DevelopSense Newsletter *Volume 1, Number 4*

June 3, 2004
Michael Bolton
mb@developsense.com
http://www.developsense.com
Back issues: http://www.michaelbolton.net/newsletter/index.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.

What I've Been Up To Lately: Edward Tufte

On April 16, my partner Mary and I drove from Toronto to Dearborn, Michigan, to take Edward Tufte's one-day course, "Presenting Data and Information". Mary had been aware of Tufte in her work as a commercial artist and Web designer; I was interested because of the need for testers to present information clearly and compellingly. The lecture was nothing short of phenomenal, and I'm going to remember it and profit from it for a long time to come.

Dr. Tufte's specialty is the display, organization, and presentation of information; the New York Times has called him "the Leonardo da Vinci of data". That's a pretty extravagant claim, but there's no questioning his scholarship. Dr. Tufte believes that people are capable of absorbing information in greater breadth, depth, and quantity than they're usually given it, and he spent most of the day proving it.

The first hour of the presentation was particularly intense. He began by firing out principles—at approximately the rate and intensity of a pitching machine in a baseball batting cage—for rendering, displaying, and presenting data. One of his first points was to emphasize that data is only truly meaningful *in context*; people are interested in bits of data in comparison with other bits of data. Thus he believes that when one is attempting to show something interesting, one should make the display multi-variant—that is, a chart, graph, or visual explanation should present several axes of data, with the goal of providing as much context as possible. The key to that, he suggests, is to provide lots of information on a high-resolution display. Our best computer screens aren't really up to the task, alas;

they're too small and the pixels are too large and too few. Dr. Tufte likes paper instead, especially very high-resolution colour laser printing.

As an example, he used a chart, rendered at 22" by 15", drawn in 1869 by Charles Minard, the French Commissioner of Bridges and Highways. The chart that shows the location and progress of Napoleon's army to and from Moscow across a *two-dimensional map,* including towns and rivers where significant losses occurred, the *path* of the army and its *size* of the army (mostly declining, of course) at each point in *time* on the way there and on the way back, the *temperature* (also declining)—six dimensions of data at once, not including the annotations that provide even more information. The relationships between the data and the events in the chart are direct and unmistakable. Display doesn't get in the way of meaning. Although—*because*—the chart is so dispassionately rendered, it is profoundly moving. The width of the line the depicting the size of the army shrinks relentlessly, and shows with absolute clarity the scale of the tragedy and the folly.

Dr. Tufte's principal crusade at the moment is to contrast this kind of beautiful, evocative work with the cognitive style of PowerPoint and other forms of presentation software (or *slideware*). These programs, says Dr. Tufte, derail narrative, decrease the rate of transfer of information, and trivialize content. Worst of all, PowerPoint is designed for the convenience of the presenter, at the expense of the audience and the content. Dr. Tufte has launched recent broadsides at PowerPoint from the pages of Wired Magazine (http://www.wired.com/wired/archive/11.09/ppt2.html) and in his own essay, "The Cognitive Style of PowerPoint", which is available from his Web site, at http://www.edwardtufte.com/tufte/books_pp. As an excellent example of the problem, Dr. Tufte cites Peter Norvig's viciously funny PowerPoint parody of the Gettysburg address—you can read that here: http://www.norvig.com/Gettysburg/.

There was plenty more. Random examples: a presentation is interdependent with the presenter's credibility; footnotes should appear next to the text that refers to them, "where God wants footnotes to be"; the author of any document presentation should sign it clearly and take credit for it, and the publisher should coöperate; show data in a repeated and consistent way to establish a pattern. He also introduced us to "sparklines"—tiny graphs, approximately the size of words—useful for conveying a lot of information at a glance. There were too many superb lessons to cover adequately in this single very text-y newsletter—and besides, Dr. Tufte recommends that rather than retransmitting notes, enthusiasts should simply point people towards his books. Fair enough; the books are cogent, beautifully designed, and exquisitely printed, and they're a bargain to boot. Better yet, attend his course; for US$320, you get a day of lecture, the Minard chart described above, and three books: Envisioning Information; Visual Explanations, and The Visual Display of Quantitative Information (which Amazon nominates as one of the best 100 books of the 20th century. More information is available here: http://www.edwardtufte.com/tufte/courses

I believe that Dr. Tufte's remarks on presentations have the same kind of value as George Orwell's essay "Politics and the English Language" has for writing. Courtesy of the course, I've been inspired to revise one presentation to good effect (and I hope to apply those lessons to my Web site and this newsletter, too, Real Soon Now).

Dr. Tufte referred to a number of great thinkers throughout the day—Newton, Galileo, Leonardo, among many others—and asserted that their ability to present their data with clarity and credibility was part and parcel of their greatness. One other name kept coming up over and over again: Richard Feynman, which ties in nicely with last month's newsletter and this month's…

Testing Lessons from Dr. Feynman (II)

In the last newsletter, I wrote about Richard Feynman's ability to render big ideas in understandable ways. This time I'd like to discuss another point from the Appendix to the Rogers Commission on the Challenger: the consequences of ignoring and willfully misinterpreting aberrant behaviour in a system.

The Challenger exploded because the O-ring in the solid fuel rocket booster failed, because the material used for the rings was not sufficiently resilient at low temperatures. This problem was well known and well understood by engineers and managers at NASA. After each launch, engineers recovered and inspected parts of the shuttle. They noticed that the O-rings were not intact, as they should have been, but had burned up to one-third of the way through (called "erosion"), allowing hot gases to pass through the seal ("blow-by"). At this point, some NASA managers made a very peculiar decision about this evidence: they created a formula to fit the observations (observations that didn't match with predictions), and then used that extrapolated formula to demonstrate that the shuttle was safe. A rough paraphrase might read "This unexpected thing happened, and nothing bad happened after that, so the shuttle is even safer than we had previously thought." Feynman had the answer to that one.

> *But erosion and blow-by are not what the design expected. They are warnings that something is wrong. The equipment is not operating as expected, and therefore there is a danger that it can operate with even wider deviations in this unexpected and not thoroughly understood way. The fact that this danger did not lead to a catastrophe before is no guarantee that it will not the next time, unless it is completely understood. When playing Russian roulette the fact that the first shot got off safely is little comfort for the next.*

Incredibly, in late 2003, we found that it had all happened again: NASA engineers had seen a problem wherein bits of foam insulation had fallen off the shuttle during takeoff, and managers used bad math, bad modeling, and minimal testing to determine that the problem was No Big Deal. Some of the analysts at NASA further obscured the significance of the data with an inept PowerPoint presentation (which Dr. Tufte analyzes and eviscerates in his class, by the way)—and the world lost another seven astronauts.

These days I'm working on some renovations to our house, and a friend of mine is helping me out. He doesn't have a lot of experience with power tools; he doesn't have experience with *good* tools, and he doesn't have experience with *my* tools. I was working on a ladder in another room while he extracted some screws from the wall. Something on my power drill wasn't working for him, so he adjusted some of the settings, and tried to go back to his task. The drill began making an alarming rat-tat-tat sound, which I could hear easily from the other room. Simply by the noise, I noticed that he was having some pretty serious problems, so climbed off the ladder, and found him trying to extract a screw with the drill's speed setting halfway between High and Low. I was incredulous at first—how could the noise not have meaning for him? After a moment, I realized that, as he was a long-time user of bad tools, negative feedback from a good tool didn't have any particular significance for him. He didn't know what the normal operation of a good tool sounded like, either.

As testers, we are offered plenty of chances to ignore feedback when we test a product that's not ready for prime time. After a while, it's easy to fall into the trap of becoming used to the noise, so we learn to ignore it, or to apply the workaround and move on. This behaviour can easily distract us from understanding the cause of the problem.

We’re usually encouraged—even mandated—to evaluate the severity of failures from the moment that we’re offered something to test. Yet at that stage, we don’t have enough experience with the product to tell the difference between smooth signal and abrasive noise. How then can we reasonably classify the severity of a symptom unless or until we know something about its cause? A bad pointer somewhere in the code can cause a system crash—that’s a severe symptom, and therefore obviously a severe bug. A few moments earlier or a few moments later, exactly the same bug could have a tiny effect, like a misplaced pixel on the screen, yet most of us would probably be tempted classify that as a minor issue.

There are at least a few antidotes, I think. One is simply to listen for noises like banging, distortion, or grinding—or a nice, smooth hum that doesn’t offend the ears. Another antidote is to recognize when we have sufficient information to understand the feedback and when we don’t. I don’t think we can reasonably abandon assumptions, especially not immediately. But when the stakes for our software are high, we should be *attentive* to our assumptions and try to dissociate ourselves from them. Each of us has heuristics, rules of thumb, by which we can determine what we *think* the code is doing, but heuristics are fallible by definition, and they’re not equivalent to understanding. Finally, we need to recognize (and we need to learn how to persuade management) that the noise won’t go away just because we close our eyes, cover our ears, and sing to ourselves.

The testing lesson here, I think, is that if our tests and tools consistently provide us either with noise or with no feedback at all, it will be harder for us to recognize a problem for what it is. A one-pixel problem doesn’t necessarily have one-pixel significance; an occasional burn a third of the way through the O-ring does not represent two-to-one redundancy; a small piece of foam that’s harmless when we toss it is not the same as a large chunk of foam at 500 miles per hour. If we become accustomed to a program’s bad behaviour, we’ll learn to ignore it, and we’ll miss lots of bugs as a consequence.

Tool, Tip, and Book of the Month: Excel, Named Ranges, and Excel Hacks

In my most recent testing project, I found Microsoft Excel to be a remarkably useful tool. I used it in combination with James Bach’s ALLPAIRS utility in order to generate test lots of test data—as many variations as necessary but as few as possible to cover a particular set of variables and specific values within those variables. We then used VBA to plug the test data into some quite sophisticated workbooks that were set up by a colleague. We used some of the sheets in the workbooks as templates by which testers could enter data automatically into the application under test; other sheets acted as oracles, tracking the availability and movement of inventory within a model system. All this stuff gave Excel’s high- and lower-level spreadsheet, database, and programming features a pretty good workout.

Real Programmers don’t use Excel. Or rather, Real Haughty Programmers don’t use Excel; Real Programmers use the handiest and most appropriate tools. However, I believe that we Excel hackers can and should use some Real Programmer principles and practices. One such practice is making the code more abstract, more readable, and more maintainable by using symbols instead of hard-coded values. Excel provides a number of related features that help, but alas, it underexposes them, and as usual the (online) documentation leaves much to be desired in terms of clarity and examples.

In Excel, it’s common to use ranges of cells for tables or matrix calculations. Usually, those ranges are identified in terms of absolute cell locations—for example, $B$5:$J$192. However, Excel allows you to give a symbolic name to ranges of cells. On this project, I’ve found that named ranges save a lot of time and trouble when we add data to the tables.

As an example, suppose that we want to look up a value in a table, based on an index value that we know already. Excel's VLOOKUP function allows us to do this, in the form

```
=VLOOKUP(index value, sheet!range of cells to look in, offset from the index
column, exact match)
```

so a typical lookup, without named ranges, would look like this:

```
=VLOOKUP($C2, 'Inventory Table'!$A$4:$J$654, 5, FALSE)
```

However, you can replace hard-coded ranges of cells with named ranges instead. Press Ctrl-F3 to bring up the Named Ranges dialog. You can thereby give meaningful names to cells and ranges, and you can also give symbolic names to values that would otherwise be hard-coded. The advantages here are many-fold. First, the VLOOKUP function becomes much clearer to read:

```
=VLOOKUP($C2, ItemLookupTable, DescriptionOffset, FALSE)
```

Second, when the lookup table grows, changing all of the references for all of the VLOOKUPs in a workbook becomes trivial—one change, in the name table, covers them all. If you find that you need to insert a column such that the value for DescriptionOffset changes, once again a single change in the Named Ranges dialog fixes everything in the workbook.

Excel Hacks from O'Reilly pointed me to this trick, so it's the Book of the Month. It saved me gobs of time, and it has some truly interesting tips and tricks, most of them quite a bit more sophisticated than this. It's not a perfect book. In particular, it's a little thinner than it needs to be; its examples are good, but the book could use more space for background explanation and exposition, especially of Excel's more arcane functions. Asking anyone to refer to Excel's own documentation is sadistic, and Excel Hacks does that in a couple of places. Nonetheless, the book is worth a look.

An Observation on Outsourcing

On the Agile Testing mailing list, a correspondent wrote recently, "With all the communication means at our disposal, I don't see why a distributed implementation/testing construct would not be feasible. It depends strongly on good will and mutual understanding, the human aspect, but I thought that was the forté of agile development/testing anyway."

I believe that a distributed relationship with a development or testing team is feasible, but ultimately undesirable in most circumstances. In addition to good will and understanding, a relationship also depends on disintermediation, a terrible word that is (in this context) a synonym for "reducing barriers to /real/ communication". Real, human, face-to-face conversation transmits a bunch of information that can't be captured by email or telephone, and that's a serious limitation on the efficiency and effectiveness of the human aspects.

Agility, as I understand it, depends not only upon the speed of feedback, but also on the quantity and quality of feedback. As I understand it, that's why, in Agile projects, Programmer and Customer are in the same room, even at the same workstation, so much.

Outsourcing can be quite inexpensive but only if you're measuring only developers' salaries. Quality of feedback is valuable; the finished product is valuable; intellectual property is valuable; the lessons learned and knowledge gained by developing a product are valuable. Although valuable, these things are difficult to measure. In my experience, when an organization is confronted with something that is valuable but difficult to measure, a common behaviour is to ignore set the value of that thing to zero. If, when, organizations compare the savings in outsourcing with the value of the knowledge and information lost due to outsourcing, the flood will turn to a trickle.

What I'm Up To These Days

I've just finished working on a contract for a large retail outfit, preparing test scenarios for a customer order fulfillment system. The project presented many interesting challenges which will make their way to the

My recent public presentation "Finding Bugs for Fun and Profit" for the Kitchener-Waterloo Software Quality Association went very well in my view. There was an ironic touch. Unlike most demos, I'm actually eager to find new and unexpected bugs in the course of the talk. I did this time, but alas the bug was in the interface between computer and project—the one area where I would have found bugs to be unwelcome. If you'd like me to present this talk for your company, interest group, or other organization, please let me know.

You may remember that in March, I proudly qualified to teach James Bach's Rapid Software Testing course, which I'll be doing when James' busy schedule makes him unavailable. At the beginning of May, I attended a public session of Elisabeth Hendrickson's Creative Software Testing course in Boston, with the goal of qualifying to teach that course for Software Quality Engineering. As befits the course's title, Elisabeth is one of the more creative thinkers in the software testing community, a student of Jerry Weinberg (as we should all be, in my view), and a very pragmatic exponent of practical test automation. Her course reflects the best of her work and ideas, and I'm excited and honoured to be brought on board to teach it. Elisabeth's Web site is at http://www.qualitytree.com/.

Two days ago, my partner Mary Alton and I welcomed my first child and her first daughter into the world—hence the dearth of recent newsletters. We'll see how it goes, but next month's newsletter may be shorter than usual! I'll be starting to resume regular work, testing and teaching, at the beginning of July. Meanwhile, should you be interested, you can check out http://www.developsense.com/ariel/ariel.html, to see what all the fuss is about.

That's it—see you next time!

---Michael B.