From: David Cook, Consultant, STSC/Shim Enterprise, Inc.
To: STSC Management
Subject: Suggestion for "Bottom 5" Projects

1. As you know, the July CrossTalk issue deals with the 2002 U.S. Government's Top 5 Quality Software Projects. Also, as you know, selecting the winners was difficult, given the large number of good projects nominated this year. It took several rounds of judging to narrow down the entries. And, once finished, there was not really a ranking from one to five – just a list of Top 5 programs. We should congratulate not only the Top 5 winners, but all the nominated projects – they were all top-notch.

2. To better meet our readers' needs, I would like to suggest that next year we also have a new competition – the Bottom 5 Software Projects. In an article that I co-wrote with Theron Leishman in the April 2002 CrossTalk ("Requirements Risks Can Drown Software Projects"), we cited a 1995 Department of Defense study that showed that 75 percent of software either didn't work or was cancelled. While I feel that the actual picture has improved today … well, things are not *that* much better yet.

3. Our readers already know how to build a project that will make the Top 5 – read CrossTalk regularly, talk to our Software Technology Support Center consultants, and learn from the Top 5 projects in this issue. However, we need to come up with criteria for the Bottom 5 next year. I suggest that we give the following criteria to eligible projects and developers.

   First, try every new and improved language, methodology, and tool that every vendor walking in the door has to sell. Listen to all of their claims, and believe them. Don't talk to previous customers. You can probably shorten your schedule 20 percent by switching life-cycle models to the Extreme Object-Oriented Visual Unified Structured Point Cost Methodology. Then save another 10 percent by switching to the newest version of C Triple Plus Double Sharp. Another 10 percent can come from almost any automated requirements tool (and remember, there is no learning curve to use the new tool!). Now you can optimistically plan on almost 50 percent shorter development time than your last project. Plus, the new tools are guaranteed to decrease testing and integration by almost 50 percent! Since 50 percent + 50 percent = 100 percent, you're already finished with the project!

   Plus, remember that when using a new tool or methodology, you only need to send one or two of your staff to training; they in turn can train the rest of the team. Send the most experienced geeks you have, regardless of the fact that they have significantly lower people skills than a dead opossum. Also, remember that the technical geeks sent to the training class will remember only the really obscure and complex issues. The basics (such as simply starting the tool) are probably pretty obvious anyway.

   Second, there is no real need to have customer/developer communication. If the developer really needs an interpretation of a requirement, then it should have been asked during the requirements phase. Once requirements have been gathered, work to eliminate those nasty clarification discussions. When the project is delivered, you can then take a lot of time finding out what the customer really wants. Good developers know that verification and validation are much quicker without end-user involvement. In addition, open channels of communication allow customers to develop contingencies if the schedule slips. This makes you look bad.

   Third, make those critical software engineering and computer science decisions without a real software engineer or computer scientist around. Just about any engineer or pilot is truly qualified to make those complex decisions that affect major parts of the schedule and deliverables. In fact, it really helps if those making the critical decisions meet two criteria: they haven't really used the system for 10 years, and they haven't really had an up-to-date engineering course since the one they took that covered the basics of vacuum tubes. Don't think of calling in an outside consultant to help; that might make you look less capable. It's easier to get the experts to try and fix the problems later than it is to have the experts help you prevent the problems from actually occurring.

   Fourth, remember that the Capability Maturity Model® (CMM®) and the Capability Maturity Model Integration (CMMI®) are really just paper exercises. We all know that there are really three processes in any serious development effort: the one that is documented and briefed to the CMM/CMMI appraiser, the one that management believes is being used, and the one that you really use. Make sure that the documented process is truly a paperwork exercise. Don't actually use the documented process; that might slow you down and force you to think about quality way too early. Heck, we all know that quality really starts in the testing phase.

   Finally – this is absolutely critical – trust existing documentation, especially regarding legacy code or systems that are critical interfaces. Trust that the code does what is says it will do, and that the interface works exactly as specified. After all, if you started looking at the interfaces before integration, you might have to change the architecture and design. Just wait until testing, and patch the code as necessary.

4. There are, of course, other criteria for ensuring a projects' inclusion on the *Bottom 5* next year, but the ones above are usually adequate to ensure that a project qualifies for nomination.

5. As long as projects continue to follow the five criteria above, it will reduce the number of projects that are of sufficient quality to merit nomination for the Top 5 contest. In fact, it almost guarantees that a project can't be considered for the Top 5 next year. And having fewer projects submitted will make the scoring and selection a lot easier.

– David A. Cook
Software Technology Support Center/Shim Enterprise, Inc.