Through a Glass, Darkly

Test-First Approaches: Are They Harmful?

Robert I. Glass

I've always felt sort of ambivalent about "test-first approaches" to building software, a big thrust in the Agile programming movement. My general feeling was that it seemed superficially like a good thing to ask for, but because I faintly envisioned some flaws in the approach, I bit my lip and kept silent on the matter.

But recently, a fan of my book *Facts and Fallacies of Software Engineering*, who liked what he saw there and was contemplating using it as the basis for the team of software folk he had recently been put in charge of, wrote to me and ask how I felt about that test-driven" idea. And for the first time, I forced myself to come to grips in depth with what I thought.

Here's what I said to him. Some of it surprised me as I opened my electronic "mouth" and listened to what flowed out of me!

Regarding "test-first" approaches, I am slightly negative on them. The fundamental idea is good. But one of the things I've always been concerned about with testing is the possibility that the software is written to make the test cases succeed, rather than to achieve its required functionality, do its job. That is, the software is "trained" to pass the test cases more than to meet its requirements. And, of course, if that happens, then the test cases are a negative influence, because lots of faults of various kinds, especially those regarding meeting requirements, may slip through the resulting testing sieve.

My experience in testing is that some things always come up in development that cause me to write specific test cases driven by unanticipated pieces of the code, things that I never would have thought of before writing it. Also, tests driven by the need for "test coverage" are impossible to write until the code exists. So I think one needs to be careful. Perhaps some tests, and certainly a lot of the testing philosophy/planning, can be done up front. But that doesn't/shouldn't end the job of creating tests.

Since I wrote that response, I've taken the opportunity to think about it some more. To be honest, it feels to me sort of unnatural to write test cases before I have designed and written the code. Perhaps that makes me an old fogey, but I really don't get into the solution of the problem at hand until I've thought about designing that solution, and then coding it. The kinds of tests I could write before design/code, in other words, would be kind of trivial but necessary ones. No harm in that, of course, but it's important that focusing on the trivia doesn't excuse us from later focusing heavily on the necessary.

My own testing philosophy involves four goals: what I call Requirements-Driven testing, where tests are created to demonstrate that each requirement has been met (that kind of testing can, to some extent, be built up front); Structure-Driven testing, where tests show that the as-built code does what it's intended to do; Statistics-Driven testing, where the

software is tested against its ability to satisfy a customer (here success is measured by customer-pleasing statistics (like, for example, a high Mean Time Between Failures)); and Risk-Driven Testing, where tests are conducted to see if identified risks have been avoided. A little thought will tell you that most of these forms of testing have to be taken against the as-built product, and therefore are not candidates for “test-first” approaches..

I’m a big believer in test coverage, for example. Although it’s nice to be able to say that 100% of the product’s requirements have been met, experience tells us that such a product can still be terribly full of bugs (for example, studies have shown us that, when expanding the requirements in order to design a proper solution, a requirements-expansion of something like a factor of 50 often takes place. The final product, we see here, may be 50 or more times more complicated than the requirements in the problem statement suggested it would be). One of the most important supplements to requirements-driven testing, in my view, is testing to make sure that all the elements of the as-built program structure are doing what they are supposed to do. And the best way to do that structure-driven approach is best personified by test coverage approaches, wherein we create test cases that test each logic segment of the program (“logic segment” is subject to several different definitions, but we will not go into that level of complexity here). Note that almost by definition this kind of testing cannot be conducted up front.

The same goes, for the most part, for statistics-driven and risk-driven approaches. Regarding statistics, the tests must be about the as-built product in order to mean anything to the customer. And regarding risk, how can we be sure a risk has been overcome until we have the real software product available to demonstrate that it has been. Oh, I suppose we could design some of these test cases up front, even though we couldn’t run them until later, but it seems to me that a high number of test cases **MUST** be written about the final product, and one cannot conceive of those kinds of test cases earlier.

So where do I now realize I stand on “test-first” approaches? That it’s a nice concept, one which is useful in moderation, which could hurt us enormously if we got carried away with it.

(Many of these ideas are derived from the thinking behind my 1992 Prentice-Hall book Building Quality Software, which has been out of print for so long that this cannot by any means be considered a promotional message! Given that the book is by now nearly 20 years old, are my thoughts here hopelessly out of date?)

I’d be curious to know how you feel about this. Contact me at rlglass@acm.org.

.....
“Through a Glass, Darkly,” is a Biblical expression for the unclear way in which we see the world around us.

Robert L. Glass (rlglass@acm.org) is president of Computing Trends, publishers of The Software Practitioner newsletter, and an Honorary Professor of Software Engineering at Griffith University, Brisbane, Australia. He has been active in the field of computing and software for over 50 years, largely in industry (1954-1982 and 1988-2005), but also as an academic (1982-1988 and 2005-present). He is the author of over 25 books and 95 papers on computing subjects, Editor of The Software Practitioner newsletter, Editor Emeritus of Elsevier's Journal of Systems and Software, and a former columnist for several periodicals. He was a Lecturer for the ACM for 15 years, and was named a Fellow of the ACM in 1998. He received an honorary Ph.D. from Linkoping University in Sweden in 1995. He describes himself by saying "my head is in the academic area of computing, but my heart is in its practice."