DevelopSense Newsletter Volume 1, Number 5

July 23, 2004
Michael Bolton
mb@developsense.com
http://www.developsense.com
Back issues: http://www.michaelbolton.net/newsletter/index.html
Blog: http://www.developsense.com/blog.html

In This Issue

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 Do: Teaching and Practicing Rapid Software Testing

I'm a software tester, trainer, and consultant. I've recently been working with James Bach (http://www.satisfice.com), who is one of the most prominent experts and in the software testing field. I both practice and teach Rapid Software Testing, as it's defined by James. Rapid Testing's purpose is to provide people with critical information about software as quickly and as effectively as possible. This means working under (usually) extreme time pressure, identifying the client's mission clearly, applying expert and practiced skill, and

reporting quickly, accurately, and credibly. It does not mean writing volumes of documentation, following archaic (and often ineffective) scripted processes, or using a particular tool, unless those tasks are crucial to the client's mission. Rapid Testing emphasizes thinking, teamwork, application of skill, understanding of risk, and focused exploration of the product. I apply it in my own testing work and I teach it to others. I'm the only person in the world, other than James himself, who is authorized to teach the class. You can read information about the class at http://www.developsense.com/RapidSoftwareTesting.html.

Because of its focus on exploration, investigation, and fact-finding, the kind of testing and analysis that I do can be used in an extremely wide variety of situations. When a company needs fast, perceptive feedback on its products, I can provide it. Combining my experience as a tester, program manager, and consultant, I can assess the quality of a software product or Web-based service, its development, and its testing, and provide expert opinion--to anyone from test managers to legal counsel--on those subjects. I can also assess due-diligence efforts from both sides of the fence, evaluating both software risks and attempts to mitigate them in ways that are appropriate to the client's context. This is a key point--many otherwise worthy projects (or lawsuits) fail on the basis of poor comprehension and application of the context of the development or testing effort.

I'm also a technical writer. I have written for every phase of the development process, from marketing requirements documents to developer-level technical specifications to end-user manuals. In this role, it's my job to connect the ideas with the audience, capturing and relating the information that people want to impart using language, description, models, and metaphors appropriate to the context of the reader. You can see some examples of my work on my Web site, http://www.developsense.com, and in particular in my newsletter, at http://www.developsense.com/newsletter/index.html. Writing clearly is an important skill for testers. Through my training and experience as a technical writer, I can write articulate, credible reports and argue them effectively.

I have a couple of goals at the moment. I'm looking to raise my profile and expand my network, and I'm looking to provide services in testing (via training, consulting or practise) or writing. With a newborn in the house, I'd like to work from home (Toronto), or near it, as much as possible, but I can be away for a few days at a time if necessary. Any help, advice, or referrals that you can provide would be gratefully received.

Testing Lessons from Dr. Feynman (III): Knowing, Naming, and Bogus Certification

Richard Feynman, one of the great physicists of the 20th century, had the wisdom to choose his parents well. From his earliest days, he was a keen observer of the world around him, and his father, an amateur physicist himself, taught and encouraged scientific thinking. In a speech that was transcribed in The Physics Teacher[1], Dr. Feynman tells a couple of stories about their conversations when he was a boy.

[1] Feynman, Richard P, "What is Science" in The Physics Teacher. September, 1969

In one story, the younger Feynman, about six years old, observed as he played with his wagon with a ball inside. When he pulled the wagon forward, the ball appeared to rush to the back of the wagon. He asked his father about why this was so, and Feynman the elder explained that, first, the ball wasn't rushing to the back of the wagon, but that the back of the wagon was moving forward to meet the ball, which took a little while to get started in its forward motion; and second, that objects at rest tended to stay at rest, and that objects in motion tended to remain in motion. "The *principle* is called inertia," he said, "but no one knows *why* it happens."[2] As Dr. Feynman said later, "He put the difference between *what we know* and *what we call it* very distinctly."[3]

In the same address, Feynman recalled that his father had taught him, "'See that bird? It's a brown- throated thrush, but in Germany it's called a *halzenflugel*, and in Chinese they call it a *chung ling*, and even if you know all those names for it, *you still know nothing about the bird*. You only know some- thing about *people*; what they call that bird.

"´Now that thrush sings, and teaches its young to fly, and flies so many miles away during the summer across the country, and nobody knows how it finds its way,' and so forth. There is a difference between the name of the thing and what goes on."[4]

I thought of these stories recently when someone asked me about certification of testers. Many—perhaps most—of the certification schemes these days are not based upon skill or even upon knowledge, but upon naming things.

Mostly for my own amusement, I recently took two certification tests from a company called Brainbench[5] which was, for a time, offering its tests for free. You certainly get what you pay for. I scored in the 87th percentile in the Software Testing test, and the 97th percentile in the Software Quality Assurance test. Before anyone rushes to congratulate me, I should point out that I view the ratings as specious and the tests as garbage. Let me elaborate.

In my opinion, the tests in no way measure the competence of testers or test managers. I believe the tests to be deeply flawed, anti-intellectual, based on false premises, poorly worded, ambiguous, and peppered with grammatical errors. The tests are apparently intended mostly to measure the test subject's knowledge of someone's bogus notion of equally bogus project management nomenclature. Because of the poor wording, the tests do even that badly. There is no citation for the body of knowledge upon which the subject is being tested (which may be a subtly positive point; most codified testing "bodies of knowledge" are deeply flawed in one way or another, and almost always in their lack of a contextual framework).

---

[2] In these passages, italics and emphasis are my own.
[3] Feynman, Richard P, "What is Science" in The Physics Teacher. September, 1969. Feynman also tells this story on video, in PBS' Nova program "The Best Mind Since Einstein?" originally broadcast on 21 December 1993.
[4] Ibid.
[5] http://www.brainbench.com

Part of the problem is the format. The two tests are each composed of forty multiple-choice questions each. In general, multiple-choice tests are rotten methods for determining skill, but for testing testers, multiple choice tests are particularly bad. Key skills for testers include thinking critically, using imagination, reporting clearly and accurately, and using judgement in context. Multiple-choice tests constrain available answers such that the subject cannot supply answers more imaginative and pragmatic than those supplied. They do not allow the tester to ask questions that would set the context for the answer. Even the most rigid of factory-school testing authorities will begin the answers to many questions with "it depends", an option that doesn't appear in the Brainbench tests that I took.

Exactly one out of eighty multiple-choice questions would, in my opinion, assess an actual testing skill, and that at a kindergarten level for any thinking or trained tester. The questions present no relevant context, and assess no analytical skill that I can determine, expect to the extent that they require test subjects to decode the intent of the test's author.

The tests do provide two benefits: they do provide a rich example on the difference between knowing something and merely being able to name something; and they provide a useful heuristic for me as a hiring manager or consultant—I would automatically and immediately disqualify for employment any tester unwise enough to present the results of these tests as a qualification.

I recorded the content of the tests as I performed them, so I am prepared to present ample evidence for my assertions to anyone who desires it; please email me, mb@developsense.com, if you'd like to receive a detailed discussion. Meanwhile, good luck to any person who tries to defend the tests to me.

In a recent conversation on Jerry Weinberg's SHAPE Forum, James Bach pointed out that certification is really about respect. That means that trusting someone's certification requires you to trust the certifier and their assessments too. *On what evidence do you base your trust in the certifier and the assessment?*

A Handy Trick for Testing for Buffer Overruns

Certain programming languages and computing environments provide input methods that don't defend against a user—unintentionally or maliciously—passing in more data that the program is prepared to handle. This is one instance of a more general kind of problem called a buffer overrun—essentially a problem in which someone, or something, tries to read from or write to memory addresses beyond an appropriate limit. Buffer overruns are at the centre of many of the security vulnerabilities found in Web servers, services, and browsers.

Consider the example of a Web service that uses the Common Gateway Interface (CGI) to receive input. A badly written CGI program might read data directly from stdin—the standard input stream—into a range of memory, or buffer. That read is destructive—the data disappears as the program reads it. Consequently, it's not possible to determine the end (and therefore the length) of the input until the whole stream has been read.

Rewinding isn't possible either. To "solve" the problem, some programmers simply allocate a chunk of memory larger than the largest input they can imagine, and copy the entire `stdin` input stream to that buffer. For an input field on a Web page that might accept up to 256 bytes, a naive programmer might imagine that a 10K buffer could handle any ordinary or extraordinary circumstance. The trouble is that programmers and hackers don't share the same kind of imagination. A hacker can sneak behind the Web page and send the data directly to the application by writing directly to the HTTP port. It's a trivial matter vastly more than 10K worth of data, overwriting other data—or code. This is a common technique for planting code on a system such that the hacker can get control of that system.

A much better solution to the input overflow problem is to use a technique called double-buffering. Read some predetermined number of bytes from `stdin` into a small buffer, perform some range checks, then copy the data into a second buffer, repeating the process until all of the data has been read or until the second buffer is full. If the latter, stop reading from `stdin` and stop copying, on the assumption that there's a hacker knocking or that some other Bad Thing is happening.

Testers with access to the source code can review it to ensure that input is being handled properly. Without access to the source, we can try a black-box approach called an input constraint attack instead[6]. Here I'll show you a little trick to help do this efficiently, and I'll give you an example of how you can use it to find a real bug in an application near you.

The trick is to create a file that already contains a variety of long strings that you can throw at an application. To do this takes only a couple of minutes.

1) Open a text editor. Create a new file called BIGDATA.TXT.

2) Type the following:

```
10
1234567890
```

3) Copy the second line (1234567890) to the clipboard.

4) On a new line, type

```
100
```

5) Paste 10 times (using Ctrl-V). That gives you a string of 100 characters.

6) Copy *that* line to the clipboard.

[6] For more information and background, please refer to James Whittaker's How to Break Software and How to Break Software Security. Also see Joel Scambray's Web Hacking Exposed.

7) On a new line, type 1000.

8) Paste 10 times, giving a string of 1000 characters.

9) Continue until you have a line of 1 million, or 10 million if you're feeling energetic and don't mind the wait when you're opening the file. Save the file.

This gives you a set of nice big strings that you can use, via copy and paste, for an input constraint attack. The idea is identify places that are designed to accept text strings—input fields in a form or a dialog, command lines, and the like. Then paste a big string—a thousand, a hundred thousand, a million, or ten million characters—into the input, step back for safety, and see how the application reacts.

The application may well blow up. If so, you may wish to find the exact amount of data needed to make it blow up. So create strings of the length associated with certain boundaries

10) Between 10 and 100, type

```
15
123456789012345

16
1234567890123456

17
12345678901234567
```

11) Between 100 and 1000, provide entries for 127, 128, 129, 255, 256, 257, 511, 512, 513. Use the 100 and 10 lines in combination with copy and paste to save time.

12) Provide similar entries around the powers of two that 32768, 65336, and 1024K boundaries, plus any others that you might find interesting (99/101, 999/1001, 9999/10001, etc., etc.) Again, use cut and paste with the existing strings to save time.

Here are a few more notes and variations.

- Some input fields might want to see letters without numbers, so use a search-and-replace to create alphabetic versions of these numeric strings; search on "123457890" and replace it with "abcdefghij".
- Note that the last number of each string is modulo-10 the same number as its length. That is, the string that is 17 digits long ends with a 7. For really long strings, this gives you a 90% opportunity to notice immediately when someone or something is truncating your string as you paste.

- James Bach also blogged an interesting suggestion recently[7] that I'll adapt here: use a scripting tool create strings that look like this

  ```
  1********10********20********30
  ```

  where the beginning of each number is the length of the string so far. Creating such a file is a fine exercise for scripting novices.
- Sometimes it's handy for testing purposes to have files that are known to be of a certain size. You can use techniques similar to these to build a library of them.

An Observation on Things That Make Us Smart

I read Things That Make Us Smart a long time ago, and I'm rereading it now. Donald Norman has lots of valuable things to say about design, the display of information, and human factors in software interfaces, so it's a Good Thing for me to review the book every now and again.

This is not intended to be a review of the book, although I certainly recommend reading it. My agenda here is something other than a review. In reading over just a couple of pages of the book (pages 66 and 67), I found these things to be problems.

- There's an asterisk indicating a footnote. Whoops—the footnote isn't there; it must be at the end of the chapter. Nope, it's not there; it's at the end of the book. (I'm with Dr. Tufte; notes should be right next to the text to which they refer. I wish Word, which I'm using for this newsletter, had better support for that).
- There's a change in format—heading and indentation—that is quite inconsistent with the rest of the text.
- In a discussion the way we learn addition with Arabic numerals, Mr. Norman notes that 4 + 5 = 5 + 4 and calls this property "reflexivity", but that's a mistake; the property is actually called commutativity.[8]
- The text cites a reference to a figure that could easily be on the same page, but is on a subject page.
- When I turned back a page to look for the cited figure, I found a figure that wasn't numbered.

I'm not attempting to nitpick. Instead, I'm wondering this: if Donald Norman, a wise and thoughtful man by all indications, can make these kinds of mistakes, what hope is there for the rest of us?

My guesses are that either too few people reviewed and proof-read the final drafts of the book; or that the wrong people did; or that the right people did with the wrong goals.

---

[7] http://blackbox.cs.fit.edu/blog/james/archives/000170.html

[8] Nelson, David, Penguin Dictionary of Mathematics 2nd Edition. Penguin Books, London, 1998. A reflexive relation (not reflexivity) refers to a relation such that for every element in a set, it is possible to use that element as both the subject and object of the relation. The example in the Penguin Dictionary is that division is a reflexive relation on the natural numbers since each number can be divided by itself, where greater-than is not a reflexive relation, since no number can be greater than itself.

One antidote would be for Mr. Norman to recruit more reviewers; another approach would be to hire better reviewers; yet another would be to review the work looking for specific kinds of problems (as long as focus can be reconciled with the peripheral vision to see other kinds of problems).

Napoleon's March: The Minard Chart On-Line

In the last issue of the newsletter, I wrote about Dr. Edward Tufte's one-day workshop, Presenting Data and Information. The course has my highest recommendation. A cornerstone of the day's work is a discussion of the chart by Charles Minard, showing the ill-fated march of the French army, under Napoleon, to and from Moscow.

Friend of the Newsletter Alex Eckelberry of Sunbelt Software tells me that a black-and-white copy of the chart can be viewed at http://it.coe.uga.edu/studio/seminars/visualization/minardmap.html. It's not as large or as pretty as the version you get with the Tufte course, but it gives some sense of the power of the chart.

What I've Been Up To Lately

- In June, Vipul Kocher interviewed me for his Web site, whatistesting.com. You can read the interview here, at http://www.whatistesting.com/. There are also other interviews with some people whom I respect, including James Bach, Danny Faught, and Johanna Rothman.

- I have accepted the position of Program Chair at TASSQ, the Toronto Association of System and Software Quality. The main function of this job is to design the sessions that follow our dinner meetings. Over the last couple of years, these sessions have typically been a single presenter with a single PowerPoint presentation. My goal is to extend that model to include more panel discussions, workshops, and experiential learning sessions. I'd also like to expand TASSQ's functions and events outside of the dinner meetings to working groups, informal get-togethers, and mini-conferences like the Los Altos Workshops on Software Testing (LAWST). I understand that's pretty ambitious, so (from the Torontonians particularly) I'd welcome any ideas or help that you can provide.

- I've been working with James Bach on learning and refining ways to present the materials and exercises in his Rapid Testing course. This feels like some of the most exciting and rewarding work that I've ever done. Rapid Software Testing is a very powerful paradigm,

- In September, I'm looking forward to the publication of an article on Best Practices (and why I would like to see the term eliminated) for Better Software magazine (formerly STQE Magazine).

- I've been setting up a blog, and have made a few tentative entries at http://www.developsense.com/blog.html. One day I'll write an account of the process, describing how to syndicate a blog. That's a task I'll leave until I have a better comprehension of it; suffice it to say that the current state of the art is pretty chaotic and dreadful.

- In my off hours, I've been raising a daughter, eight weeks old as of this writing. If you're looking for a place to exercise problem-solving skills, I suggest that getting a baby will teach you a lot about decoding mixed signals, investigating, and learning how to type with one hand.

That's it for now—see you in the next issue.

---Michael B.