

The Making of a SAS/AF Programmer

John T. Stokes, JJT Consulting, Inc.

We will discuss what we have learned during the past two years in building a team of SAS/AF Frame programmers: which traits and qualities make it and which ones break it. We will also discuss our formula for building a successful team of Frame programmers, and how we trained these individuals in the latest SAS technology.

The team began with two experienced programmers with an in-depth knowledge of SAS. From this basis, we have evolved into a team of top notch Frame/SCL developers. Along the way we have made a few mistakes, but largely we have had uncanny success in adding staff. We started by looking for SAS/AF/SCL experienced programmers. There aren't many. Many who claimed to be, weren't. Consequently we developed a two hour test to weed out those not having the necessary experience. No one passes the test, but it does determine if the programmer has a clue about AF/SCL. In fact, we once had an applicant travel over 200 miles to take the test. He left without saying a word. We found out when he called from home and said it wasn't the job he was looking for.

Early on we were fortunate to add a couple of programmers with SAS, but no Frame experience. In order to share what little we knew and what we were learning, we needed to build a kinesis. So we worked elbow-to-elbow — we literally had five people in a 11' x 14' room. In truth it was probably the other way around. All we had was one small room for five of us, and it forced us to share our experiences and our knowledge. In retrospect, this confined situation was probably crucial to our development as a team.

As we needed more staff, it became apparent we weren't going to find ready-made Frame/SCL programmers. Our experience thus far in recruiting showed us that SAS experience did not necessarily mean an aptitude for Frame/SCL programming. So we started looking for top quality programmers who had certain attitude traits and experience.

We determined that there is a distinct difference in SAS Datasstep programming, which is more data (dataset) driven, than tradition procedural programming. Additionally, we determined that SCL was much closer to procedural programming than to Datasstep programming. So, we searched for traditional procedural programmers with some kind of GUI programming experience, such as Motif or Visual Basic. As we added people with this background to our team, we found that their assimilation was rapid. They were generally able to pick up SAS quickly, and were marginally productive within 2 to 4 weeks. We have found that, while SAS experience can be an asset, it can also be a hindrance. We had one or two programmers who could not make the transition from solving all problems in Base SAS to considering the SCL alternative when appropriate.

In addition to the type of programming experience we were looking for, we determined that there was a special attitude we wanted our AF/SCL programmers to have. We wanted them to be go-getters, who would be willing to experiment, capable of working on their own, and able to ferret out information

from obscure places. Because a lot of what is in AF and SCL is not overwhelmingly well documented, programmers must have an ability to be unusually creative in order to get a program to work properly, or to find a means of working around things that the Institute didn't consider when they put SAS together. Consequently, we also needed somebody with a willingness to learn new things, and positive attitude toward innovation. Some programmers are basically entrenched in what they do and are convinced that their way is the best way to do something, and are not willing to open their mind to new things. They believe SAS is not a real programming language, that it's not a good development environment. We strive to avoid these kinds of prejudices, and seek instead a very positive attitude about the possibility of learning and doing things differently. Additionally, we need programmers with initiative, who can work without a lot of guidance.

We also wanted individuals who were risk-takers. People who have worked for companies where they were in salaried positions often don't really look at the future, growth, and the possibilities available. Consequently, they might have a hard time making the transition to the consulting environment. We had to make it very clear to applicants that we are dependent upon our clients, on developing that clientele and expanding it. They need to understand that there is some risk involved in working in a consulting environment. Our consultants need to be able to accept this risk as well as be able to tolerate some of the problems inherent in a client-based consulting environment. This means that you've got to please the clients in spite of the fact that their demands may not necessarily make sense from a programming perspective. Nonetheless, these kinds of frustrations must be tolerated, since the clients are the ones that ultimately pay the bills.

Educational background is not as important in our hiring decisions as a person's life experiences. We have people with Bachelors, Masters, and even Doctorate degrees, but we also have auto mechanics on the staff and other personnel without college degrees. An applicant's life experience tells us if the person is truly a problem solver, or if they're just a follower.

In addition to the successful hiring of talented staff, I believe that the management philosophy Motorola, our major client, operates under has played a role in our success to this point. They have a rare management/development philosophy which is oriented towards innovation and they are always looking for us to do new and different things. They want their company to be on the cutting edge of technology, and they want us to be there with the kind of things we provide as analysis tools for their fab plant. However, they've also been extremely demanding: they want everything yesterday, even if they didn't think of it until today. This means that we have to find way to be both as efficient as possible and as organized as possible, which is a continuous struggle.

We call our technical development life-cycle a "prototype life-cycle." Our client will come up with an idea which we'll then prototype and show to them. Usually this first prototype will be extensively revised based on client feedback. Then maybe by the second prototype, we'll actually involve the users to get their comments and feedback, and revise the prototype again. Our products evolve

this way until they're fairly well established. Then we go into a more conventional requirements, update, and modify type of life cycle. Initial life-cycles, however, are all of a prototyping nature. This has enabled us to do a lot of experimentation, a lot learning, and it also is what requires that our people be able to live up to the kind of demands that are placed upon them.

While hiring the right people is critical to success, keeping them on board is just as important. Essentially, in a consulting business, your people are absolutely the most important resource you have, and we think this is true in almost any aspect of software development. Your software is only as good as the people you have developing it. So, you need to have top people if you expect to get the best results. Accordingly, you have to be able to pay for the top people, since you basically get what you pay for. If you're able to afford the best, then you'll get the best results. We think in the long term you're better off with one good person than two mediocre people. Or on a larger scale, you're better off with 5 good people than 15 mediocre people because the 5 give you back-up and redundancy. They also give you a team and collegiateness. However, they have to be able to work together. Compensation involves more than a good pay rate; the benefits a company offers is also an important consideration for most employees. For a very small company we offer excellent benefits in terms of health care and vacation/sick policy. We also offer employer-paid health insurance for our staff and their families.

Another thing we think is crucial, and something that a lot companies make mistakes with, is working hours. We expect people to work reasonable hours. Basically, we have work the usual 40 hour week, without a lot of overtime. We don't find that we need to if we plan properly. Typically we get our projects done in the time frame in which they are scheduled. Occasionally, we do work overtime, and a lot of people work at home. We provided the ability for them to work with the products we're working on at their home if they choose, since a lot of our people like to go in and tinker for an hour or two at night. This is usually done for special projects, new things, or basic research so that they can learn more about what they want to do, and then use it in some of the things that we're developing. So, the home platform is another form of benefit. It also allows them to work at home when they need to take a day off for personal reasons. This kind of telecommuting is something we find very beneficial. We couldn't keep the good people that we have without these benefits.

Once personnel are hired and well-compensated to perform their tasks, training becomes the key to making them successful. Probably the most important thing is what we call elbow-to-elbow training. This literally involves packing people into a small area to work together. Although caution needs to be exercised in using this approach, since some real difficulties can arise. One of the most positive aspects of this approach is that everyone is immediately available when someone has a question. Asking a question of the group may result in two or three answers, and the combination of answers may provide the best solution to the question. The interchange of ideas in that environment is absolutely phenomenal, and helps us bring people up to speed in the time frame that we do.

For more formal training, we've found that SAS TUTOR to be very beneficial. Because our people are self-guided, they can easily acquire a good basic understanding of SAS by putting themselves through the tutorial. We also use SAS training. Typically, we schedule SAS training after a person has been on-board a little while and has gotten familiar with SAS. As a result, the class is used more to round out their knowledge rather than to give them a basic introduction, and they tend to get more out of the class than if they go into it without knowing anything about SAS. Our people learn more about SAS on the job, often through trial-and-error, than they do through formal training. Additionally, we promote peer review and peer criticism. People can't have thin skin around here, you have to be willing to have people look at your work and evaluate it. Someone may tell you that there's a better way to do it, and that you should think about doing it differently. About half the time, the person will come back to their evaluating peer and agree. At other times s/he will come back and explain that the work was done the way it was for a particular reason, and the evaluating peer will agree with this assessment.

In conclusion, what it takes to make a good AF programmer is not necessarily the same as what it takes to make a good base SAS programmer. We think a good AF programmer can be a good Base SAS programmer, but we think someone with a lot of Base SAS experience and no other procedural programming experience may not be able to make the transition well. We have found that you can take C, MOTIF, Visual Basic, or Windows programmers and turn them into AF/Frame programmers much more easily than you can turn traditional Base SAS programmers into AF/Frame programmers. The thought processes and the things that go into a GUI development environment is simply more similar to the AF and Frame environments. Also, something about procedural programmers and procedural languages and SCL just makes doing SCL a lot easier: picking up the SAS part is easier than teaching somebody to be a procedural programmer. Again, you've got to look for and hire talented and flexible thinkers in order to create a successful team of AF/Frame programmers. We think we've successfully done that.