The Proactive Programmer

I'm writing this on January 4, 2003, eleven days before departing for Portland to attend a much-anticipated summit on writing better code. The people convening the summit have been working on it since last August. (I've been helping out a little myself). Two months and two hundred emails later, we invited a small number of willing and noteworthy participants. Hopefully our discussions will benefit many who seek to improve the quality of their code and encourage more to want to.

I am writing about this summit before it has convened. because: 1) many of the other participants will certainly write about the results, and I'm happy to "grease the skids" for them, 2) it's a topic I believe CUJ readers are always interested in, and 3) there is enough wisdom in the 800+ emails the attendees have already exchanged to offer something worth while right now. (The quotes that follow, with two exceptions, are from these emails.)

It is no mystery, of course, that any lack of quality in software is due to external as well as internal forces. Budget constraints or technical decisions made by management that shouldn't have been are beyond a programmer's control. As a ZDNet News article once stated[1], "Fear of competition causes one to race to market with poorly tested features because you believe feature differentiation will win you sales." The sales gain may indeed occur, but this view ignores the consequences of having dissatisfied customers. Alistair Cockburn talked about this "second kind of cost - lost customer or bad-taste-in-mouth reputation… you won't make money fixing the bug, but you [may] save 4 or 6 or 10 future bad tastes. Possibly 1/2 hour of work for that saved negative value could be worth it."

These external forces can also give programmers themselves a "bad taste." Angelika Langer has "been talking to programmers who cared about code quality and were discouraged and eventually suffocated by the work environment they were working in… The reason [for lack of quality] often is not programmers' ignorance, but simply the fact that quality is not asked for. To my mind, one step towards better code is explaining the relevance of better code not to programmers, but to product and project managers."

This may be music to your ears (it was to mine), but Jack Ganssle responded, "Highly ethical programmers do things as correctly as they can all the time. They fight management's stupid decisions… Unethical programmers don't - resulting in lousy code and worse products. To an ethical programmer, [some] things … (filename space problems, buffer overrun vulnerabilities, etc.) just cannot happen." You can't blame a buffer overrun on management. A responsible programmer's internal forces just won't let it happen. While a quick code review would catch such an error, some programmers don't want anyone to look at their code, even though their code is part of a product shared by team members and consumers alike. Besides, as Angelika added, "constructive

[1] Found at http://zdnet.com.com/2100-1106-955432.html

feedback is extremely important, not only for code quality, but also for the programmer's wellbeing.”

There are and have always been good resources to help us do a better job at creating good software. I’m looking at twelve such books in a stack on my floor right now, half of which were authored by summit attendees. Scott Meyers remarked, however, that “it's important to have a list of best practices, but it's also important to talk about why such practices aren't routinely employed and what can be done to change that.” It’s crucial for any craftsman, and programmers in particular, to “keep up”, or as Bruce Eckel put it, to be a “life-long learner.” It takes effort, but the learner is well rewarded. Eric Hoffer (not an attendee) has observed that, "In times of change learners inherit the earth while the learned find themselves beautifully equipped to work in a world that no longer exists."

I look forward to the upcoming summit and the Good that will surely follow in its wake.