I'm Tired of Finding Bugs

by Alan Page

I have been testing software for more than thirteen years, and I'm tired of finding bugs. It isn't that I regard finding bugs as unimportant—I would much rather find a bug myself than have a customer find it. And I'm certainly not tired of testing, but the goal of testing isn't to find bugs. The tester is responsible for exercising the product, reporting information, and many other activities; finding bugs is merely a side effect. I am tired of testers finding too many defects that should have been found much earlier or never introduced in the first place. We find too many issues too late in the product cycle and let too many bugs get through to the customer. I don't want to *find* bugs anymore; I want to *prevent* them from happening.

I read a lot about software testing, but the books, blogs, articles, and other materials seem to focus primarily on approaches for finding bugs. Popular techniques, such as exploratory testing and error-guessing, can be effective tools, but they are focused on finding. These and any other approaches predominantly intended to discover errors already in the code are attempting to achieve quality through testing. In other words, they attempt to "test quality into the product."

Nearly every book on software engineering has a chart or graph that demonstrates how the cost of fixing a bug increases the later it is found in the product cycle. If everyone agrees this is the case, why are we not doing more to find bugs earlier? Many engineering teams are beginning to involve testing early in the product cycle, and I think this is a good trend. I have heard the shouts of "Let's drive quality upstream!" for years, but I still think we aren't doing nearly enough.

How can we reach higher levels of quality and find fewer bugs late in the engineering cycle? Most efforts to drive quality upstream involve testers' reviewing specifications early and creating test cases or models from the specifications or early prototypes. These are certainly much better methodologies than waiting for code to be "thrown over the wall" to the test team, but a considerably better solution is to prevent bugs from happening in the first place. As software complexity increases, some bugs may be inevitable—but most are preventable.

Software bugs usually exist because someone made an error. The person writing the specification may have accidentally left out a detail, a developer may have made a typographical error, or the wrong library function may have been used. Can human errors be prevented? Of course they can. Shigeo Shingo, an industrial engineer from Japan, invented a concept called *poka-yoke* ("avoiding inadvertent errors"). In a manufacturing plant, poka-yoke may be implemented by putting a notch on a widget so it can only be attached to a larger widget in a specific position.

Every day, I encounter dozens of systems put in place to prevent human errors. The circuit breaker in my fuse box prevents my overloading an outlet and starting a fire. My bathroom sink has a small hole near the top that prevents me from overfilling it. My favorite example is that my new car will not lock if the keys are inside.

Prevention techniques are just as applicable to software. I once worked on a project where I filed a handful of bugs against various components that were using a library function incorrectly. The error was an easy one to make and did not cause an obvious bug, but it did cause a significant performance problem and memory leak. After I tracked down the problem, I wrote a quick code-scanning script to detect all occurrences of the error across the entire product source. Once the relevant bugs were filed, I added this script to the list of checks that were run every time a developer checked in code. While I did not actually prevent the initial creation of this bug, I did prevent the bug from ever finding its way into the product we were testing. Furthermore, due to the checks, the developers quickly learned to use the library function correctly.

After that success, every time I found any error that could be detected through automated code analysis, I wrote a routine to detect the error and then turned the detection into prevention by adding the script to the check-in routine. By the time the product shipped, there were dozens of checks preventing hundreds of bugs.

Prevention techniques do not need to be tools. I recently was working on a project where a significant number of design-related errors were occurring in one team area. I could not devise a way to automatically detect the issues, but I was convinced an informal code review would have caught them. I showed the problem to the manager in charge of this area, and he agreed with my assessment. Shortly thereafter he began requiring a code review for every new design implementation or change. After that, I saw very few design errors from that team. Better yet, other errors I had missed also were prevented.

Defect prevention ultimately requires some investigation into the underlying

From SmartBear Software
the experts in code review...

Code Collaborator

A remote peer code review tool that makes inspections efficient and effective.

Free Book!

Read about the largest case study on peer code review ever published.

Free Copy, Free Shipping.

Visit our web site today!

Best Kept Secrets of Peer Code Review
Modern Approach. Practical Advice.
Jason Cohen

www.codecollab.com

SmartBear Software
877.501.5651

cause of the bug. There are several methods for analyzing bugs to determine the core origin of an issue, but I have experienced a great deal of personal success using a much lighter form of analysis. The best part of my technique is that I get to act like a three-year-old. Every time I find a bug, I ask as many "Why?" questions as it takes to get to a point where I can implement a prevention technique. Why did this bug occur? Why was that particular function used? Why was it possible for them to use the function incorrectly? Why was this error not caught in the code review? I can't guarantee that these questions will uncover a prevention technique for every bug, but every prevention technique you find and implement will significantly reduce the number of bugs left for you to find later.

Software will continue to have bugs for many years to come. I can accept this fact, but I cannot accept the sheer number of bugs we allow—and expect—testers to find. If we want to be serious about quality, it is time to get tired of finding bugs and start preventing their happening in the first place. **{end}**

Alan Page is a test architect in Microsoft's Engineering Excellence group, where he develops and conducts training for testers and works with product teams across the company to identify and promote best practices in testing. Email Alan at alanpa@microsoft.com.

Index to Advertisers

American Society for Quality	www.asq.org	24
AutomatedQA	www.automatedqa.com	Inside Back Cover
Empirix	www.empirix.com	17
IBM	ibm.com/takebackcontrol/flexible	10
iTKO, Inc	www.iTKO.com/lisa	19
Mercury Interactive Corporation	www.mercury.com	Inside Front Cover
*m*Verify Corporation	www.mVerify.com	13
NVP Software Testing	www.nvp-inc.com	16
Parasoft Corporation	www.parasoft.com	8
PNSQC	www.pnsqc.org	31
ProjectWorld	www.projectworld.com	42
Rally Software	www.rallydev.com/bsm	41
Seapine Software	www.seapine.com	2
Smart Bear Software	smartbearsoftware.com	44
Software Quality Engineering Training	www.sqe.com/training.asp	25
Software Quality Solutions (SQS)	www.sqs.com	Back Cover
STAR*WEST* 2006	www.sqe.com/starwest	15
StickyMinds.com	www.StickyMinds.com	5
Telelogic	www.telelogic.com	Opposite 7
Visual Studio 2005 Team System Training	www.sqe.com/trainingprograms	40
Worksoft	www.worksoft.com	1

Display Advertising

Shae Young syoung@sqe.com

All Other Inquiries

info@bettersoftware.com

Better Software (USPS: 019-578, ISSN: 1532-3579) is published eleven times per year. Subscription rate is US $75 per year. A US $35 shipping charge is incurred for all non-US addresses. Payments to Software Quality Engineering must be made in US funds drawn from a US bank. For more information, contact info@bettersoftware.com or call (800) 450-7854. Back issues may be purchased for $15 per issue (plus shipping). Volume discounts available.

Entire contents © 2006 by Software Quality Engineering (330 Corporate Way, Suite 300, Orange Park, FL 32073), unless otherwise noted on specific articles. The opinions expressed within the articles and contents herein do not necessarily express those of the publisher (Software Quality Engineering). All rights reserved. No material in this publication may be reproduced in any form without permission. Reprints of individual articles available. Call for details.

Periodicals Postage paid in Orange Park, FL, and other mailing offices. POSTMASTER: Send address changes to *Better Software*, 330 Corporate Way, Suite 300, Orange Park, FL 32073, info@bettersoftware.com.