Improve Your the Quality of Your Application by Delegating Testing

We all know that the worst person in the world to test an application is the programmer who wrote the code. Right after that are the user and the programmer's mother. Nonetheless, testing is an integral part of the application development process; if you don't test to find bugs, your users will.

Many large firms use separate departments for testing. For example, RDI Software Technologies, a 70 person firm in Chicago, has had a Quality Assurance Group for unit and integration testing for some time now, and their applications frequently run for months without a bug being found by a user.

Of course, small shops and independent developers can't afford this luxury of a separate department just for testing, right? Wrong! Over the past two years I have greatly improved the quality of my shipping applications with a dedicated testing crew despite being a one-man operation. The purpose of this article is to show you how to do the same thing.

Using High School Students as Testers

After spending some time touring through RDI's shop, I realized that I needed to have the same capability - a separate group responsible for ensuring that the product going out the door met or exceeded the specifications it was designed for. However, being an individual working out of my house posed a number of special constraints. I didn't view these as dealbreakers, merely as a different set of constraints than, say, RDI was subject to.

The first constraint, obviously, was funds. I couldn't afford to hire someone full-time for what essentially was a non-billable task. The second constraint was that, even supposed I had the funds, I usually didn't have enough work to keep a tester busy full-time. And the third constraint was limited turnaround time. I could foresee that I'd often need to be coding during the day, and have the testing performed in the evening.

I don't know how it came to me but at some point, I realized that the local high school held the answer. Whitefish Bay High School is a top-rated college prep school that sends dozens of students to universities like MIT, CalTech and Stanford each year. It was my alma mater, so I was familiar with the types of students, the environment of the school, and many of the teachers. Furthermore, it was located just a half mile away - and our suburb is one where nearly all the kids lived close enough to walk to school. As a result, I had a ready pool of intelligent, ambitious high school seniors who were geographically convenient.

I figured I could train one or more of these kids to perform the mundane but important chores of testing. As it turned out, this guess was right on the mark.

Looking for Candidates

The first step was to introduce myself to the new calculus teacher at the school. My calculus teacher had retired a few years earlier, and the new teacher was extremely helpful in recommending a few of her students. I ended up interviewing several likely candidates. In this interview, I described what I did for a living, explained what I was looking for, and why.

If they still appeared interested, I asked them a number of questions that helped me get a reading on how smart they were, and what their attitude was like. After all, I had to remind myself that these were adolescents who were as likely to be thinking about Prom as finding a part-time job. They probably weren't going to have any "real-world" computer experience, and were certainly not going to know anything about business. Their native intelligence and attitude were the deciding factors.

Training the Students

Once hired, I had to train them to use FoxPro. Again, they weren't going to be programming, but they had to be proficient enough with interactive FoxPro to access raw data to determine what was going on behind the scenes. Essentially, I had to get a smart, but completely inexperienced, user up to speed as soon as possible. It turned out that having them do data entry for a couple weeks solved the problem. (And it seemed that I always had some sort of data entry project laying around at the appropriate time.)

Remember the scene from The Karate Kid where the youngster asks the old man to teach him karate, and the old man has the kid wax the car, then paint the fence, and then scrub the floor? After a couple of weeks of using FoxPro to enter and edit data, they were comfortable setting up Browses, relating tables, searching for data (both through SEEK and LOCATE), creating filters, and so on. They also learned a few things about DOS - using PKZIP, copying files, moving between directories, and so on.

In order to lower resistance to this seemingly rudimentary work, I explained that, just like in The Karate Kid, they were learning the fundamentals of FoxPro well enough so that they were able to develop a "sixth sense" of when something went wrong. Ever had a user

do something stupid, and you ask them about it, and you just can't believe that a bell didn't go off in their head at the time? The experience with repetitive data entry helps the students develop a "feel" for the program.

The Testing Process

Now that they're comfortable with FoxPro, it's time for them to start testing. First, I walk them through the program, explaining what the application is supposed to do and how it fits into the bigger picture of the customer's business. Next, we go through the Functional Specification, and step through each menu item and screen.

The next step is to break the system down into modules that can be individually tested - usually, these are screens, functions, or data transfer operations. For each module, they'll sketch out a spreadsheet-style grid, listing each "Function Point." These include every SAY, every GET, and every action on the screen. Then they go through a three step process.

The first step is to "push all the buttons." This sounds like a trivial, almost ridiculous, step. But it's the rare programmer who hasn't had that gut-wrenching feeling that occurs when something that "was working perfectly yesterday" blows up in front of the customer. It should have worked, because you copied that button from another screen with identical functionality. Except that you forgot to change the memory variable, or some other small but disastrous change. This step eliminates that surprise. It also acquaints the student with the application. Remember, these are high school kids, whose only other business experience might have been watching Charlie Sheen and Michael Douglas in the movie Wall Street.

The second step is to verify that "pushing each button" does the right thing. Pressing the Browse pushbutton actually results in the correct table coming up in a Browse window, and the correct fields showing, as defined in the Functional Specification. Pressing the Delete pushbutton actually results in the user being asked if they really want to Delete that record, and that selecting "Yes" does go into the table and delete the record.

This is why I look for extremely bright kids. It's too easy to hit the buttons and have it appear that they worked. The tester has to be skeptical - just because a **different** record displayed on the screen when they pressed "Next" doesn't mean that the **next** record was displayed. The tester will double-check by looking at the current index order and watch the record number change in the debug window or the status bar. The right kid has enough curiosity not to accept the status quo, but to continually ask "Why."

The third phase is to step back and look at the big picture. Are the business rules being followed from screen to screen, and does the operation of the application make sense? The student is often in a much better position to sense this; they don't know enough not to ask "Why do you have to leave this screen just to look up the last date that the truck was put through maintenance?"

Additional Functions

After they've gained some experience with testing, they've developed a feel for the way my applications operate. They're then ready to offer comments and critiques about the interface. Most of the interface is consistent due to the use of common functions and reusable code. This means that parts of the program that don't behave in a consistent fashion stand out.

In some cases, they're also able to write out documentation that goes in the help file. After all, they know how the program works and while it's important, my time is probably better spent doing something else. After reviewing and polishing up the help files, I have the customer review it. This helps down the road; if the customer comes back complaining that something doesn't work like they thought it was supposed to, we refer to the help file that they accepted, and it should be clear whether I have something to fix or if they're asking for an enhancement.

A Win-Win Situation

I'd like to stress that this relationship has been a win-win situation. I've been able to sleep a lot better over the past couple years, not having that feeling of "Boy, I sure hope it works" when I go to bed at night. And of course, my customers are happier as well. They're finally getting software that works as it's supposed to. This benefit is extremely inexpensive.

I've also found that I don't mind bugs as much. When you test something you wrote, how do you do it? Typically, you gingerly press a key, whispering "Please don't break, please, please, please." When it doesn't break, you breathe a sigh of relief and go to the next function. The students have a different attitude: "OK, so you didn't break here. Now, let's try this way!" They have to be merciless. And then when they come to me with a list of anomalies, it's actually almost as much a challenge as writing the program the first time. I'll often try to figure out what went wrong before touching the keyboard.

And the students? First, they've got a job - and a real job, not one cutting grass, baby-sitting, or washing dishes. Second, this job actually gives them real world experience - a chance to do important work as well as exposure to the imperfections to the reality of the work-a-day world - with limited resource and error-prone humans. And they get a chance to see firsthand how critical it is to keep

customers happy. Third, this job looks good on their resume - since most of them are applying to highly competitive institutions, every edge helps. Finally, the nature of the work lends itself to hands-off management. I give them a diskette, a specification, and tell them "go at it." In other words, I'm treating them like an adult, a rare but much appreciated attitude in the eyes of a 17 or 18 year old. I've found that they return the favor tenfold - they work extremely hard at showing that my trust in them is not misplaced.

A hidden benefit is that the students learn a lot about communicating, and I mean this two ways. First, they have to learn how to give their boss bad news - "Hey, boss, you're a moron. This is the third time you said you fixed it, and it's still doesn't calculate the amortized cost correctly." But they also have to learn how to clearly explain concepts and processes that can be rather complex at times. Rather than "It doesn't work when I add a record," they learn to explain "I press the 'Add' pushbutton in the Tool screen, and enter data in any of the fields. When I hit 'Save,' the error message 'Variable m.cIDTo not found' displays."

The Downsides

While this testing mechanism has worked well for me, it's not necessarily optimal. I often forget that they have other "important" things to do, and aren't always available at the drop of a hat. More than once I've had a student call and say "I know I was supposed to come in and test this afternoon, but you know that girl I've been talking about? Well, she said she could go to the concert tonight and, well, I was hoping I could take today off because I gotta do some stuff to get ready." I guess this is no different than other employees; it's just that their priorities are of a different sort.

There's also a limit to what they can test, given their inexperience with business. And it's easy to forget to review their work. Just because they came back with a list of three bugs doesn't mean they didn't miss others. You still need to monitor what they are doing. Nonetheless, the advantages have far outweighed the disadvantages and I'm hooked on using students to do my testing. Even if I wanted to stop, I doubt my customers would let me.