

QA for Everyone

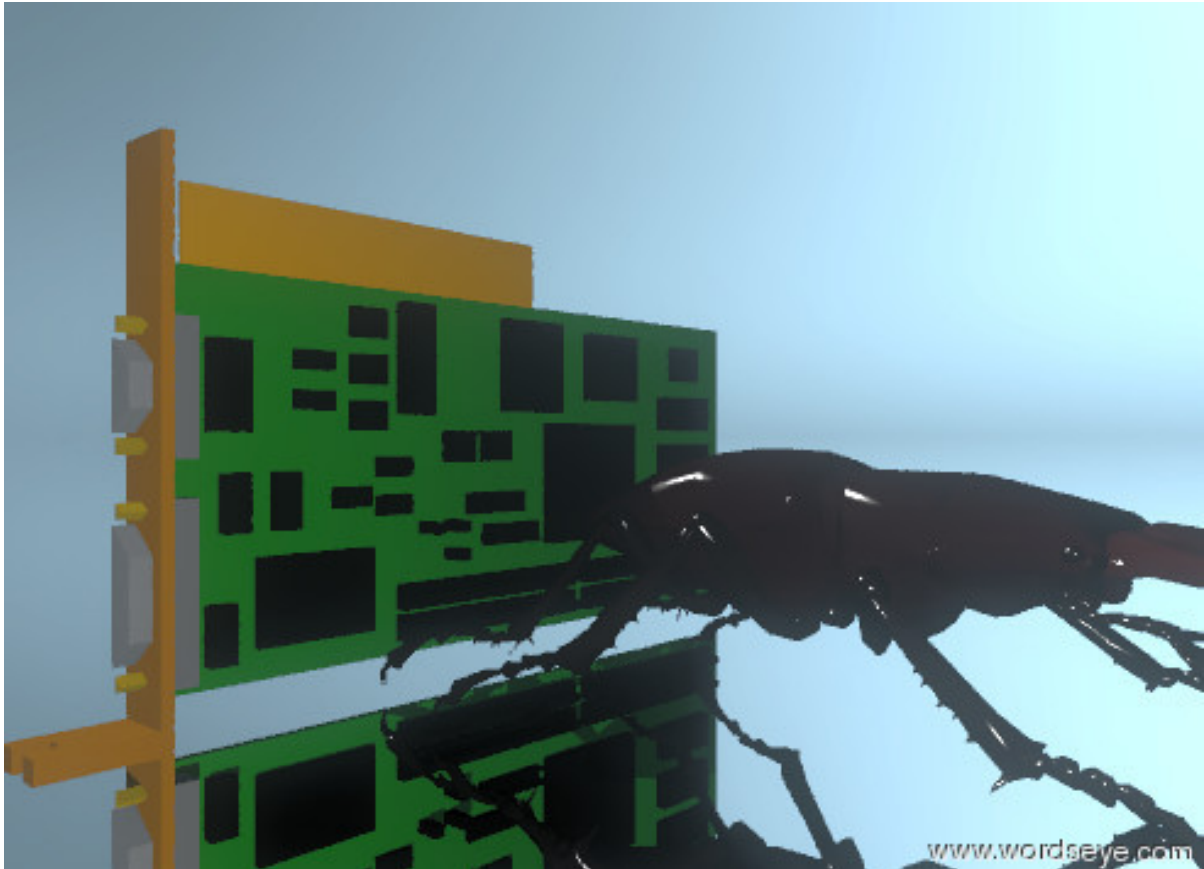
by Philip Chu

Table of contents

1 Publication Information.....	2
2 Test Early.....	2
3 Test Often.....	3
4 Test Everywhere.....	3
5 Test Everything.....	4
6 Test Methodically.....	4
7 Bug Reports Welcome.....	5
8 Show Me Yours, And I'll Show You Mine.....	5
9 Protect the Database.....	6
10 You're Not the Boss of Me.....	6
11 But You're Not Peons.....	7

1. Publication Information

Copyright ©2005-2009 by Philip Chu All rights reserved.



"Is this going to be a standup fight, sir, or another bug hunt?" - Marine Private Hudson to Lieutenant Gorman, in *Aliens*

2. Test Early

Testing isn't something you do when you're almost finished. It's something you do as soon as you have something running. It's not just management that is prone toward delaying testing - programmers are often just as bad.

- On my first commercial software project, the software was so incomplete and crash-prone that I was reluctant to give the first "alpha" build to our test group. Fortunately, the QA manager insisted, and the resulting bombardment of bug reports

forced me to immediately focus on the problems that prevented the testers from exercising the application in any meaningful way. Left to my own devices, I would have worked on what seemed to be the harder and more important core 3D issues, and probably delayed too long on seemingly mundane issues like the user interface, load-save functionality, and compability with all the varieties of consumer hardware we were planning to support.

The bug count on any project has to go up before it goes down. Get that spike early and you can whittle it down gradually, instead of attempting to deal with it at the end.

3. Test Often

Once you start testing, keep testing.

- On my first console game project, our publisher didn't have QA department look at our game until just before E3, the big game industry trade show where retailers get a look at the upcoming games. That bit of testing just provided foreshadowing of bug reports to come - after the show they didn't look at the game again until less than three months before the submission date. Within a week we had several hundred bugs logged.

Testing should be a continuous part of the development process, not a footnote.

4. Test Everywhere

Test on all the platforms that you plan to support. You can't assume that a product running on a system will run on all minor variations of that system.

- At a computer graphics software company developing applications for SGI workstations, our chief sales guy proudly informed me he sold thirty seats of our product to a prominent video game developer. Unfortunately, those were thirty licenses of a product that was still in beta, and the customer was planning to run on new SGI workstation models that we hadn't even seen, yet.

I had to make two followup visits to the customer - one to make a feeble attempt at diagnosing our product crashes on the new machines, and the other to get yelled at by the customer (they were yelling at one of our sales support guys, but he couldn't take it any more so they sent me in as a replacement)

In an ideal world, everyone would have access to the every platform of interest. Usually, there will be compromises due to constrained budgets or limited availability of the target hardware, particularly if you're in a small company. I've seen several situations where the QA group was first in line for all new types of hardware, probably reasoning that QA is

responsible for testing on all platforms (turf may have something to do with it, too). But assuming there's logic behind it, that logic is misguided.

- Soon after I joined the aforementioned graphics software company, the QA manager chastised me for not testing my code before I integrated it into the test build. Well, I did test it on my workstation, but it happened to fail only on hers because it was the latest and greatest model and she had the only one (This was at a time when it seemed like SGI was cranking out new models every couple of months, each with new subtle hardware and software incompatibilities) This meant I had to sit at her desk to try to diagnose the problem.

Every programmer knows it's ridiculously more difficult to track down and fix code on a machine that's not set up with a development environment. So you can save everyone a lot of time and grief by making sure the programmers get first dibs on new hardware (as long as they use it!)

5. Test Everything

There's not much point in just testing the obvious - test what the developer may have overlooked. Exercise boundary conditions (zero dollars, empty text values, empty files), feed the program garbage data (negative numbers, text instead of numbers, non-alphanumeric characters, random files). Treat it like a game, where the goal is to find a way to break the program.

6. Test Methodically

It's just as important to know what works as what doesn't work. So record all test results. Better yet, write down a test procedure. Even better yet, automate testing as much as possible. Not only does automated testing, in the same manner as automated build and backup procedures, give you confidence that you have a known and reproducible process, but it allows you to test conditions that are impossible to create manually.

- Automated testing is especially important in simulating network usage. When I worked on a networked virtual military exercise, manual testing revealed a lot of interesting bugs (submarines flying above water, ships plowing through continents, planes flying backward), but problems with handling high network traffic didn't become apparent until the exercise day when I turned the switch on and our whole system crashed.

Web-based systems in particular should always have automated test systems. A lot of web startups see their products work with a few users and just extrapolate. I was horrified, after moving from the Bay Area to a startup in Venice, CA, to see the web server team test by asking a few employees to try connecting to the server at the same

time. And I was more horrified to see that test crash the server immediately.

7. Bug Reports Welcome

Testing is everyone's responsibility. Hopefully everyone involved in developing a software product will actually run that product. So take advantage of that manpower by making it easy for anyone to submit a bug report.

- When I worked with Silicon Graphics as a vendor, back when they were top dog in the graphics world, I had a love-hate relationship with them. Mostly the latter. Their developer support program was great, unless you wanted to report a bug. They officially could not receive bug reports - those had to be submitted to their (paid) customer support program which only accepted problem reports over the phone, via a receptionist who would take down your phone number and workstation system ID, and attempt to jot down the description of your problem. Weeks later, a junior engineer would call you back so you could repeat the problem description. Then the engineer would be transferred to another group and another engineer would call you.

It's one thing to solicit bug reports from everyone, it's another thing to actually get them. Only the truly motivated will report bugs if the process is at all inconvenient.

- I worked with one QA group that put together a sorely-needed online bug entry form. The bad news - it looked like a tax form. Field after field, all mandatory and without convenient default values. The result, hardly anyone outside QA would submit a bug report, and any that were submitted tended to have misleading wrong data.

Keep the bug reporting process simple. It's a bug report, not a TPS report.

8. Show Me Yours, And I'll Show You Mine

By the same token, use your customers as testers. Companies often do this with beta and even alpha releases, but there's no reason to stop there. Assuming you have a real customer base, the real testing will start when your product ships.

- I wasted several months fretting over a mystery bug when developing a product for SGI workstations and ended up releasing the product with a convoluted workaround. After several months of complaining alternately to our compiler vendor and to SGI, and then tracking the behavior down to a particular ordering of the libraries passed to the linker at build time, our SGI support contact informed me that it was a known bug that had been on the books for a year.

In contrast, one of my favorite experiences in adopting Java has been the Sun Bug

Parade, which has allowed me to check if a bug I've encountered has been encountered by others, and what the current status is. As a bonus, the Bug Parade allowed you to vote for the bugs that you consider most critical and displays the current vote for each entered bug.

9. Protect the Database

QA has their own product - the bug database. The bug database represents the state of the product and should drive all development decisions. Thus it should speak the truth.

- A month before the release date of a 3D graphics program, the company president insisted that VRML support had to go into this release ("I just want it", was his cogent argument). After we wasted a couple of days trying to cram it in, I pointed out that our bug database listed 300 bugs, most of them categorized as high priority. The president said "Oh, I didn't know we had so many bugs" and changed his mind.

Just about every QA group I've dealt with would escalate originally low priority bugs to higher priority toward the end of a project as a not-so-subtle way to tell developers what to fix. That is a misuse of the bug database - if all the critical and high-priority issues are resolved, don't relabel low-priority issues as critical. Just say it's OK to fix low-priority bugs, now. If management has prohibited work on low-priority bugs, then fix that (at least the edict, better yet, fix management). And don't downgrade a bug just because no one wants to deal with it.

- The grandest act of wishful thinking I've seen applied to a database was invoked by a game publisher who ordered their QA staff to mark all remaining bugs in the database as fixed after we made what we hoped was our final submission. As if the bugs would really just disappear just because the product was going to ship. The tactic made even less sense considering the European version of the game still had to undergo localization, testing and submission.

Cooking the QA books is not as illegal as cooking financial books, but it should be. Protect the integrity of the database.

10. You're Not the Boss of Me

I've met a few QA people who somehow thought that QA was on top of the organizational pyramid.

- When CMM was all the rage, a QA coworker of mine gave a presentation on the topic and concluded that his group should oversee all of development, meanwhile waving the CMM book that clearly stated that shouldn't be the case (there should be CMM oversight

of both groups, but considering we had fifteen people, I don't know why were talking about it at all)

Most QA professionals I've met, depending on the industry, were in it as an entry level job (in game development, it's touted as a first step to becoming a game designer) or as a less stressful alternative to programming (one tester told me she just didn't like thinking about code after coming home from work - I don't blame her!). But it's the type of job that can attract cops and bureaucrats.

- I worked with one QA manager who, while a nice guy, liked to lecture the development staff about proper procedure. It surprised no one when he casually mentioned that he used to be a traffic cop.

Now, I love working with testers who want the product to be the best possible. But the best way for QA to estrange the development staff is to take things personally.

- On the first project where I was a manager, the QA lead was originally going to assign a senior artist to test the graphics product we were developing, on the theory that experienced professionals in the field were best acquainted with the valuable features of the product. That might have worked out OK, but I'd seen enough problems dealing with some senior (and even junior) testers who were accustomed to thinking of bugs as impediments to their own pet art projects (one guy's bug reports consisted of "It's broken", "I didn't do anything", and the ever helpful "Fix it!") rather than behavior they should attempt to discover, reproduce and track down. Since I was more interested in getting quality bug reports (or at least peace and quiet) instead of whining or tantrms, I almost begged the QA manager for the services of one even-tempered entry-level hire. I think that paid off - he did a great job sitting through crash after crash and submitting dozens of detailed bug reports every day until the code got stable.

Remember, as a tester you're paid to find and help track down bugs. Disgruntled, whiny customers can be had for free.

11. But You're Not Peons

On the other hand, management needs to understand that QA is an intrinsic part of software development, not a cleanup crew shuffled in at the end of various projects. Especially, when you have a salaried QA group.

- While working on a long-term hugely spec'd proposal processing server for the Hubble Space Telescope, I was both bored and stymied (I'm not proud of this) and instead spent my time on a little stealth project, a GNU Emacs mode for editing proposals. It attracted some interest from in-house users and management, so I trudged down to the QA room,

where I found everyone was sitting idle (as you may guess, we weren't cranking out a lot of software). The lead tester showed some interest in trying out my program but said he couldn't spend any time on it without permission from his boss, then returned to his solitaire game. Your tax dollars at work.

It's not just a matter of keeping people busy. QA knows how to use your product as well as anyone else in the company, and probably better. Use that expertise.

- Sometimes it seems people want demos to fail. I've had several managers who habitually dropped into my office expecting impromptu demos for a VIP right while I'm coding. One was apparently unaware that I arrived at work three hours early to prepare a demo and decided to bring the visitor back a few hours later so he could watch me compile and crash. I got so irritated at another that I let him run the demo himself and flail in the debugger. In each case, the QA guys were in the room next door. Not only were they painfully aware of what potholes to avoid if they wanted to try out various features ("don't click there!"), they also knew which build from the past several days would demo best.

But testers aren't just useful for demos - they are your experts on the state of the product on the state of your product. If you want to know how close your product is to release, ask them.

- The one project where I had total greenlight authority, I consulted not upper management (who kept demanding new features), or marketing (who wanted to delay the release date for no good reason), or sales (who proudly sold a site-license for an unfinished and crash-prone version of our last project to a high-profile customer) - I asked the guy who'd been testing it every day for six months. If he said it was ready, it was ready.