DevelopSense Newsletter *Volume 2, Number 3*

12 December 2006
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

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.

In This Newsletter…

Rapid Software Testing in Toronto

I'm teaching a semi-public offering of Rapid Software Testing in Toronto, January 9-11, 2007 for the Toronto Association of System and Software Quality. That's "semi-public" because it's open to the public at large, but TASSQ members get a $100 discount on the course fee—which TASSQ has already set low to try to make the course accessible to as many testers as possible. Joining TASSQ costs $85 per year, so do the math, and join Toronto's largest testing and software quality community at the same time! More information is available here.

This is likely to be the only time I'll be presenting Rapid Software Testing in an open setting in Toronto until at least June.

Testing Training News

James Bach has recently honoured me by making me the official co-author of the material for the Rapid Software Testing course that he wrote originally and that we both teach. Since late August, we've been working on developing and workshopping a major rewrite of the course. James has most recently been focused on test design and test procedures, and I've been developing work on recording and reporting, especially in relation to the exploratory aspects of testing. As part of this process, we've developed a number of new exercises, and refined approaches to some of the older ones.

The other big news in testing training is that Cem Kaner and James have launched the online Black Box Software Testing course on James' Web site at http://www.satisfice.com/moodle. This is a full-fledged, comprehensive university-level course in the issues and practices involved in software testing. There are more than 40 hours of video, presentation slides, additional notes, self-tests, study guides—all kinds of stuff. This course has a somewhat different focus from Rapid Software Testing; it's a more general overview of testing, where RST is focused on the mindset and skill set that allows you to do excellent testing quickly under conditions of uncertainty and extreme time pressure. The two courses do dovetail nicely, though.

Personally, I think that the course should be nominated for a Jolt Cola award (https://www.joltnominations.com/Jolt/). Nominations are still open for a couple of days—what do you think?

Cem Kaner in Toronto

On October 17, 2006, the Toronto Association of System and Software Quality brought Cem Kaner to town to speak to the organization. Cem gave a presentation titled "Software Testing as a Social Science", in which he talked about the things that he considers to be the big issues in software testing these days. There were several potent observations in the talk, and I'd like to highlight two of them.

The first is that the testing mission is far more complex than it was 30 years ago, when people started thinking and writing about software testing, laying the ground for many of the ideas that we still hold. In those days, said Cem, a typical program was ten thousand lines of code, and everyone who worked on the program could grasp the whole thing, with all of the functions and variables and their interactions. The program ran on a single machine, and the choice of peripheral devices tended to be controlled by the system's vendor. Then he held up his two-year-old cell-phone—not a state-of-the-art device by any means—and noted that its software was probably expressed in something like four *million* lines of code. With support from modern programming languages, libraries, and integrated development environments, developers can now be several orders of magnitude more productive than they were 30 years ago, but the same can't be said for testers. Unit tests—typically and appropriately written by developers—can address a few lines of code reasonably rigorously, so for a small enough body of code, the analysis issues are fairly tractable. But by the time the system has been integrated to include all of its subcomponents, third-party code libraries, operating systems, and peripheral devices, the system is too complex to do the kind of testing that was feasible in the old days, when testing could look more like hard science. These days, testers are faced with complexity that has more in common with human beings than with wind-up toys.

The social sciences have approaches for dealing with these problems, said Cem. Social sciences recognize that our power to understand complex systems is fundamentally limited; that researchers are biased, but that bias can be understood and managed to some degree; that conclusive results simply aren't available, but that the social sciences can obtain "partial answers that might be useful"—a phrase that had me diving for my notebook. Because of the impossibility of complete testing, testing itself provides us with partial answers that might be useful.

Cem's second deeply important point, for me, was that a computer program is not merely a set of instructions that can be performed by a computer. Instead, he said, it's a communication between people separated over distance and time, intended to express the solution of some problem, where the problem can be expressed in code and solved with the assistance of a computer.

To me, this emphasizes that to make progress in testing, we're going to have to become more and more familiar with general systems approaches. General systems thinking allows us to hack away at the complexity of the testing task by modeling and simplifying problems, consciously recognizing that different perspectives will lead to different observations. Knowledge of general systems allows us to take patterns from other disciplines and see them reflected in software systems. In addition, general systems thinking reminds us that a program is only a part of larger systems—business processes, companies and other organizations, markets, societies, cultures, and so forth. A product can't be tested on its own; it has to be tested in the context of these larger systems. This makes the testing job harder, but richer—and means that as individuals and as a profession, we're going to have to continue to develop a very broad range of very diversified skills. That's a big job, but a challenging and exciting one.

Recording Exploratory Testing

One of the Big Questions about exploratory testing is "Without a script, we don't know what's going to happen, so how can we record an exploratory testing session and be sure we're not losing any information?" There are a bunch of answers to that question. The zeroth answer is itself a question, one that we have to ask before we provide all the other answers: "how can we be sure we're not losing information even if we *do* have a script?"

The answer is that we can't, and if we're going to live in reality, we have to learn to be okay with losing some information. We humans aren't able to observe everything, and we also aren't conscious of all the things that we do observe. (Not that machines are better overall. They are in that they can execute tasks quickly and capture information reliably. But they only observe what they're equipped to observe and what they've been programmed to observe, and they're not conscious of anything that they do.) Humans are also subject to a cognitive issue known as in cognitive psychology as "inattentional blindness": when we pay attention to one thing, we're not paying attention to other things, and we're often remarkably oblivious to them although they're right in front of our eyes. Magicians, con men, and software bugs all exploit this phenomenon. Even with these limitations, we're capable of observing and describing a great deal—but would we really want to do that? We could write pages and pages to describe a single dialog in a simple program, but would that information be important?

A test script doesn't record what we did; it simply records someone's idea about something that we *might* do, or that they'd like us to do. It asks us to pay attention to certain things, and at best is mute about other things that we *could* pay attention to. As usual, when we're testing, we don't

observe everything, and we don't record everything that we do observe. That's usually mitigated somewhat by moving our focus around, and alternating between careful, narrow observations and broader overviews. We're more likely to see something noteworthy that way, and if we see something noteworthy, we note it.

Another question that's worth asking is one that I learned from Cem Kaner several years ago: is the record that we make intended to be a *tool*—something to help us remember what we've done, or to help us to organize ourselves? Or is it a *product*—something that we intend to show someone else, or that we need to produce as an essential element in the testing mission?

There's a continuum between products and tools. A tool might be strictly for our own purposes and our own consumption. In that case, it can be as completely free-form as we like—a scribbled note, a quick sketch, a freehand diagram, or even be a digital picture of a whiteboard. In this case, we can take exactly as much or as little time as we need to grab the essential information—the stuff that's important to us, ourselves, to help us remember. If we're not going to share it with others, there's no real reason to follow any standards other than our own.

A tool might have a little bit of product-nature associated with it; that is, it might be a set of rough notes that we might like to share with close colleagues. In this case, if it's going to be useful, it has to be at least readable and understandable by other people—but not by everyone, just by people close to you. Here abbreviations and acronyms will often suffice without explanations or footnotes or formality, since the readers can reasonably be expected to know them. If the information is what's really important, people are more likely to tolerate shorthand, crumpled paper and coffee stains.

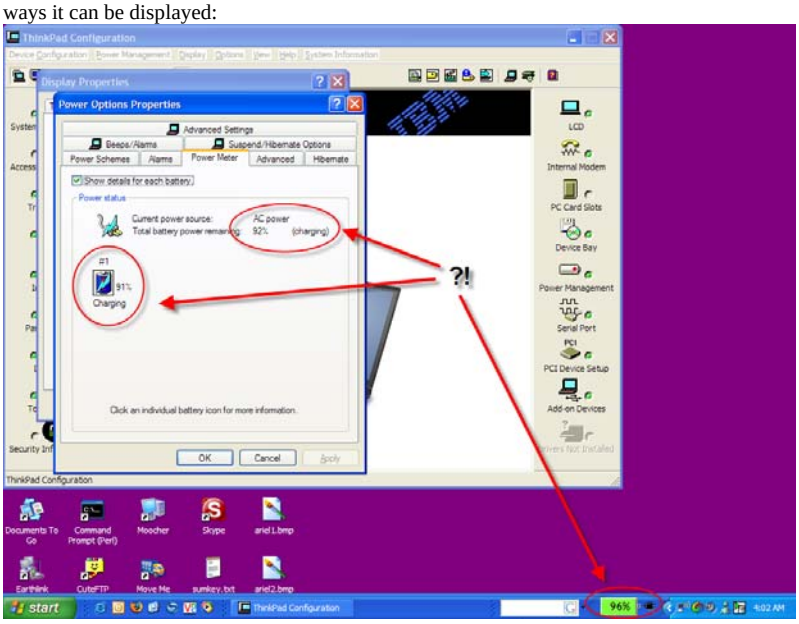
In both of these cases, where the record of a session is mostly a tool to serve my memory, to help focus my thoughts, to list new ideas, to record a preliminary set of observations, my favourite medium these days is the Moleskine notebook, which I described in the last edition of this newsletter: (You can see that article here: http://www.developsense.com/newsletter/DevelopSense%20Newsletter%202006-01.pdf.) But in fact there are lots of other media in which to record an exploratory testing session. I used to like lined or graph paper pads, and in an office setting I still might well use them and put them into a project binder. Graph paper affords more space for bigger diagrams and longer lists than the Moleskine; the Moleskine does better in a coat pocket or a computer bag.

If I'm doing a quick survey of a product, I might be inclined to use an ASCII text file and a text editor as my recording tool of choice. Text files are great for gathering rough notes; they can be pasted into an email or instant message easily; and I find that the sparseness of formatting features helps me to concentrate on the important stuff by removing distractions. That is, since it looks rough to start with, it's easy to stay rough when the content, not the format, is the focus. Really good text editors like TextPad can cut and paste columns of text, such that you can export tidy lists and tables, and provide support for regular-expression-based search and replace, such that you can extract interesting information quickly.

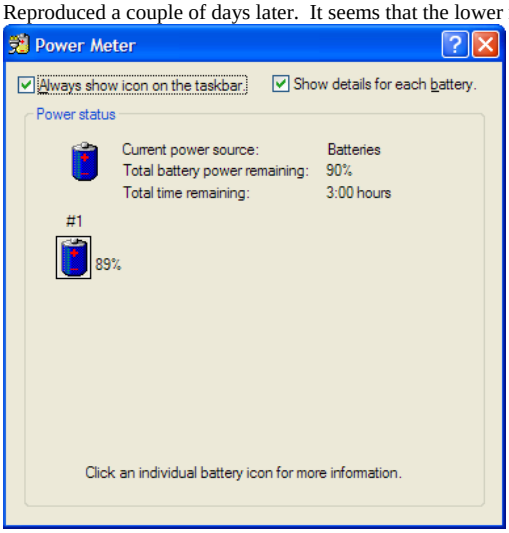
Another approach for documenting an informal exploratory session is to use a word processor document. Sometimes a screen shot gets to the heart of the matter more quickly than words can. I find that I can sometimes move more quickly by entering text to describe what I'm doing, and then interspersing it with screenshots to show what I'm observing on the screen. I use SnagIt, from TechSmith Software to capture the screen, a region, or a window and to annotate the

captured image. If I'm on a system where SnagIt isn't available, the Windows hotkeys are; Shift-PrtScr grabs the entire screen, and Alt-PrtScr captures the contents of a window.

Upon popping up the Power Management , power remaining is inconsistent in all three ways it can be displayed:



Reproduced a couple of days later. It seems that the lower number (under #1) decreases more quickly.



A product is distinct from all this; it's something that we are required to provide for other people as part of the testing mission. This tends to imply more formality and more detail, but that's not always the case. When my notes are intended to be a product—a deliverable of the overall testing effort—then I am most likely to use Session-Based Test Management to document an exploratory session. I find that it strikes an excellent balance between structure and informality. For now, you can read about it at http://www.satisfice.com/sbtm; I'll have more to say about Session-Based Test Management next time.

Time for Better Test Strategy

Back in the early days of 2006, I was proud to add an item to the Project Elements section of James' Heuristic Test Strategy Model (HTSM), one of the key components in the Rapid Testing methodology. It happened that, in the previous year, I had found a number of bugs in a project

that I was testing, and a common element for all of them was *time*. I wrote about this in my Test Connection column in Better Software Magazine; you can read the article on my Web site at http://www.developsense.com/articles/Time%20for%20New%20Test%20Ideas%20(8-5).pdf

The theme of the article is that time is an enormously influential—and often ignored—aspect of many, many bugs, and if only we tested using ideas about time as a basis, we'd find more problems. But there's subtext to the article, too: while the HTSM is a powerful tool, it's not static, and it's by no means the only way to model software strategy. Learning and applying the thirty-six guideword heuristics in the HTSM is a valuable skill; learning how to develop models of your own is an even more valuable skill. Is there a word that you can think of, related to the product that you're testing now or to some greater aspect of testing, that will help you to generate test ideas that you might miss otherwise? If so, add it to the HTSM, or build your own strategy model.

You can read about the Heuristic Test Strategy Model at http://www.satisfice.com/tools/satisfice-tsm-4p.pdf. Many of the Better Software articles on my site, which you can find at http://www.developsense.com/articles, are about categories and items in the model.

I Do Have a Blog

Those of you who have been receiving this letter doubtless recognize that it has become increasingly sporadic as I've been traveling more than ever this year. I've been putting smaller nuggets of stuff up in my blog, at http://www.developsense.com/blog.

I still owe you an experience report on the Exploratory Testing Research Summit—and now one on the Workshop on Heuristic and Exploratory Testing. And the short session that we had on ET at Consultants' Camp. I'll get to it; I promise.

Coming Up…

- I'm doing a day of consulting with senior managers at a corporate client in Bengalooru (formerly Bangalore), India, January 16, 2007.
- I'm doing two days of Rapid Software Test at a different corporate client in Bengalooru
- I'll be presenting a half-day tutorial on Rapid Software Testing on January 18, and a plenary address on Exploratory Testing: The State of the Art on January 19 for the STeP-IN Summit 2007, in Bengalooru, India.
- I'm working with a corporate client on developing their exploratory testing and rapid testing skills in the week of January 23, 2007.
- I'm pencilled in to teach Rapid Software Testing to a corporate client in Los Angeles in the week of February 12, 2007.
- I'm slated to teach Rapid Software Testing to another corporate client in Silicon Valley the week of February 26, 2007.
- I'll be giving three presentations--a half-day tutorial on Rapid Software Testing; a class on Testing Skills That (Almost) No One Talks About; and another class on An Exploratory Tester's Notebook--at the SD West Conference and Expo in Santa Clara, California on March 19-23.

Cheers,

---Michael B.