Features Editor: Dale Strok ■ dstrok@computer.org

"Googling" Test Practices?

Web Giant's Culture Encourages Process Improvement

Greg Goth

In the wider world, Google has become a common verb as well as a noun; you can "google" any person, place, or thing, and more likely than not obtain some sort of information.

But Google might also become a benchmark term for a new wave of improved software-testing practices. Numerous emerging elements, beyond Google's sheer size and cachet as the Web's most-used search engine, could make this possible. For example, Google's development of Web-based applications and toolkits brings the company's code into the far reaches of the public domain. In addition, the company's stance on testing has been made public at a higher level in such places as its testing blog (http://googletesting.blogspot.com) and in more detail in guidelines on testing mobile-phone applications written by a Google employee (www.stickyminds.com/sitewide.asp?Function=WEEKLYCOLUMN&ObjectId=13215&objecttype=ARTCOL).

And, as the predominant software architecture shifts to service-oriented architectures (SOA) and software-as-a-service (SaaS) models, developers in widely disparate organizations will find that their code is interdependent, often in ways the original programmers can't imagine. So, predicting which pieces of code will need to interface will be extremely difficult, if not impossible. Thus, development and testing will have to be reconsidered from a new perspective. Google's adoption of a new approach taking these dynamics into account is well underway.

Early testing is integral

"In the past, we did sort of like two processes," says Mark Striebeck, an engineering project manager at Google. "The engineers did their part with unit and development testing; afterward, the testing folks' path was actually testing the overall application deployed on a production-like or staging environment."

However, in 2006, Google began a more incremental and granular testing approach.

"We tried to merge these two worlds much, much closer together," Striebeck says. "The engineers work directly together with testers, and the testers start testing much earlier and at a much lower level—not just black box from the outside, but really going into the application on the component level."

To accomplish that, Striebeck says, the testers and developers must interact intensively, with the developers creating numerous hooks into their code to enable incremental testing. Google has also introduced practices that institutionalize testing culture. One example is "Testing on the Toilet," in which Google test experts regularly write fliers about everything from dependency injection to code coverage and then plaster them on Google's bathroom walls. Another is Test Mercenaries, teams of veteran engineers who are also testing specialists. The mercenaries are an outgrowth of a "20 percent" project called the testing grouplet, founded by senior staff engineer Bharat Mediratta. (Googlers get one day a week, or 20 percent of their time, to devote to work of their own choosing.) The mercenaries spend three months maximum on a given project and are allowed to test and refactor code as necessary. When they leave that project, it's assumed they have left behind a legacy of good testing practices.

Google has also implemented a "test-certified" program, a multitiered process of practice improvement, which inculcates the emphasis on testing into all the company's developers. Also, in February 2007, Striebeck organized the first Testapalooza, a conference where 800 Google employees worldwide who perform testing shared information.

All these efforts, Striebeck says, are driven by Google's bottom-up, engineering-driven culture. In theory, these elements feed off each other. The Google approach has received notice at development conferences, engendering wider curiosity about whether what works at Google will work elsewhere. Google test experts, in turn, bring back ideas from other enterprises and adapt and adopt those they think will succeed at Google.

Time for a wider discussion ...

Industrywide, however, several experienced testing-and-development experts say the permutations and combinations of testing, development, and quality assurance are still falling short in defect control, cost control, and user satisfaction. The wholehearted embrace of proven testing practices is more fantasy than reality, they say, although there are signs of that changing.

Rex Black is president of the International Software Testing Qualifications Board (www.istqb.org), a global organization created in 2002, and president of Rex Black Consulting (www.rbcs-us.com). Black says when he took over as ISTQB president in 2005, the organization had certified 20,000 testing professionals; by late 2007, that number had grown to 65,000.

Black says the fast growth is symbolic of a desire for change in a portion of the software industry that runs well behind other engineering areas. He says testing as a discipline might lag 25 to 30 years behind programming. The vast majority of testers—up to 90 percent—are ignorant of the practices espoused by pioneers Glenford Myers, Boris Beizer, and Bill Hetzel in works going back to the late '70s.

"We have just not done a good job of building on the foundations that our colleagues in the programming side of the house have done," Black says, "so ISTQB's goal is to help build on foundations already laid. We certainly do not want to be proscriptive, to claim there's only one way; that's not the way we work. We're looking for the lowest common denominator, if you will, and filling people's toolboxes with tools they can use."

Ottawa-based developer and entrepreneur Chris Justus says he encounters more evidence of "seat-of-the-pants" testing practices—both on consulting jobs he contracts for and in stories he hears from other developers—than well-documented instances of quality testing practices. For instance, Justus says his wife worked as a software tester at a large software firm.

"I asked her what the process was for regression testing, and basically they just sort of clicked around in the tool," Justus says. For example, testers weren't required to track new code against written test cases. "That's really stunning," he says. "Between builds there's no way to know as a developer, if I was changing code, if the code was better today than yesterday. It would be like you're just guessing as to the quality of the software, and I think that's pretty common still."

Justus says that on more than half of his consulting contracts, there's no documented regression test procedure. "We're just really at the beginning of people realizing they have to do regression testing," he says.

Justus also says he has been told that another large Ottawa-based company has recognized the necessity for more rigorous testing and is shifting its development model.

"The company has been around for years, doing all sorts of different things, and now they're moving to test-driven development through the whole organization, moving from more of a waterfall model to a more agile development space. And this is not a company writing little 10,000-line programs; it's a company writing multimillion-line programs."

However, one software quality pioneer is quite blunt about his perception of test-driven development.

"You never get quality software with a test-driven process," says Watts Humphrey, the "father" of the Software Engineering Institute's Personal Software Process and Team Software Process (www.sei.cmu.edu/tsp). "You *can* use testing as the ultimate verification and evaluation process, and that works."

The foundational fallacy of test-dependent methodology, Humphrey says, is that "there are an extraordinary number of possible ways you can test a system, and you literally cannot cover them all. So any testing program is going to find a fraction of the defects in the system."

Humphrey says that, in addition to the inherent inability of testing to discover all the defects in a given program, backloading the test procedure can "typically cost five to 10 or more hours per defect. We can show engineers how to find defects at an average cost of six minutes each, which is much, much lower, and you find the same defects. This is what we've found with our teams, that we have essentially found all defects before testing."

Superficially at least, many of the principles behind the PSP/TSP methodology and the Google bottom-up model, such as team ownership of a project and collegial negotiations of project goals between project managers and developers, sound similar. However, because PSP/TSP is a vetted methodology spanning many organizations worldwide, detailed quality data is available (www.sei.cmu.edu/tsp/results/teradyne.html). Google's results are, despite the amount of granular information and advice published, proprietary. Striebeck and Mediratta say they unfortunately can't share information such as

- how many defects they've found per KLOC,
- where in the development process they've found defects, or
- whether the testing-grouplet and Test Mercenary programs have produced demonstrable cost or time savings.

For Google users and partnering developers, ultimate satisfaction might be a matter of intuitive perception plus simple quantitative results. That is, is the application easy to integrate code with? Does it work in a pleasing way?

The vast majority of testers—up to 90 percent—are ignorant of the practices espoused by pioneers Glenford Myers, Boris Beizer, and Bill Hetzel.

As more applications are built and integrated by discrete organizations with various degrees of openness, assembling and disassembling modules as needed, that less-than-granular measure of quality might become more prevalent—at least for public consumption—for good or ill.

… And a wider definition?

Given this new distribution paradigm, Google's position as a leading Web-based applications platform and its embrace of rigorous incremental testing might be the vanguard of a new definition of what software testing encompasses.

“Things are simple when you can control everything in your organization, but adding even one external dependency makes the project five times more complex,” Justus says. “That's where software quality testing needs to be done really rigorously, because you have these downstream impacts that go not just to your immediate partner, but partners downrange.”

The diminution of adverse effects further downrange depends mostly on individual developers learning to design, create, and manage their own smaller piece of a given project. Humphrey says that trend has been sliding inexorably closer to the individual developer throughout his career.

“Fifty years ago, I was the architect and program manager for a big computer system we were developing at Sylvania,” he says. “There was one other guy, a PhD in electrical engineering, who did the circuit design. We had a host of junior engineers and technicians building all this stuff, but I knew the logic and he knew the circuits. We were really the two knowledge workers. Later on, managing one of my early software projects, no one really understood everything. Individual developers were all making detailed design decisions themselves, and that's the essence of software.”

Google's Striebeck points to two particular aspects of development at Google that lend themselves to more granular testing. First, Google uses the same production environment to build, run, and test its software instead of shunting tests onto a separate (and usually less capable) infrastructure.

“That allows us to use the really big clusters we have that we run our applications on—the CPU power, the memory, the bandwidth, everything to make our testing very, very fast and very efficient.”

This raw horsepower is becoming increasingly important for Google as it grows and is critical for the second aspect that's dictating faster testing at Google. As Striebeck says, “Our products are actually highly integrated. It's amazing, if you work with one team, to see how much functionality they use from other products.”

So, Google is building a companywide technical capability to share information about how code from one project interacts with another early during development.

“We realized, when I started here two-and-a-half years ago, individual project testing is good, but with the platform we have, it can't stay that way,” Striebeck says. “We have to have a more integrated system that can test these things, almost test them all in real time. We have some product release cycles that are one week, so if I have to wait three or four days before I know I broke someone else's code, it doesn't work at all for us.”

Perhaps the next great test of how Google's internal quality assurance process works in concert with code from external entities will be the upcoming launch of Google's OpenSocial social-networking platform.

“Our commitment to that is to make sure the OpenSocial framework is stable and reliable and has the features people need,” Mediratta says. “So to a certain extent, you could argue that if each of the contributors across the industry makes sure of their own platform, then the combined effort should be good.”

Google is building a companywide technical capability to share information about how one project's code interacts with another's in development.

The next testing hurdle

ISTQB's Black says he discerns some confusion in the industry surrounding the automation of early phases of unit testing.

“Unfortunately, a lot of that got wrapped around agile, and the common perception is, ‘You only do that if you're doing agile,’” he says. “One thing we're trying to accomplish with ISTQB and in my (separate) consulting company [is that] no matter which life cycle model you're following, it's a real good idea to get bugs out when they're cheap and not on the critical path.”

In addition, as more programming is done by developers who aren't trained in traditional computer science disciplines, testing will have to become simultaneously more rigorous and easier. So, more automation of unit tests at more frequent intervals might be common.

“I'm convinced, long-term, except for a small corner, that software engineering is going to go away as a discipline,” Humphrey says. “It will become a skill that everybody will have. You discover on most of these systems, the knowledge of the domain is so much harder to pick up than the knowledge of programming.”

Humphrey predicts that in the Web 2.0 era, “we'll see a lot of pragmatic management of quality,” with allowances being made for code complexity and the ultimate application for which that code is intended.

“It's really a question of cost/benefit trade-off,” he says. “Sometimes things you interface with are so poorly defined and so dynamic, you don't know until you try it. On the other hand, [on] applications such as weapons systems, you obviously have to do exhaustive quality efforts.”

Black says software quality's “eternal verities” will endure.

“Fundamentals still apply,” he says. “Things like testing throughout the development life cycle and having a brimming tool chest of test analysis and design techniques to apply in different situations. These are concepts that are well known and driven by organizations with a track record of releasing high-quality code, and there's no reason to think those same ideas couldn't be employed in creative ways in SaaS, Web 2.0, or you name it. I'm bullish on software testing, and the way we see the growth of the ISTQB program, I think long-term prospects are good.”