PharmaSUG 2014 - Paper MS02

Building Better Programming Teams with Situational Exposure Training

Elizabeth Reinbolt, MS, DataCeutics, Inc., Lincoln, NE
Steven Kirby, JD, MS, Shire, Chesterbrook, PA

PROGRAMMING IS A TEAM SPORT; GET YOUR TEAM READY TO PLAY

A programmer's life in the broadly collaborative, timeline driven world of clinical research requires the ability to handle a wide variety of situations. While it is (relatively) simple to find good materials to teach a programmer specific SAS skills such as ODS or macro processing, the most challenging issues programmers (and programming teams) often face are not a lack of technical skills but difficulty thriving in unfamiliar situations.

Although there are a host of companies ready to share general strategies for common workplace issues such as time or stress management, the authors suggest that, just as sports teams mimic game situations in practice, one useful way to help programmers and programming teams thrive in new situations is to expose them to those situations through role playing within the group.

The situations that adversely affect programmers tend to cluster in areas of experience that are not commonly encountered in college or associated technical training. As much of our work lives were spent managing the efforts of programmers and programming teams at CROs, we will focus on a few representative situations related to processing information from multiple clinical trials at the general direction of a variety of external clients and subject to their review.

YOU PITCH GREAT BUT WE NEED YOU IN LEFT FIELD

One common issue programmers face is how priorities and tasks can shift rapidly due to outside forces. This can, for instance, occur when clinical trials are expanded, canceled or put on hold and can also be encountered when internal workloads are rebalanced. While unexpectedly shifting tasks and priorities is naturally disruptive and inefficient, such adjustments can have a dramatic effect on the production of some programmers. When a programmer is found to struggle far more than average when asked to shift priories or tasks, a bit of guided exposure may help the programmer discover effective coping strategies.

One useful way to expose a programmer to shifting tasks and priorities is to have the programmer touch base with a manager each morning to get the task for the day, with the task assignments shifting far more than they typically would. A few moments of that interaction would be used to discuss effective ways to maintain overall efficiency and continuity of any existing tasks. This type of experience gives the programmer an opportunity to discover effective and palatable approaches to shifting priorities; and weekly follow-up can allow time for a guided evaluation of what worked and what did not.

Another way to expose a programmer to shifting tasks and priorities is to have two (or more) programmers regularly switch assignments, say once a week for a month, asking that the programmers meet to share practical details with each switch. This approach builds in quite a bit of change and variety and may also help foster supportive relationships within the group. The natural dynamics of the situation can also help emphasize the possibility that a task may not be completed before a priority shift and the associated need to keep work organized and accessible.

Focusing the programmer's attention on a wide variety of tasks can also foster a greater appreciation for all the different moving parts that have to work together to generate a complex deliverable. As an added benefit, that broader awareness can help a programmer look at the work as part of a group effort as opposed to as an isolated set of tasks. Playing well is good; but a team win is what matters.

THE SCOUTING REPORT IS GREAT BUT WE PLAYED THEM YESTERDAY

Programmers need to thrive in an environment where work has to be done accurately and as quickly as possible. This type of environment is naturally stressful and, without some expectation control and workload management savvy, programmers can unintentionally add to the natural stress that comes with the job. Programmers who are too overconfident or too eager to please can easily find themselves overwhelmed with projects; programmers who are too bluntly resistant tend to gain little benefit, and can compromise relationships within the group and between groups.

One way to manufacture safe practice situations related to timeline and workload management is to leverage granular completion status milestones. The milestones used can be part of standard processes or manufactured for training. For example, have a programmer estimate and then track progress using a system that has an individual milestone for review of each table or generation of each unique table. During the course of the exercise, regular meetings could be held to evaluate how well the initial estimate is holding up and to investigate other options that may produce better estimates.

By forcing programmers to actively discuss and process questions about a number of sub-deliverables and milestones (that would typically not be tracked by other groups) they can get valuable experience managing timelines and expectations with limited risk. And as an added bonus, that focus on very granular timelines and load balancing can help foster a better understanding of the work flow as a whole.

IT'S NOT YOUR BALL AND YOU CAN'T TAKE IT HOME

Programmers need to work together on a single study in a wide variety of ways. And they need to do it collaboratively and courteously. But just as with group projects in school, working in a group sits awkwardly with some programmers. All programmers can learn to more effectively coordinate their efforts to close out high priority deliverables, and exposing them to situations that artificially put a premium on coordinated effort can be an effective way to help them learn this critical skill.

When two or more programmers work together to complete a set of deliverables, the work is typically split up into more or less discrete chunks such as one programmer does the AE and Concomitant Medication analysis tables while another does Clinical Labs and Vital Signs. A couple practical advantages of such an approach are that the programmers are unlikely to get in each other's way and the work is split into (roughly) analysis type (frequency vs summary; BDS vs ADAE/Other); but those advantages also tend to make it so they have a limited need to cooperate.

Consider how the need for cooperation increases if the work is split with each programmer doing every other table in the SAP list or if one programmer generates ADAE and ADCM, the other generates the associated tables and then they switch roles for ADLB, ADVS and the vital signs and lab tables.

When a need for greater cooperation is explicitly built into the workflow, the situation encourages programmers to find productive ways to work together. And as an added bonus, cooperative training situations naturally emphasize how taking the time to have a solid plan of attack and well-documented programs can simplify coordination and take some of the burden off e-mails, phone calls and other less structured, informal communication.

THE GAME IS STARTING AND WE CAN'T READ THE LINE-UP

Some programmers are naturally tidy, some program with abandon and some wake up at night in a panic at the thought of having a misspelled word in a comment. While all successful programmers learn to adhere to all applicable local rules on program organization, documentation and naming, there is often enough flexibility in those rules that it is possible for a programmer (perhaps overly focused on short term efficiency) to have a programming area with enough clutter that anyone asked to take over a project (due to shifting priorities or illness or job changes) would be obligated to spend considerably more time figuring out what was going on than optimum.

As it is somehow more palatable to clean up your own messes than someone else's, creating situations where programmers need to (in some sense) clean up after one another tends to be a simple and effective way to teach the importance of clean and organized programming work areas.

To expose programmers to the perils of clutter at a broad level, it can be effective to ask a programmer to verify that all the programming folders associated with a study are ready to be considered final for a downstream purpose such as archiving. That type of review typically leads to identification of excess content (such as unused or replaced programs) and tends to highlight any content whose purpose is not easily understood. That kind of housekeeping task can motivate the programmer to maintain a personal programming space that is easily accessible to other programmers. It is even possible that this exercise may help a programmer understand that programs are not just for personal use.

Another useful way to emphasize the importance of clean programming areas is to create situations where programmers need to jump into an unfamiliar programming area and modify or explain the content. For example, take a program with all annotations removed and have the programmer provide a usefully detailed annotation of each section of code that covers what the code does and (if relevant) why a specific programming approach was chosen. Once that is complete, have them add header information that is informative and consistent with local rules. That task will naturally emphasize how useful well-formed comments can be and can help them better understand the practical need to keep programs clean and well documented. And as an added bonus, this exercise will help programmers be better able to document and explain code.

YOU'VE GOT A SWEET SWING, BUT YOU CAN'T HIT A CURVEBALL

Being able to operate efficiently typically leads to reliance on standard (or otherwise suitable) programs; and successful programmers typically are very good at leveraging standard code to quickly generate accurate output.

Given sufficient time pressure, programmers can be tempted to assume that a program that works correctly under most circumstances always works perfectly and can (on occasion) suggest that a wrong number in a table is caused by a program and not a programmer. That is never the case.

If programming could be fully reduced to dumping in data and pushing a button, companies could save considerable expense by hiring chickens to peck colored buttons as opposed to experts with the ability to reason through scientific and technical challenges. Thankfully (for programmers at least) being able to think through those scientific and technical challenges is an unavoidable part of the drug discovery process. Training that forces a programmer to have a broad, deep understanding of how the standard programs and macros work is critical.

One way to encourage understanding of standard code is to have a programmer repurpose a program for another use. Assume that your company has one program to generate vital signs summaries. Asking a programmer to adjust it to generate ECG summaries and to document the adjustments in plain language forces the programmer to understand the code. Another way to encourage understanding of standard code is to provide the programmer with a new task and a limited selection of (standard) starting programs, none of which are perfectly suited to the task. Have the programmer select which program is the best starting point and explain the reasons why the selection was made. If a better choice is available, then the manager can share why the other program is a better place to start; if the best choice is selected, the manager can ask that the programmer make and document needed adjustments. The manager would then review the final code to ensure that the modifications make sense and nicely lead to the desired outcome.

As an added bonus, such in-depth encounters with standard code may help trigger a better understanding of how various data contingencies need to be accounted for, of how the code could be further optimized, and of the challenges faced when generating standard code.

READY FOR THE PLAYOFFS

Programmers do not work in a vacuum. New situations tend to require new skills; and existing skills can be hard to access in unfamiliar surroundings. While there are many serviceable training options available for general work and SAS skills, it is important to have training that specifically targets the local environment in which programmers need to thrive. Situational exposure training can help programmers fully leverage their training and talent and can help them build a broader, more durable skill set that is ready for all the challenges in their local environment.

ACKNOWLEDGMENTS

The authors would like to thank DataCeutics, Inc. and Shire for supporting programmers and our presentation of this paper. And we would like to specially thank Dan Rezac, one of the first (and certainly the most memorable) participants in our situational training exercises.

CONTACT INFORMATION

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

Name: Beth Reinbolt
Enterprise: DataCeutics, Inc.
Work Phone: 610 970 2333| Ext 3123
E-mail: reinbolte@dataceutics.com
Web: http://www.dataceutics.com/
Twitter: https://twitter.com/DataCeutics

Name: Steve Kirby
Enterprise: Shire
Work Phone: (610) 321-2849
E-mail: skirby@shire.com
Web: www.shire.com