4 of the workflow shown in Figure 7 appears to capture the TLF programming workflow to reasonable accuracy, enough for most audit trail purposes. To make things complete and make project management easier, however, the workflow can be upgraded further by adding a feedback loop from the QC process to the first step. Any feedback from the QC process can be used to reset the state of the first step to *Programming in progress* so that the programming updates made during the QC process can be captured in the workflow and will be visible to project managers tracking the project status. This feedback loop is illustrated in Figure 8.

For most part, version 5 shown in Figure 8 has the potential to capture the important aspects of the programming process while also allowing for programmers to mark work items complete as and when they complete programming activities. Figure 9, Figure 10, Figure 11 and Figure 12 illustrate a set of step state diagrams that the Version 5 workflow would go through for a single TLF artefact. The state of each step in each state diagram is shown in an oval within the step. Most SCEs provide a comprehensive and possibly visual view of the status of all work items in a project for the team lead or project manager to get a better view of the state of the deliverable.

Additional detail could be added to Version 5 to make it more comprehensive. If log checks and version control check-in of a program is to be recorded, then that would be another step. If code review is

needed, another step and another loop is needed and so on. There is a however, a limit to the usefulness of adding too many finer details in a workflow since it also complicates a reasonably simple process and adds the overhead of too many step notifications and actions to perform for programmers who would rather spend a better chunk of their time writing code.

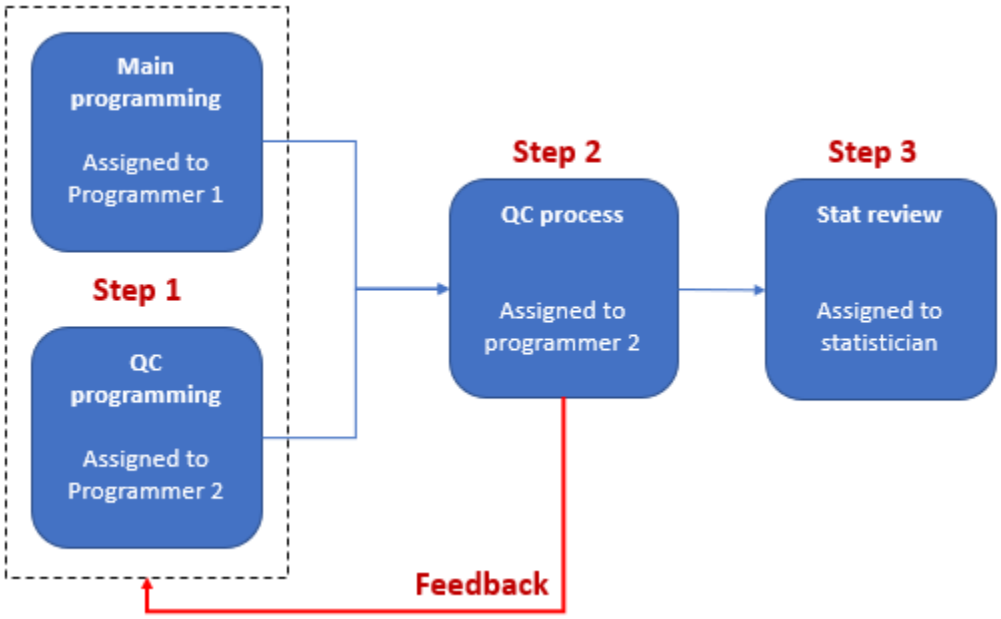

Figure 8: TLF Workflow Version 5: Adding the feedback loop

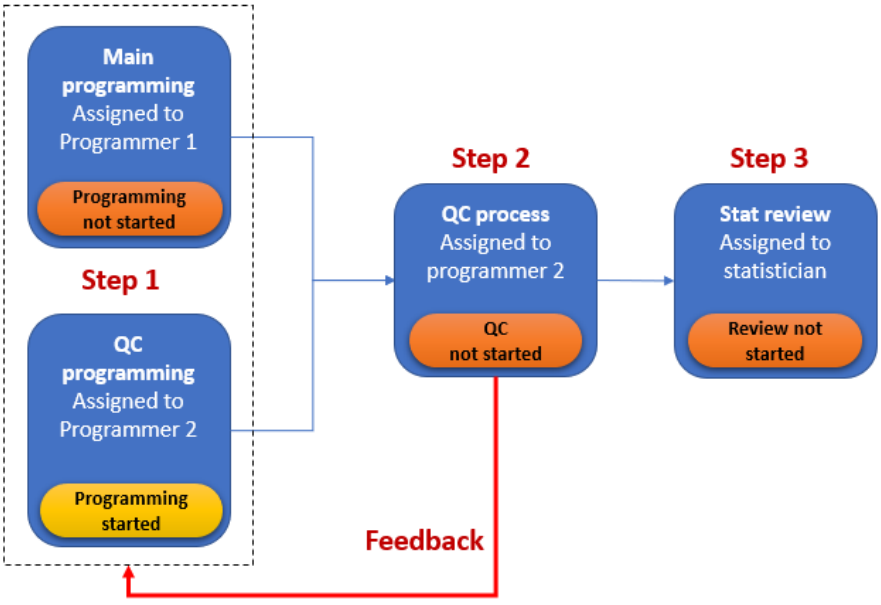

Figure 9: An early state diagram of version 5 of the TLF Workflow

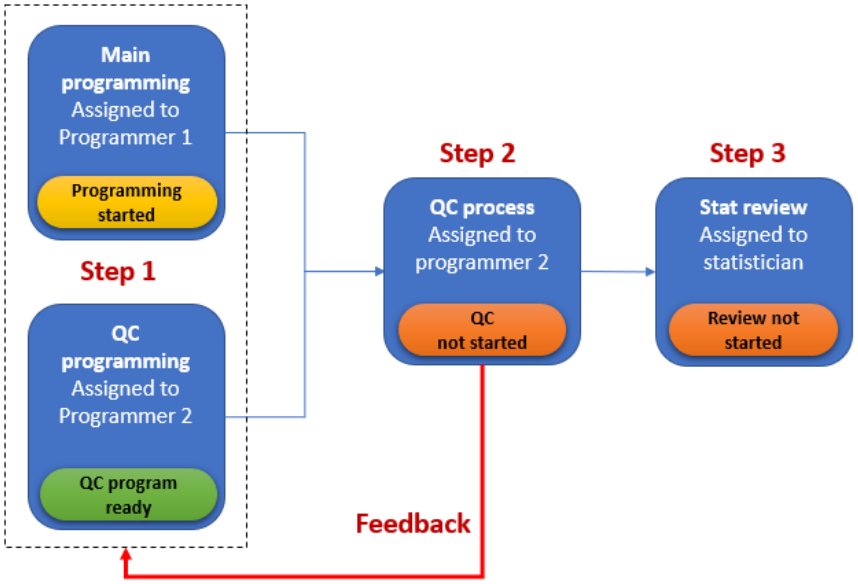

Figure 10: A mid-process state diagram for the Version 5 TLF Workflow

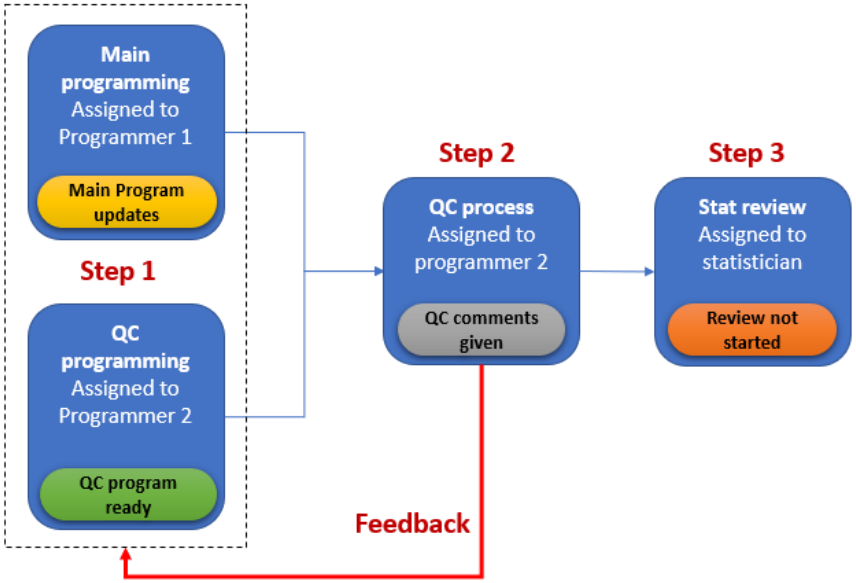

Figure 11: A state diagram for the Version 5 TLF workflow illustrating the QC feedback

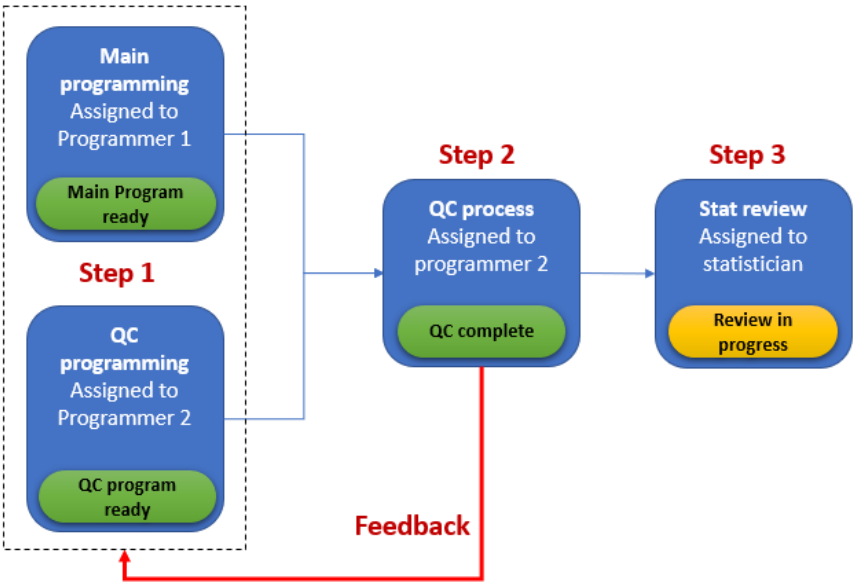

Figure 12: A state diagram for the Version 5 workflow showing the completion of programming and start of review

ADDITIONAL WORKFLOW TEMPLATES

Like the workflow presented above for TLF programming, SCEs need additional workflows for other items that are part of the deliverable such as datasets, define.xml etc. depending on the scope of the study and the deliverable. These workflows could be similar to the TLF programming workflow discussed above or very much different depending on each organization's SOPs. What is more important here is how these workflows are inter-connected to one another. When the SDTM workflows are completed, ADaM workflows should be triggered and similarly, when ADaM workflows are completed, TLF workflows need to be triggered. This will always ensure traceability within the system. Configuring such top level workflows consisting of multiple smaller workflows is no simple task considering the amount of foresight and analysis needed to configure something that can be frozen only after a certain amount of trial and error and real-time project usage.

CONCLUSION

SCEs are the next big wave all set to upgrade decade-old processes in statistical programming to improve productivity, subsequently leading to lower time-to-market for drugs. While many other features of new-age SCEs are close on the heels of what our conventional SCEs already offer, one of the features that stand out is that of workflows within SCEs. Workflows have the potential to vastly improve traceability and provide detailed audit trails while also making project management for study deliverables a seamless activity that is integral to the SCE.

SCEs with customizable workflows are preferred over SCEs that come with hard-wired workflows for many reasons not limited to - better fit for disparate organizational SOPs, allowing for easier upgrades as processes evolve and, allowing for tweaks to suit slightly differing processes across teams within the same organization. In this paper, a hypothetical SCE with customizable workflows was considered for configuring a TLF programming workflow to be used to create various TLFs needed for a deliverable. It was demonstrated how a seemingly simple TLF programming workflow can be configured in many ways using customizable workflows. Using a simple state diagram, a not-so-complex version of the workflow was shown to suit the needs. There is no perfect way to configure such workflows and often, what works for one organization or team may not work for another. Setting up of workflows takes time and an in-depth analysis of the underlying processes being mapped. Customizing and setup of workflows can be a long-drawn process with a lot of lessons learnt along the way.

It is very easy to see that programming workflows can vastly improve traceability, enhance audit trails in the system and significantly reduce the scope for human error in statistical programming deliverables. The subsequent advantage for project managers is fewer spreadsheets and implicit project tracking which is a welcome relief. But it is important to keep in mind the overhead needed for setting up programming workflows in SCEs for large organizations since project tracking becomes more than just starting with an SOP and an empty spreadsheet template. Too many steps in a workflow can impede productivity and stifle natural process flow whereas too few steps can lead to poor process capture resulting in a very diluted audit trail. It is also important to keep in mind how programming workflows can affect the way programmers function and the kind of resistance programming teams may have to work with elaborate workflow setups.

There is also no one-size-fits-all solution for setting up a catalog of workflows for capturing the end to end processes in statistical programming starting with the raw data until the define.xml. There are plenty of ways to accomplish the same result and anything works if end to end traceability is possible and project tracking is reasonably uncomplicated.

In summary, modern SCEs are paving the way forward to improve traceability and audit keeping and programming workflows are the need of the hour for improved project tracking in these newer systems. For multiple reasons, customizable workflows are better than hard-wired ones. Customizing workflows for programming processes is very much possible but entails significant planning and plenty of trial-and-error and fine tuning before a suitable solution can be set in place for regular usage by programming teams. It is important to stop at the right level of detail in programming workflows since most of the time, a simple solution is quick and better than a complex and tedious one.

REFERENCES

[1] Hopkins, Alan et. al. 2010. “Statistical Computing Environments and the Practice of Statistics in the Biopharmaceutical Industry.” Drug Information Journal, Vol. 44.

[2] PhUSE Working Group. April 2016. “State of the SCE, A White Paper by the PhUSE CSS Emerging Trends and Technologies Working Group Statistical Computing Environments (SCE) Project.”

[3] Peng Yang, et. al., 2013. “Using Workflows and Metadata Information to Standardize Business Processes in Pharmaceutical Programming.” PharmaSUG 2013. Chicago, USA.

[4] Tyagrajan Swaminathan, et. al., 2018. “Regulatory Compliance in the Digital Age – A cloud-based statistical computing environment (SCE) to the rescue.” PharmaSUG 2018. Seattle, USA.

[5] Yeqian Gu. 2017. “How Process Flow Standardized our Process.” PharmaSUG 2017. China.

CONTACT INFORMATION

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

Tyagrajan Swaminathan
Ephicacy Lifescience Analytics
tyagrajan.swaminathan@ephicacy.in
www.ephicacy.com

Sridhar Vijendra
Ephicacy Lifescience Analytics
sridhar.vijendra@ephicacy.in
www.ephicacy.com