DevelopSense Newsletter *Volume 1, Number 3*

March, 2004

Michael Bolton
mb@developsense.com
http://www.developsense.com

Welcome

Please note: I've sent this newsletter to you either because you asked me for it explicitly, or because I genuinely thought that you would be interested in it. If neither is the case, please accept my apologies and let me know by clicking here, or send a message to remove@developsense.com.

On the other hand, if you like this newsletter, *please take a moment to forward it to friends or colleagues that you think might be interested*. If you'd like to get on the list, please click here, or send a message to addme@developsense.com.

Your email address is just between you and me. I won't give your email address to anyone else, nor will I use it for any purpose other than to send you the newsletter and to correspond directly with you.

Your comments and feedback are very important to me, and I'd love to share them with the rest of the recipients of the letter. Please send them on to me at feedback@developsense.com.

What I've Been Up To Lately: Rapid Testing

I'm delighted to report that in March, I visited Front Royal, VA for a few days of intensive work with James Bach on his Rapid Testing course. As most of you know, James is one of the leaders of the Context-Driven School of Software Testing, and one of the most incisive and original thinkers around on the subject of software testing and quality.

Rapid Testing is about being able to test any software, at any time, under any conditions, such that the information you provide is meaningful and your work can stand up to scrutiny. The course is based on James' philosophy of teaching *skills*, rather than techniques—teaching how to cook, rather than how to follow a recipe. Rapid Testing is particularly useful when you are required to provide valuable feedback when you have been granted less time and less information than you'd like—that is, the typical lot of the software tester. A Rapid Tester doesn't have to waste time griping about insufficient specifications or terrible process; a Rapid Tester uses heuristics, observation, exploration, and critical thinking to find out as much as possible as quickly as possible. As James says, it's the closest thing in the business to a martial art of software testing.

Over the week of working together, we discussed a number of things related to the course and our philosophies of testing and teaching. We developed some exercises and ideas that we're going to develop and expand upon for the Amplifying Your Effectiveness Conference (http://www.ayeconference.com) in Phoenix in November 2004. As a result of this work, James has granted me the privilege of being the only person other than himself qualified to teach his course, which is a great honour for me.

Testing Lessons from Dr. Feynman (I)

At the Workshop on Teaching Software Testing in 2003, a number of us contributed to a list of books and movies that we thought were important for testers to know about. Several of us agreed that Dr. Richard Feynman's The Pleasure of Finding Things Out[1]—and in particular the Appendix to the Rogers Commission Report on the Space Shuttle Challenger Accident—is wonderful book for those involved in software testing and quality.

Richard Feynman was one of the great physicists of the 20th century. He was renowned not only for his brilliance in physics, but also for his ability to explain and model extremely complex ideas in terms that practically anyone could understand immediately.

He was unfailingly curious—indeed, his biography "Surely You're Joking, Dr. Feynman" is subtitled "Adventures of a Curious Character" (there are at least valid interpretations of that subtitle) His determination to (in his own words) "understand about the world", combined with his reputation as one of America's greatest scientists, led him to be chosen in 1986 for the Rogers Commission that investigated the loss of the space shuttle Challenger. In this role, Dr. Feynman demonstrated that the rubber used as a seal in the shuttle's engine behaved aberrantly in cold conditions; at a press conference, he used a glass of ice water and a small clamp to show that the rubber held its shape when it was cold, which turned out to be the lethal flaw in the system.

Some reports of the day mistakenly credited him with finding the problem single-handedly, but in fact Dr. Feynman reported that he was given very broad hints about the problem by another member of the Commission, US Air Force General Richard Kutyna. The general, in turn, was told about the underlying problem by a NASA engineer who wished to remain anonymous.

The Commission produced a lengthy report, but Dr. Feynman felt that the report left out crucial information about the root causes of the problem. In the face of considerable opposition from the Commission's chairman, Dr. Feynman managed to publish his findings in an Appendix to the Commission's report.

While the report is specific to NASA, I've seen several instances in which software organization behaved exactly as NASA did in terms of its managerial and engineering cultures. In developing his arguments, Dr. Feynman also uses some reasoning skills that every tester should have in her toolbox. I'll be looking in more depth at a few of these in the months to come; here's the first.

In the very first paragraph of the Appendix, Dr. Feynman reports that the engineers estimated that one shuttle flight in one hundred would result in loss of life, where NASA managers estimated that the probability was one in one hundred thousand flights. Which number is more immediately credible?

For many people, the mind blurs when numbers exceed certain values. A cluster of 10 is pretty easy to imagine, and I can visualize 100 things by arranging them in a grid, 10 by 10. I just might be able to handle 1,000 as a cube with 10 items per edge, but there's something weird about that—it wouldn't feel like 1,000 items to me. 10 by 10 by 10 is definitely a thousand, but 10 wide by 10 high by 10 deep doesn't seem like a very big pile somehow.

---

[1] Feynman, Richard P., The Pleasure of Finding Things Out: The Best Short Works of Richard P. Feynman (Jeffrey Robins, editor), Perseus Publishing, 2000. ISBN 0738203491

After that I begin to lose my ability to comprehend the numbers without a shift of gears. 10,000 might be the number of seats a small professional hockey arena, and 100,000 the size of the crowd at a concert at football stadium.

Dr. Feynman pointed out the significance of the managers' underestimates by reframing the numbers based on more comprehensible units: he noted that, if the managers' estimates held, one could expect to launch a shuttle *every single day for three hundred years* with only one failure. Tragically, as of this writing, the engineers' estimates seem to have been astonishingly accurate; the first shuttle catastrophe occurred on the 25^{th} mission, and the second on the 133^{rd} mission.

Recognizing the significance of a large number, and being able to place that number in an understandable and useful context, is an important skill for testers. In particular, testers should be able to handle quickly calculations that involve manipulating numbers until they're within the realm of familiar, understandable sizes or time scales.

Skills that we learned way, *way* back in math class can help to chop down big numbers by manipulating them in terms of orders of magnitude. I define an order of magnitude informally as a difference of a factor of ten, or one decimal place, either to the left or the right. We use "scientific" or "exponential" notation to express numbers based on orders of magnitude. (I prefer the term "exponential" notation, by the way; there's nothing *unscientific* about using other ways to write numbers.)

Exponential notation divides numbers into a mantissa—a number between 0 and 10—and an exponent, which multiples the mantissa by 10 raised to some power. Thus in a year, there are 3.65×10^2 days; in Canada, there are around 32,000,000 people, or 3.2×10^7. To multiply using this notation, multiply the mantissas (the numbers before ten-to-the-something), and multiply the orders of magnitude by *adding* the exponents. To divide, divide the mantissas and subtract the exponents.

To make a rough, quick estimate that involves large numbers, render the numbers you're using into exponential notation. How many people in Canada have their birthday today? The answer is the number of Canadians divided by the number of days, easily handled without a calculator by using exponential notation.

- 3.2 / 3.6 is a little less than one.
- 10^7 divided by 10^2 is 10^5

So Happy Birthday will be sung in Canada just under 1×10^5 (or one hundred thousand) times today. In Greater Toronto, there are around four million people—4×10^6 divided by 3.65×10^2, around 1.1×10^4. If we're going to throw a party, we should have 11,000 cakes ready—or we should book around half the seats in the Air Canada Centre for today's birthday boys and girls.

It's useful for testers to consider as many ways as possible to turn big numbers into something that we can relate to. When we see big numbers, how can we reduce them into terms that we can comprehend? Feynman's example is very useful; 100,000 is a figure blurs the mind, but the idea of doing something once a day is something that we can imagine much more easily. The 300-year figure puts the ludicrousness of NASA's estimate into perspective—especially when we compare one shuttle mission per day with the half-dozen or so per year that have been closer to the average.

Here are a few exercises:

- *How many bricks in a typical house?*

- *Ten thousand is a pretty big number. When a Web site is expected to handle ten thousand page hits per hour, how many transactions is that per second?*

- *Is that more or less than ten million transactions per year?*

- *Are these kinds of averages meaningful without a healthy dose of context? For a site that is expected to handle ten thousand hits per hour in normal conditions, what might a peak look like? For one answer, check out what the U.S. Geological Survey has to deal with:* http://www.washingtontechnology.com/news/17_10/emergingtech/18761-1.html

- *The U.S federal deficit in the 2004 budget is projected at $500 billion dollars. How much is that for each working person?*

- *Here's an example using some smaller numbers: when a project manager calls a meeting of 30 technical staff people, and the meeting starts ten minutes late, how much money does that cost the project?*

Estimating, comparing, and manipulating big numbers can be kind of fun. A while ago, when Toronto's SkyDome was under construction, a friend of mine and I invented a unit called the Dome Unit, which measures both the amount of money and the political will required to build a state-of-the-art sports palace—currently around a billion U.S. dollars.

Thus, if we use the National Science Foundation's figures, we can estimate that software errors are costing the United States something on the order of 60 domed stadiums per year. At that time, one Stealth Bomber translated to just about exactly one Dome Unit. The Mars Exploration Rovers, Spirit and Opportunity, together cost something under a Dome Unit.

Another way of looking at the Dome Unit is that each one costs about $3.50 per US citizen, but about $35 per Canadian citizen. Little things can make a big difference, but big things can make a big difference too.

Tip of the Month: Some Tests Should Fail

As testers, we like to see tests pass. Is it ever a good idea to write or run a test that you know is going to fail?

Bret Pettichord (http://www.io.com/~wazmo), one of the saner voices on the subject of test automation, has an important principle in his one-day seminar on test automation: automated tests should fail when the product under test is not installed. This derives from an experience that James Bach had at a customer site, where of the 400 tests that purportedly passed, only 11 of the tests actually ran. How do we test the test automation? A simple way to do it would be to rename the file under test. If any tests pass, there's something wrong with the automation.

Although it's a pretty blunt instrument, renaming a file is a way to get answers to certain questions quickly. If we want to know something about the ways in which a program handles missing or corrupt

files, the easiest way to do it is to rename a file that looks important and see what kind of error message we get back—if we get any message at all. James Whittaker describes renaming the key file for the Content Advisor in Internet Explorer, MSRATING.DLL. In his account, Explorer simply proceeded to allow all content to be displayed on the browser without any notice at all about the missing file. This is still true, although the operating system now restores MSRATING.DLL when we rename or delete it. However, if we start Internet Explorer before the file is restored, we have access to anything on the Net, and there's still no error message in sight. From this we can conclude that IE checks for MSRATING.DLL exactly once, when IE starts up, and doesn't check for it again.

Test-Driven Development (TDD) is an approach to writing code that is used in eXtreme Programming and other Agile processes. The premise of the practice is that you write a test for a new function, run the existing code and watch the test fail. Then and only then do you write the new code such that the test passes. The failing test is a subtle but crucial step in the process; it shows integrity in the test itself. If a test passes and there's no code yet written for it, there's something wrong with that test.

It's useful to recognize that tests should be designed to fail in the absence of success, rather than to succeed in the absence of failure.

Tools of the Month: TextPad

I've been a licensed user of TextPad for several years now. It's a terrific text editor, shareware, from Britain. It was one of those rare pieces of software that had my wallet out within five minutes of using it.

I've never observed a crash or bug or GPF with the product. TextPad is an excellent tester's tool; there is a file comparison engine, and a find-in-files feature. There's excellent support for UNIX and POSIX regular expressions, which makes complex tasks in searching and replacing text a breeze. TextPad also includes a block selection mode, handy for picking up columns of text and manipulating directory listings. It supports customizable syntax colouring, with built-in libraries for dozens of programming languages and extensibility for plenty more. While it's not a WYSIWYG HTML editor, TextPad is as close as a text editor will come to the task. Its online document is first-rate, clear and articulate.

If I find myself sitting at a computer without TextPad, I usually download it within a few minutes and try to persuade the owner to license a copy, especially if the owner is a tester. At US$29, there's no good reason to say no.

Online Resource of the Month

James Bach's site is at http://www.satisfice.com. "Satisfice" is a portmanteau, a word created by sliding two other words together. In this case, the words are "satisfaction", and "suffice", and the meaning of the word is essentially "Good Enough". There are people in the testing business who have decried the idea of "good enough", but it's an important facet in thinking about what we do. Software can be proven to be perfect, and time, budgets, and market forces won't let us get close to perfect anyway. The idea, then, is to write and test products such that they're good enough to satisfy the people who develop and manage and use and buy them. When you think about it, why—and how—could we go any further than that in any sort of meaningful way?

You'll find lots of well-written and insightful articles on James' site. And, since it's James, you'll also find plenty of vigourous challenges to conventional thinking about the work that we do. Highly recommended.

What I'm Up To These Days

For the next few weeks, I'm working on a contract for a large retail outfit, testing a large and complex order fulfillment system. It's presenting some interesting challenges that should inspire a number of important lessons and observations on complexity, visibility, and testability. Stay tuned.

I'm going to be giving a public presentation on "Finding Bugs for Fun and Profit" to the Kitchener-Waterloo Software Quality Association for their April 28th lunchtime meeting. If you're in Kitchener or Toronto area and would like to attend, please drop me a line to let me know—for the Torontonians, I'll have up to three seats available if you'd like to hitch a ride.

At the end of May, my partner Mary and I will be welcoming my first child and her first daughter into the world. The newsletter may be shorter than usual that month! I'll be starting to resume regular work at the beginning of July.

That's it—see you next time!

---Michael B.