Paper 188-2013

Communicating Standards: A Code Review Experience

David Scocca, Rho, Inc., Chapel Hill NC

ABSTRACT

We need ways to pass along good programming practices. All but the smallest companies will have programmers with varying levels of tenure and experience. Standards and best practices change, but in a deadline-driven world, we re-use old programs with minimal revision. Programmers develop habits and can be slow to incorporate new approaches that might simplify code or improve performance.

We developed and rolled out an in-house code review process to address these issues. This paper reports our strategy for promoting and performing the reviews and describes the results.

GENESIS

Rho is a full-service contract research organization, supporting both commercial clinical trials and federal research projects. A few years ago a corporate organizational shift made our company more project-focused but dissolved a single statistical programming department. SAS® programmers were spread throughout the company (organizationally) and around the building (physically). To maintain relationships and communication between employees performing similar tasks we created internal groups called Communities of Expertise (COEs), including the Statistical Programming COE.

The Statistical Programming COE organized a number of subcommittees including one charged with “Code Quality”. The programmers in this group discussed the kinds of code we were seeing and noted several common issues, many of which resulted from the common practice of starting work with a copy of an existing program:

- Outdated routines were still called in current programs
- Headers were often not updated completely and contained outdated or incorrect information
- Some code did not follow the current company programming standards guide
- Changes and modifications were inadequately commented

We decided that we could address some of these situations by providing feedback to programmers through code review.

PRINCIPLES

From the outset, we agreed on some basic principles:

- The goal of code review is educational, not punitive. We assume that programmers want to learn and get better, so we will draw their attention to particular issues we find during review. We do not want programmers to fear code review.
- Code should be reviewed by active programmers, not by managers. From the COE membership, we were able to identify a number of experienced programmers, including some members of the subcommittee, who could serve as our reviewers.
- Code review will be treated and tracked as a training activity and recorded as professional development, both for programmers and for code reviewers.
- Code review is for everyone whose job routinely involves programming, whether classified as a statistical programmer or as a biostatistician.
- There will be no long-term record keeping of the code reviewed or the issues found. We will only track whether the programmer had a code review.

We tried to develop a process that would be consistent with these principles.

We also decided that we wanted to have each programmer reviewed at least once a year, with new hires reviewed an additional time within the first year. We had enough reviewers that this would require each person to perform about one code review each month.

STANDARDS

Our corporate programming standards are specified in a few controlled “Guide” documents; these had last been updated a couple of years before we got started. The guides were our starting point in developing standards for code review. We started with a list of the specific standards, which we integrated into a single document, grouped by topic and listed in a logical order.

For each rule taken from the standards we added some notes to expand on or explain the rule, to provide additional information for the programmer or code reviewer, and in some cases to provide links to external sources for more information. The result was a Code Review Checklist document that reviewers could reference and that programmers could use to recognize and resolve issues in code.

We realized that there were cases in which previous versions of the standards called for practices which are no longer followed and in some cases actively discouraged. One obvious example is code that was developed in SAS version 6 and had statements to save the contents of the program editor window. Issues like these were not incorporated into the formal checklist, but reviewers were encouraged to alert programmers if these legacy statements were encountered.

A final issue was that of “inherited” code. Since many programs start as copies of existing programs, issues were frequently found in the parts of the code that the most recent programmers had not written or modified. We agreed on some general points of philosophy for inherited code:

- When you base a new program on an existing program, you take responsibility for the existing code you incorporate.
- Your header block should indicate that you created the program; while it may include a pointer to another study from which code was copied, the inherited programmer history should be removed.
- We recognize that deadline pressures do not always allow time to go through inherited code and bring it up to date. For the learning process of code review, we will examine and provide feedback on entire programs.
- At an absolute minimum, header block information must be reviewed and updated when copying programs. Even under extreme time pressure, it is better to delete the old information and leave it blank than to have outdated or incorrect information in the header block.

PROCESS

At this point we also outlined a process by which reviews were performed. We chose to track code reviews with an instance of the ticketing system used by our IT support group and other groups in the company. This allowed us to track when code reviews were scheduled and when and by whom the reviews were completed.

The review process goes as follows:

1. A few programmers at a time are notified that it is time for a code review, and are asked to send a request for review to the ticketing system. The request should include a pointer to about five projects the programmer has worked on and indicate the parts of the project on which the programmer worked.
2. When the programmer sends a request to the ticketing system, all the reviewers are notified that a new ticket has been created. A reviewer then takes responsibility for the code review by acquiring the code review ticket in the ticketing system.
3. The reviewer goes to the project directories indicated by the programmer and looks for three to five programs to review. The headers are used to identify the programmer’s involvement. The reviewer’s goal is to select an assortment of programs that provide a good basis for review with an eye to having a diverse sample and choosing code where the programmer’s role is relatively significant.
4. The reviewer copies the programs from the production location and uses the checklist to review the code. Findings may be recorded on a paper copy of the code, in a separate document, or as inline comments in the copy of the code.
5. The reviewer schedules a one-on-one meeting with the programmer to discuss the code review. After meeting the programmer, the reviewer closes the ticket and the programmer records a code review in the training tracking system.

We also sought support for the process from management. During the time we were developing the code review proposal, one of our original subcommittee members became the acting head of one of Rho’s main divisions, called “sectors”. With her help, we received approval to start the process by reviewing the programmers in her sector.

PILOT CODE REVIEW

We did a pilot test of the code review in two phases. First, each member of the subcommittee gave a handful of programs to another subcommittee member for review. We did not use the ticketing system, and we used the subcommittee members instead of the usual pool of reviewers. During the course of these reviews we made a few more updates to the Code Review Checklist document to make it clearer and more comprehensive.

At this time, we also drafted the message to be sent to programmers when it was time for code review, explaining how to request a review through the ticketing system.

After completing this first pilot stage, we had a meeting to train our group of experienced programmers in the code review process. We introduced the checklist to the reviewers who had not been part of the subcommittee and we provided training on the use of the ticketing system, which our SAS programmers had not previously used. We then asked the code reviewers to go through the full process of reviewing one another's code. Immediately after this meeting, we sent the code review message to all the reviewers. Each reviewer then sent in a list of projects to the ticketing system, which created a ticket and notified the pool of reviewers. Then, each reviewer took responsibility for one other reviewer's ticket, performed the code review, and then closed the ticket.

This pilot stage was fairly uneventful; reviewers were able to learn about the ticketing system and each determine a comfortable approach to recording code review findings.

CODE REVIEW

After the pilot phase, we rolled out the full code review process with an announcement on our company intranet. The code review message was rolled out to approximately a third of the programmers in the company over the course of two months. The majority of those programmers sent a message to request a code review and went through the review process. At the time, most code reviews were completed within a month of the time the request was sent. Each code review took two or three hours; the reviewer would gather the programmer's code, review it, and then have a meeting with the programmer to discuss the findings.

In most cases, the immediate reaction from the programmers was positive.

There was a delay of several months before the next batch of programmers were notified that it was time for their code reviews. When those notifications went out, things did not go as smoothly. Requests for review were sent, but the resulting tickets languished in the system. Although all the reviewers were notified of ticket creation, there was no particular system in place to assign each ticket to a reviewer—and under heavier workloads, many reviewers felt they did not have the time to commit to taking a code review ticket. For the time being, no more code review notifications will be sent out until the current queue of tickets has been eliminated.

REACTIONS AND RESULTS

Programmers who went through the first round of code review were generally positive about the experiences. Among the observations:

- The code review was more useful to one programmer than a more general training presentation because it applied directly and specifically to code he had written.
- In at least one case, the process led to a mentor relationship between a programmer and the reviewer, who was helpful well after the actual code review was done.
- For a programmer who was new to the clinical trials industry, the code review provided exposure to industry and company conventions and standards.
- The review process provided an opportunity, without the time constraints of project deadlines, to ask a more experienced programmer about specific issues that had been encountered when writing the code.
- For a more experienced programmer, the review served as a reminder that other people will see and use his program code, and helped uncover “blind spots” where he hadn't realized he wasn’t following the programming standards document.
- A reviewer indicated that the role made her feel that her code should set an example and made her more conscious of the guidelines while programming.

One programmer expressed concern about the potential for discomfort: “…it’s hard not to be defensive when someone finds fault with your work (even when justified), and it’s never comfortable to be confronted with instances where someone realizes you cut corners or missed something….” However, this programmer also said his own code review experience had been a positive one, describing it as “very collegial”.

A reviewer described the code review process as “time-intensive,” noting that the same experienced programmers who serve as reviewers are in high demand for billable project work.

CONCLUSION

The roll-out of the code review was promising. The feedback from the programmers and reviewers was generally positive, and initially code review requests were being processed within a month. However, it appears that we may have started doing reviews at a time when the company-wide demand for programming was relatively low. A return to higher demand for programming has left our reviewers with less time to do code review, and the pile of open tickets has grown higher and more intimidating. In addition, the several-month delay before the second round of programmer notifications got reviewers out of the habit of incorporating code review into their regular routines.

From the beginning we had management support for implementing the code review process as a form of training, but neither management nor the reviewers had planned on accounting for code review when discussing resourcing and when making project assignments and timelines. Reviving the process will probably require a more conscious effort to carve out time for the experienced programmers to serve as reviewers.

Although the reviews eventually ground to a halt, we still feel that the process thus far has been beneficial. We had an opportunity to revise and update our programming standards, and the documents we created are still useful outside the context of formal reviews. The responses from programmers who were reviewed suggests code review was beneficial, even if it turns out to have been a one-time activity rather than an ongoing program.

While not an unqualified success, the code review implementation was a net positive and has left us better positioned to continue to communicate programming standards from here on.

ACKNOWLEDGEMENTS

I would like to thank all the programmers at Rho for being a great team to work with, and Rho management for supporting the code review implementation and the writing of this paper. Particular thanks are due to the Coding Quality subcommittee of the Statistical Programming COE; to Shane Rosanbalm, who headed the subcommittee and took the lead in making code review happen; and to the programmers and reviewers who gave me feedback on the process.

CONTACT INFORMATION

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

David Scocca
Rho, Inc.
6330 Quadrangle Drive
Chapel Hill, NC 27517
919-595-6274
Dave_Scocca@RhoWorld.com