Editor: Robert L. Glass ■ Computing Trends ■ rlglass@acm.org

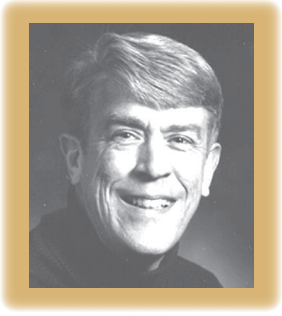
An Ancient (but Still Valid?) Look at the Classification of Testing

Robert L. Glass

… in which I challenge the belief that everything old in the software field is obsolete (a challenge that seems particularly relevant to Software*'s 25th-anniversary issue).*

To what extent can you trust a 17-year-old software engineering book?

For most of us, the answer is easy—little or not at all. After all, things change so fast in the software field, and new books come into print so often, that there's no need to go back into the ancient history of the early 1990s. That's so "last millennium"!



That's my answer, too, I think. However, I've recently had occasion to challenge that belief.

Lost and Forgotten

I was recently called on, in my guise as an occasional reviewer for academic publications, to review a paper on test-case classification. It was a good paper and of particular interest to me because I've written on that subject.

As many of us do (although we might be embarrassed to admit it), I glanced at the end of the paper to see whether it referenced my own test-classification work. It didn't. Now, that's a not-uncommon experience—most of us tend to overvalue our own work and think it should be cited by anyone who even comes close to writing about a topic dear to our hearts. But in this case, I was severely disappointed because I was particularly proud of my test-classification work.

Now, here's the rub. This work had appeared in my 1992 book *Building Quality Software*, published by Prentice Hall but out of print for nearly a decade. Worse yet, it was buried in a lengthy discussion of testing approaches, finally emerging on page 144 of a discussion that began on page 108, three dozen pages earlier! So how could I possibly expect modern-day researchers to have known what was buried deep within (strike one) a 17-year-old book (strike two)? In this analysis, I'm close to striking out.

But, doggone it, I'm still proud of that work. So, at the risk of appearing obsolete, I offer here a snapshot of those oh-so-old thoughts on software testing.

Why should you care about my software testing work? Because (he immodestly said), it contains insights I haven't found in much more recent studies and books. And because testing is arguably the most important software development phase—the one that expends the most time and money, and the one that influences product reliability the most.

So, Here Goes

Picture a 2D matrix. Down its vertical axis I include the following four testing goals.

Requirements-Driven Testing

This testing examines the requirements for the artifact being tested and explores whether the artifact satisfies them. This most often happens in practice through a requirements/test-case matrix. Such testing is often called "black box" testing because those conducting it—either programmers or testers—need not know how the software is built; they need only to see the requirements or user documentation.

Continued on p. 111

 0740-7459/08/$25.00 © 2008 IEEE

Authorized licensed use limited to: University of London: Online Library. Downloaded on July 12,2021 at 08:24:30 UTC from IEEE Xplore. Restrictions apply.

Continued from p. 112

Structure-Driven Testing

This testing determines whether the as-built artifact's elements work correctly. These elements can be modules, statements, logic branches, or logic paths. Such testing is often called "white box" testing because those conducting it—usually, programmers—must know how the software is built in order to perform it.

Statistics-Driven Testing

This testing determines how well the software product satisfies the customers' or users' need for trustworthiness. Customers are often unsatisfied with knowing the results of the two previous goal-driven approaches because those approaches fail to address what the customer really cares about—to what extent he or she can rely on it. This kind of testing is a primary focus of approaches such as "cleanroom" testing, which chooses test-case inputs on the basis of a random sampling of the typical usage profiles for the product.

Risk-Driven Testing

This testing wards off the most serious problems the software might encounter. It identifies potential software risks—things most likely to go wrong—and constructs tests to determine whether the software product is vulnerable to those risks. Such testing is vital for software requiring high reliability.

How Much Testing Is Enough?

There's more to this testing classification than just a set of goal-driven approaches. For example, these approaches should be considered in a prioritized hierarchy, as I said in that ancient book. All software requires 100 percent requirements-driven testing; it's the first level of attack that testing should include.

The next goal in the hierarchy would be 100 percent structure-driven testing, except that it's impossible for the typical software product. Researchers have shown that structural testing at about the 85 percent level is the best we can normally hope for. We must manually inspect the remaining obscure and elusive pieces of software structure, those same researchers add.

Now, in the best of all possible worlds, that would be enough testing. If the requirements are all met, and some impressively high level of the structure is working correctly, what more could we hope for? A lot more, experience tells us. For example, even 100 percent structure-driven testing isn't enough. If pieces of structure are missing or if combinations of structure cause faults when the individual pieces don't, errors can still slip through the requirements or structure screening. How often? When writing that ancient book, I found that as many as 75 percent of software errors were due to missing logic or combinations of structure. (Are you getting a sense of déjà vu? I discussed these findings in my column on testing and the test-coverage-analyzer tool in the July/Aug. issue.)

The bottom line is that software that must be thoroughly reliable needs much more than requirements-driven and structure-driven testing. That's where statistical and risk-driven testing come in. They supplement the belt and suspenders of the first levels of testing. (However, other testing approaches have their advocates. For example, some folk see statistical or risk-driven testing as the first and primary approach, not a supplemental approach as I've called them here.)

The Other Axis

We're far from through here. I've described only one axis of my testing classification matrix. The other, horizontal, axis involves "phase-driven" approaches. It addresses the issue of when, during testing, we apply the four goal-driven approaches I just discussed.

Remember my warning that describing my testing classification system in that ancient book took three dozen pages? What we've seen here is enough for this column, I think.

My next column will deal with the phase-driven approaches. I'll describe (he said, trying to create suspense) how these goal-driven approaches play across the three phases of software testing: unit testing, integration testing, and system testing. Stay tuned!

Please forgive what might appear to be self-promotion in this column. I'm as offended as the next person by people who hype their own work. In this case, I offer three excuses. First, this discussion fits nicely into *Software*'s 25th anniversary issue. Second, *Building Quality Software* is out of print, and citing it won't earn me a dime. Finally, I really believe in the value of that ancient, unnoticed work.

Robert L. Glass is editor emeritus of Elsevier's *Journal of Systems and Software* and publisher and editor of the *Software Practitioner* newsletter, and is a visiting professor at Griffith University, where he's affiliated with the Australian Research Center for Complex Systems. He likes to tell people that his head is in the academic end of computing, but his heart is in its practice. Contact him at rlglass@acm.org; whether you agree with him or not, he'd be pleased to hear from you.

Copyright and reprint permission: Copyright © 2008 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US copyright law for private use of patrons those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Dr., Danvers, MA 01923. For copying, reprint, or republication permission, write to Copyright and Permissions Dept., IEEE Publications Admin., 445 Hoes Ln., Piscataway, NJ 08855-1331.

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1314; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 1730 Massachusetts Ave. NW, Washington, DC 20036-1903. Subscription rates: IEEE Computer Society members get the lowest rate of US$49 per year, which includes printed issues plus online access to all issues published since 1988. Go to www.computer.org/subscribe to order and for more information on other subscription prices. Back issues: $20 for members, $136 for nonmembers (plus shipping and handling). This magazine is available on microfiche.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855-1331. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.

Authorized licensed use limited to: University of London: Online Library. Downloaded on July 12,2021 at 08:24:30 UTC from IEEE Xplore. Restrictions apply.