Jan Steinman Barbara Yates

Mentoring

Does anyone out there think the word mentor is overused and abused in our industry? We are often dismayed at the ads we see from companies trying to recruit Smalltalk "mentors." Judging by some of these ads, it appears that anyone with a year or more of Smalltalk experience is deemed capable of mentoring. We want to tell you that it just ain't so!

WHAT A MENTOR DOES

A dictionary defines a mentor as "a wise counsellor (one who advises and warns)."[1] A good mentor gets great personal satisfaction from helping others to learn and grow. They can be compared to good teachers; they are not Smalltalk gurus who take over your keyboard and write your code for you. Mentors need patience and excellent communication skills, and must know the technical content of the subject they are mentoring. Mentoring is a one-on-one activity, so it is important that there be a "personality click" for the mentoring to be successful.

A good mentor gets great personal satisfaction from helping others to learn and grow.

When managers want to recruit one or more mentors for a team, it's important that they check references. When talking to a manager of a team the mentor has worked with, it would be useful to ask to speak with some of the team members about their experiences. Given the time crunch most managers are in, there might be little time for checking references. At a minimum, here are some questions to ask the candidate–mentor :

- How would you handle a team member who doesn't want to ask for your help?
- How would you deal with a team member who asks very low-level questions most of the time?
- What aspect of mentoring do you like most?
- What percentage of your time would you expect to spend programming in Smalltalk vs. mentoring?
- What is the hardest problem you've had to deal with as a mentor and how did you solve it?

Mentoring is a lot more difficult (and sometimes a lot less fun, to be honest) than programming in Smalltalk. Every mentor can probably tell a few horror stories about his or her experiences. We'll briefly share a couple with you.

We were called upon to mentor a small team of novice Smalltalkers. We were not supposed to mentor full time, having major architecture and development responsibilities as well. We found that the proportion of mentoring to other tasks would vary in different stages of the project, so that when people were deeply into coding, our mentoring would only be about 30% of our work load. (More mentoring occurs at the early stages, and then it gradually tapers off.)

The project had one team member who was a very smooth manipulator. This person would ask for help with a given problem and within a few hours we would find ourselves sitting at the keyboard writing their code! This keyboard switchover took place without our intention. Regardless of the mentoree's hypnotic abilities, it was our responsibility to help the team member to learn, not to do the coding for him.

This is where patience comes in. Of course the mentor can do it faster (maybe 10 or 20 times faster!), but that's not why the project needs a mentor. If the project required just cranking out code, management wouldn't have given you a mentoring role.

The second horror story points out the importance of having communication skills and being able to anticipate peoples' actions. In the course of a six-week iterative cycle on another project, one of us worked with a select group from the team on a special prototype. One member of this prototyping group appears to be unable to deal with abstraction at all. He repeatedly asked extremely low-level questions that had the mentor stumped. The prototyping group had a tight schedule and a lot to accomplish. The low-level questions didn't need immediate and exhaustive answers because, well, this is Smalltalk and we trust that

Jan Steinman and Barbara Yates are cofounders of Bytesmiths, a technical services company that has been helping companies adopt Smalltalk since 1987. Between them, they have over 22 years of Smalltalk experience. They can be reached at Barbara@Bytesmiths.com, Jan@Bytesmiths.com, or at their website at http://www.bytesmiths.com.

the longstanding base classes do what they are supposed to do.

The mentor's response to many of these questions was, "Don't worry about that right now. You don't need to know that level of detail." It wasn't until several weeks later that the mentor found out the team member interpreted those answers as condescending. Unfortunately, the mentor wasn't told how the member felt early on, and the mentor had no clue he had caused problems with those answers.

The subsequent resolution came about when a third person on the team talked to both the mentor and the mentoree about how the prototyping project had gone, and discovered the communication impasse. So, for the mentors out there (and the mentor wannabes!), be careful about the way you phrase things, and pay close attention to the reactions of your mentorees. Epilog: Once the mentor was told of the problem, the mentor-mentoree relationship became productive again.

Just because a manager has recruited one or more mentors for a team doesn't mean they will be well utilized.

It is crucial in a project featuring mentors that there be a regular mechanism for mentorees to be told of their progress, and for mentors to be told of their progress. In a previous column we touched on the subject of end-of-cycle reviews. If the team is reflecting about what went well and what needs improvement at the end of each iteration, then telling the mentors how it is doing should be incorporated into this process. If necessary, make this feedback anonymous to encourage people to be honest in their comments.

MENTOREES

Just because a manager has recruited one or more mentors for a team doesn't mean they will be well utilized. The old expression "You can lead a horse to water but you can't make him drink" appears to be applicable to mentoring. No matter who the mentor is and how terrific he or she might be in that role, some people just won't drink! No doubt there are various reasons for it, and it is not something the manager can mandate. It might be helpful for managers and would-be mentors to know that team members typically fall into one of three categories when it comes to mentoring: eager, neutral, and (dare we say it) hostile or suspicious.

Mentor-Eager. The mentor–eager developer doesn't fear looking "dumb." He or she doesn't hesitate to ask questions, being concerned about proper OO design and learning how to do things "right." This sort of team member asks the mentor for reading suggestions, explores the class library, and doesn't want the mentor to do the work for them. The member wants reviews of decisions about design, algorithms, and so forth and requests design and code reviews. Working with this kind of team member is a pleasure—it's what gets the mentor through the tough assignments!

Mentor-Neutral. The mentor–neutral developer needs to be shown the mentor's value. If there is a good personality match, the neutral developer might become eager. If there's a personality clash, he or she can become mentor–hostile. If personalities are not an issue, this developer might still not make much use of the mentor.

A "neutral" might simply be a person who always prefers to figure out things for themselves. Regardless of how good the mentor is, a "leave me alone" developer will not make use of him or her. There's no point in the manager or mentor forcing the issue.

If a neutral feels comfortable "picking the mentor's brain" on his own terms, then short periods (a couple of hours) of "two-on-a-tube" might be the best way to work with him. Our Smalltalk team used this technique at Tektronix in the 1980s. The mentor sat with the mentoree, with the mentoree in control of the keyboard and mouse. They both worked through a specific problem, making use of the white board and exploring in the image. This short-term, focused activity, aimed at solving a specific problem, was very effective in demonstrating the mentor's value to a neutral.

Some people thoroughly enjoy working together closely, and two-on-a-tube periods can sometimes be over a week in duration, depending on the problem to be solved. We recently used this technique for several days in a Smalltalk boot camp we ran, and some people enjoyed it so much they did it for almost the entire two weeks!

Mentor-Hostile. A senior person in nonOO areas who lands back at the beginning of the learning curve with Smalltalk will sometimes reject any attempts at mentoring him. We've found that sometimes the mentor-hostile feels so threatened or insecure that he becomes downright hostile. The mentor hostile is used to being the mentor, not learning from a mentor.

In other cases, team members pay lip service to the value of mentoring, but are in fact disguising the desire to prove they can do it without help. They express this desire as needing "the freedom to make my own mistakes." This is a sure indication that the mentor-hostile developer is one who challenges all suggestions the mentor makes, and delights in finding and pointing out bugs in the mentor's code (hey, no one is perfect).

As we already said, there is no point in trying to force these developers to accept mentoring. The **S**anager needs to determine whether their insistence on making their own mistakes is hurting the project. If it is, perhaps the person belongs back in a role that makes use of their existing expertise.

Organizations that adopt object technology—Smalltalk in particular—must realize that their whole team might not learn and progress at the same rate. Some people may

never really "get OO." After a reasonable time period (perhaps as long as nine months), the people who still haven't gotten it need to be given alternatives. Neither the mentor nor the developer is to blame. Not all people are able to think abstractly, and they need to be given the chance to contribute to the organization in a job for which they are suited.

CONCLUSION

"Smalltalk guru" is not the equivalent to Smalltalk mentor. Not all team members will accept mentoring, and not all team members will get OO. Do the best you can as a mentor and as a developer, and try to keep egos out of the equation. If personality clashes are a problem, maybe the mentor has to go. This is a tough call that the manager will have to make. Good luck and happy mentoring! S

References

1. THE WORDSWORTH CONCISE ENGLISH DICTIONARY, Hertfordshire. Wordsworth Editions Ltd., 1994.
2. Steinman, Jan, and Yates, Barabara, "Secrets to Building Successful Smalltalk Teams," tutorial at Smalltalk Solutions, March 1996, New York, N.Y.
3. Steinman, Jan, and Yates, Barbara, "Special" Team Members," *The Smalltalk Report*, V5N6, February 1996, pp. 15-17, 28.

continued from page 23

OVERALL

Overall I'm quite impressed with both of these implementations. As prerelease products, they're obviously immature in some areas. For example, I had difficulty with the debugger in both of them. The GUI builder in Dolphin didn't work yet, and MT doesn't appear to have one. On the other hand, in a lot of areas, they're surprisingly mature. They already have advanced features like finalization and exception handling in place. Inevitably, it will be a while before they're fully loaded with those features that have nothing to do with a language, and everything to do with a successful project: industrial-strength source code control, native database connections, extra widgetry, report writers, business graphics, and so forth. Nevertheless, they show enormous potential, and are well worth your while to investigate.

Of the two, I expect MT to be the first choice for those looking for cool new features, and for those doing things that are traditionally difficult in Smalltalk (e.g. server-based Smalltalk, very tiny apps). Restriction to the newest version of NT will lessen their impact in the short term. Dolphin Smalltalk, with a very low price, Win95 support, MVC, and ActiveX applet support, has real potential to become the Smalltalk for the masses.

Both are filling niches that are under-represented by current implementations, and I hope they will enjoy great success. S