BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/3237196 http://cacm.acm.org/blogs/blog-cacm

Discovering Bugs, or Ensuring Success?

Finding errors is not the same as making certain a software product works correctly.

Yegor Bugayenko
The Era of Hackers Is Over
http://bit.ly/2LfUci6
May 30, 2018

You gather up your team of testers. You throw them some money. You give them a schedule. And then you sit back and watch them go, tearing through your product trying to break it. They find bugs, report them, find more bugs, report those too. Soon enough, they'll leave you with the perfect product ready to ship without any worries and total customer satisfaction. Right? Wrong.

In *The Art of Software Testing* (2011, https://amzn.to/2J7UFRE), Glenford Myers explains that "testing is the process of executing a program with the intent of finding errors." Testing fails because the intentions behind the task are very often misplaced. Finding errors is not the same strategy as making sure a product works. Instead of focusing on whether the product functions under parameters, the main focus of a testing team must center on discovering bugs. This "destructive, even sadistic, process," as Myers calls it, focuses on breaking the product, looking for various inputs to crash under stress.

However, most testers feel like they're helping ensure the product's success when, in fact, they're missing that component altogether. Myers notes this misguided focus, stating, "You cannot test a program to guarantee that it is error-free." Software by its nature has an unlimited number of bugs. Boris Beizer said in *Software Testing Techniques* (1995, https://amzn.to/2N1yOhg): "The probability of showing that the software works decreases as testing increases; that is, the more you test, the likelier you are to find a bug. Therefore, if your objective is to demonstrate a high probability of working, that objective is best achieved by not testing at all!" Most testers fail to understand this. They tend to see it as a neatly packaged product that only needs a bit of polishing to run smooth.

The real number can be limitless. No matter how big or small, simple or complex, old or new a product is, the potential for bugs is astronomical. Myers underscores this, arguing that "it is impractical, often impossible, to find all the errors in a program." Even with limitless time and funding, testers cannot find all the bugs. Bill Hetzel in his book *The Complete Guide to Software Testing* (1993, https://amzn.to/2m6f5BM) wrote, "We cannot achieve 100% confidence no matter how much time and energy we put into it!" William E. Lewis in his book *Software Testing and Continuous Quality Improvement* (2009, https://amzn.to/2KUUsXg) even calls this a "testing paradox," which has "two underlying and contradictory objectives: to give confidence that the product is working well and to uncover errors in the software product before its delivery to the customer." If this is the case, then what do you do?

There has to be a certain point where testers stop looking for bugs. Meyers points out that "one of the most difficult questions to answer when testing a program is determining when to stop, since there is no way of knowing if the error just detected is the last remaining error." Finding bugs motivates testers, and they'll keep looking for them. At some point, you have to launch the product. But what happens, though, if you launch a product before you find all the bugs? If you do that, then won't you launch it with bugs? Yes!

Despite all that work, all that money, all that effort, you still launched a program riddled with bugs. It seems rather pointless. Your product is out there flawed. Filled with bugs. But you have to ask yourself, How many critical bugs remain? You could have provided your team more time to complete the impossible task of finding all the bugs, but would they have found more critical bugs?

It is better to launch a product that you have confidence in than waste time and resources trying to make it perfect. Quality control will always find itself pressed hard against the deadline, but there are solutions you can take to make sure testing benefits the product. Instead of allowing testers endless time to find errors as they tear apart the programming, give your testers a goal. Meyers notes that "since the goal of testing is to find errors, why not make the completion criterion the detection of some predefined number of errors?" This enforces the need to find bugs, but limits the total amount and draws focus toward critical bugs rather than general ones.

Once testers pass that marker, you then have clear confidence the product will successfully launch. "Software is released for use, not when it is known to be correct," David West points out in *Object Thinking* (2004, https://amzn.to/2J6jyNT), "but when the rate of discovering errors slows down to one that management considers acceptable." At some point, there needs to be a line, a limit, a goal. If your testers lack a goal, then they end up wasting time and money finding bugs that most likely don't improve the overall quality of the product. Steve McConnell in the *Software Project Survival Guide* (1998, https://amzn.to/2u3Womi) even suggests that "by comparing the number of new defects to the number of defects resolved each week, you can determine how close the project is to completion."

By setting a definite limit for the testers, you guide their targeted approach to product testing with a predetermined goal. This goal helps testers rid the program of enough bugs for it to run smoothly after launch. If you don't do that, you could end up spending unnecessary time and money finding and removing bugs that may not even be a problem. I briefly described this concept in my blog post "When Do You Stop Testing?" (2015, http://bit.ly/2zphq46).

By setting a definite limit for the testers, you guide their targeted approach to product testing with a predetermined goal.

Comments

Testing is good after the fact and needed, but is there an option for coding better in the first place? Are there design-by-contract libraries or lightweight proof tools that can prevent bugs from getting into the code in the first place?

It seems to me there's a heavy reliance in industry on QA teams and developers begrudgingly have moved to writing unit and integration tests (and in some cases avoid writing any tests at all, ditto for documentation of any sort).

Essentially, is there a way to frontload the costs of testing into the design and development so that we can start closer to the ideal error discover rate?

—Rudolf Olah

Testing is essential but not sufficient. Best software is developed/delivered/enhanced with the right set of developers, and not really testers.

Many times developers may not test from end-users' perspectives (not just ensuring correctness of functionality) or might not have developed the product with production-first mindset. This is where a tester can add a great value. I think this blog is more from this independent (blackbox) testing perspective.

Earlier, each developer (a very good programmer) was expected to spend ~20% time in design (includes understanding requirements) and 60% in coding. Rest should be reserved for debug (~10%) and testing (~10%). When introducing innovations/differentiation (and new architecture), you may follow different cycle. The same ratio cannot be extended to a team of developers who are developing a product. What kind of testing they will do? Unit or the integrations I do? Who does end-to-end testing? Also, in such cases developers spend some time in collaboration, integrations, etc. This is where independent testing, testing as a discipline, started.

In today's world, systems are more complex, highly competitive, evolve continuously, and need integrations that challenge a developer's proficiency beyond his or her core skills. Everyone wants to develop and release a new version of their software product/application in a shorter span, but there are very few skills around to deliver the quality and differentiated extended product considering all dimensions, like a product's current status, expected usage, state of integration, and deployment.

Methodologies like Design by Contract, OO, etc., do help and are followed. But again, it is the right set of developers and right technical leader who balance and develop a great product. Beyond that, there are market pressures and resource limitations.

One of the alternatives is to use packaged products that one configures to deliver required quality in a limited time. Unfortunately, packaged products themselves are becoming mammoth. Check a few of the CRM products around, and you will find testing periodic configurations of these products is a big business.

Another alternative is to add extra cycles of testing before and as part of the release of your product ... which is where you get a more independent view of the quality and usability of a product. Sometimes such testers work as part of DevBox testing leading to Test Early, Test Often, and Test Continuously.

Well, ... but this also requires good testers. I feel that is exactly what this blog and references highlight.

So, in summary, you need to balance design, development, and testing as per the product complexity, requirements size/complexity, domain, individual skills ... and also compliance expectations.

All the best.

—Vivek Buzruk

Yegor Bugayenko is founder and CEO of software engineering and management platform Zerocracy.