DevelopSense Newsletter **Volume 2, Number 1**

January 10, 2005
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.

Teaching Rapid Software Testing in Bangalore

In early October, I took a trip to Bangalore, India, to teach James Bach's Rapid Software Testing course[1] for Hewlett-Packard. It was a very rewarding experience on all kinds of levels.

First, there was a large dose of culture shock. Unless you've been to India, you probably haven't seen *anything* like the traffic in Bangalore. It's one thing to be in a hurry; it's another thing to move nimbly in a crowd, but it's something entirely different to be violating the laws of physics. In Bangalore, there seems to be some kind of weird quantum effect that allows two (or more) vehicles to occupy the same space at the same time without touching each other. All this happens on roads that are subject to a couple of months a year of constant, heavy rain, and that weren't terribly well engineered in the first place. The roads are a big political issue in Bangalore. Even though the technological infrastructure is impressive, successful companies need people to be able to get to work. Public transit isn't up to the task, and even though zillions of motorcycles and scooters congest the roads, relatively few people can afford them. Many companies provide buses for their employees, and some commutes are very long indeed.

The culture shock was less profound when we got into the training room. My observation is that the testers in India were very similar to those in the United States and Canada, both in terms of their skills and in terms of the challenges they had discovered in the testing task. Like testers

[1] See http://www.developsense.com/RapidSoftwareTesting.html

here in North America, they test under uncertain conditions and time constraints. Like testers in North America, they are under pressure to report quickly and accurately, in a way that will stand up to scrutiny. Alas, like testers in North America, they're not often given training in critical thinking and analytical skills. They often believe that they're not able—or at least not empowered—to identify problems other than the ones they've been told to look for, and that make them justifiably nervous. Rapid Testing, both as a course and a practice, is designed to confront these issues, and the feedback that I received suggests that the course was as successful in India as it is here at home.

To its great credit, HP India has active and vigourous training program. The testers that I worked with were exceedingly friendly and thoughtful in class, and we also had very pleasant and stimulating chats during the breaks. Everyone in India was very kind and hospitable. Bangalore in particular appears to be on a roll; there was a powerful feeling of optimism, pride, and possibility in all of the places that I visited. I look forward to the opportunity to visit again.

The AYE Conference 2004

From the 7th of November through the 11th, I attended the Amplifying Your Effectiveness (AYE) Conference in Phoenix, Arizona. This remarkable conference (which I wrote about in the first issue of this newsletter) has incalculable value for me and for the other people who attend it. The focal point of the event is the work of Jerry Weinberg and his colleagues and students.

I was honoured to be a co-presenter for several of the sessions. Three of these I did with James Bach: Rapid Testing; Testing Teasers; and No Best Practices. The fourth was with Jerry himself: Using Your Yes/No Medallion.

The Rapid Testing and Testing Teasers sessions were both composed of some lecture and some exercises from the Rapid Software Testing course. One of the goals of the course is to teach people to become *very good* at software testing, so the exercises are designed to be expansive, and to inspire creative thinking. In most settings, by the end of each exercise, most participants—including the instructors—have realized a few new possibilities in what to test, how to test it, and new ways to think about risks. AYE attracts some very astute thinkers, so the usual tidy flow of ideas became a torrent of white water in the exercises that we led.

I've written about my misgivings with respect to "best practices" before, most prominently in the September 2004 issue of Better Software magazine. The "No Best Practices" session was particularly valuable for me (and I think for James too), in that we received an epiphany on exercises in general. As part of the proceedings, we presented a game that James and I invented and that I've done elsewhere on a couple of occasions: *Don't, Because, Instead*. The game is played in pairs, one player proposes a "best practice"; then the other notes a circumstance in which that practice might be a bad idea, and proposes a different "best practice". The players switch roles; the first player rejects the new "best practice" and proposes another; and the game continues. There are various constraints by which the instructors can alter the flow of the exchanges, but the basic idea is for the instructors to manipulate the game such that a person who passionately argues in favour of a practice eventually argues against it.

The theme behind the exercise is to cure the "best practice" disease by identifying contexts in which a given practice is definitely *not* best and might even be inadvisable. One of the participants noted that during the game all of his attention was focused on thinking of a new practice. He found that the game didn't give him motivation to consider what was good and valuable, in an appropriate context, about the practice he was about to attack. We hadn't intended the game to be an exercise in which we simply gainsay every practice that gets proposed. The AYE participants pointed the way to us refining the exercise so that it would be affirmative, not just negative, which make it a much more powerful thinking exercise.

Jerry Weinberg hired me in advance of the Using Your Yes/No Medallion session to do some role-playing: I played several bosses, each exhibiting some form of pathological behaviour.

Managers sometimes ask us to do impossible things, and saying No to them is difficult for many people. Even if the request isn't impossible to fulfill, it might be more than we're prepared to handle, inconvenient, unpleasant, or an impending disaster; there can be plenty of very good reasons to refuse a request. Yet most of us don't want to disappoint anyone, least of all a boss or a hiring manager; some of us like to think of ourselves as accommodating, or super-competent; some of us are martyrs, and some of us are merely optimistic. Whatever the reason, almost everyone has had trouble saying No in some circumstance.

The Yes/No Medallion is a reference to Virginia Satir's Self-Esteem Toolkit. The Medallion is a talisman representing the ability to say Yes or No (Thank You), and the ability to decide on the spot which is appropriate for you. Through the session, several (very brave) volunteers stepped into scenarios, based on their own experiences, that helped to reveal common issues related to saying No. Jerry then stepped into each role, and demonstrated a more congruent approach in which the No was sent and received in a way that was clear, emphatic, and as graceful as possible. As The Boss in each of the scenarios, I found that forcing the Employees into saying Yes was easy, but I found it impossible to undermine the approaches Jerry used when he played The Employee. To understand more about the Yes/No Medallion, have a look at Jerry's books The Secrets of Consulting and (especially) More Secrets of Consulting.

The real value of the AYE Conference comes from the fact that the sessions are experiential. That provides to us, as participants, rapid feedback into how we're absorbing and applying the material. In most conferences, information is simply transmitted, often without experimentation, participation, or even discussion to back it up. Everyone at AYE is a participant and may become a presenter at any moment; as such, we're all in the centre of the ideas and the centre of the action. The conference receives my highest recommendation.

Two Interesting Speakers

Thanks to XPToronto, the Toronto eXtreme Programming interest group, I've been fortunate to hear addresses from two industry pundits this fall—Joshua Kerievsky and Scott Ambler.

Joshua is the founder of Industrial Logic, a company that specializes in XP. In his presentation to XP Toronto, he added at least two terms to my lexicon. I don't think he originated either expression, but they're certainly useful.

First, he spoke of "software debt"—a term coined by Ward Cunningham to describe the cost of failing to change and update software that definitely needs it. His argument is that the cost of change isn't static; it tends to compound the longer you leave legacy systems gathering dust and cruft. Joshua also used the term "project community" as an all-inclusive handle for the stakeholders, service providers, and customer representatives associated with a software project. I'll be using that term a lot in the future.

At his presentation, Scott Ambler pointed out that a software development project has a lot in common with producing a stage show. That's a metaphor that I've used in the past. In a previous career (a *long* time ago now), I was a theatre stage manager, and indeed there are many similarities between theatre and software. Perhaps the most important parallel is that every production is unique. Even though plays like King Lear and A Midsummer Night's Dream have been produced thousands of times, each production has its own distinctive character, design, pace, and performance. Each is created for a different audience, or customer, and each theatre company is different. In other words, every production has its own context. Finally, good productions typically choose to innovate, often taking ideas from previous but always bringing a new life and perspective to the task.

Good theatre critics—analogous to testers in this metaphor—review each production in terms of its context. Critics must take their own audience—their readers—into account too. It's important to note that criticism in the literary and dramatic world isn't necessarily disparaging or negative. Good criticism is made better by a knowledge and appreciation of the play, its author, the author's literary and historical influences, the theatre company, its audience, its mandate—in other words, the context of the production. Similarly, good testing is aided by an appreciation of the context of the product and the project, and the customer. It's also influenced by the needs and desires of the person to whom you're providing the test results. James Bach has pointed out that, as testers, it's our job to provide information to management; we think critically about software. Like theatre critics, we provide a service, investigating things and giving people information that we believe is important for them to know. Like theatre critics, our reputation depends upon the quality, relevance, and timeliness of the information that we provide to our audience. And like theatre critics, if we don't maintain a good reputation, we'll be ignored.

Getting Jazzed About Oracles

In traditional testing parlance, an oracle is something that provides a correct answer. W.E. Howden provides the definition "any (often automated) means that provides information about the (correct) answer."[2] Some writers in the Mathematical and Factory Schools of software testing often emphasize that an oracle provides a *predicted* outcome of a test.

One of my colleagues, Lois, reported to me about a recent testing project in which test scripts were being assigned to testers. The scripts specified in extensive detail each step that each tester was to follow. They included specific preconditions, inputs, and predicted output values that the tester should observe at each step. Testers followed the scripts to the letter—yet when the product was deployed, hundreds of bug reports came back from the field. What went wrong?

The testers on the project were using too few oracles. Static test scripts aren't usually very useful as oracles. One problem is that scripts are derivative, in that, at best, they simply transmit the result of some other oracle. The bigger problem is that unskilled testers often assume that scripts have the right answer, and that the answer is the only thing worth caring about. No

[2] W. E. Howden. "A Functional Approach to Program Testing and Analysis". *IEEE Transactions on Software Engineering*, 12:997-1005. Quoted in Boris Beizer, *Software Testing Techniques*, *2nd Edition*, Coriolis Group, Scottsdale AZ, 2003

disagreement between the script and the program?—then there's no problem with the program. In this way, the script can help to limit perception, especially in an untrained, brain-turned-off tester. To use an obvious example, if the program produces a correct result, but turns the display upside down, there's a problem regardless of whether the script has anything to say about it. Almost every tester would remark on the problem: the display should be right-side up. That expectation is informed by an oracle that suggests display orientation should be consistently right-side up. Testers sometimes feel less confident in reporting problems that could be more subtle, but more serious. Part of the problem, perhaps, is a narrow concept of what an oracle is and how we validate it.

Testing literature sometimes gives, as example of oracles, spreadsheets that perform equivalent calculations to the program under test, or some part of it; competitive programs, or previous versions of the same program; tables of data; specifications; reference documents; existing tests; and so on. Oracles can be as simple as a pocket calculator or as complex as an entire past version of a program.

These oracles tend to be more or less *algorithmic*. Algorithmic oracles compute a result according to some set of steps or formulas. These are relatively explicable and understandable. In my experience, when people talk about oracles, they usually focus on algorithmic oracles.

James Bach has given me a more expansive definition of "oracle": "*an oracle is a principle or mechanism by which we determine whether something has a problem*." The traditional interpretation—oracle as spreadsheet—speaks to oracles as *mechanisms*, rather than principles. When we consider oracles as principles, an oracle can be *anything* that gives an expected result to which an actual test result can be compared. An oracle need not provide the result of a calculation; an oracle can be anything that alerts us to trouble. Ideas, not just reference programs, can be oracles.

As with practically everything that governs testing, oracles are *heuristic*. As such, oracles are neither infallible nor complete. They're provisional, designed or used to aid in answering a specific question about the product, with the goal that the answer will help to us learn something. Note that oracles provide "*a* right answer", rather than "*the* right answer". Oracles have a context: when they provide a correct answer, that answer is correct according to *someone*, for *some behaviour* under *some set of conditions*. No oracle will be able to tell you if the entire program is working perfectly. All oracles are limited to observing some subset of the product. Moreover, oracles don't have to be predictive; they can just as easily be retrospective. "Wow—I didn't expect that to happen!" is a statement that has an oracle lurking behind it.

A good tester will have dozens of oracles at work simultaneously as she tests, most of them in her head. As an example, when I click on a button, one oracle suggests to me that the button should appear to be depressed; another might suggest that, while I'm waiting for the result, an hourglass should be visible; yet another might suggest some reasonable time for the response—reasonable within the application's context. More oracles suggest things about the display of the result, consistency of behaviour, spelling, and usability issues. In running the test, I could have observed the right answer according to the script, but if my brain is switched on, I'll also observe problems with any of the things above.

Whatever your expectation, an oracle informs it. The tricks are to be conscious of the oracles that you're using, to use those oracles to identify problems, to expand continuously your library of oracles, and to be able to justify the use of those oracles in your context. Diversity and learning rapidly from experience are important. Good tests inspire more oracles; good oracles inspire more tests.

In the next newsletter, I'll write about some attributes that strengthen or weaken your oracles.

What I've Been Up To Lately

Through the fall, I worked on an interesting project, testing the foreign exchange component of a retail application for a major Canadian bank. That project, incidentally, involved generated a powerful algorithmic oracle, using all kinds of fascinating, new-to-me Excel features. The spreadsheet, VBA, and array formulas all got a good workout.

I did presentations on testing for XP Toronto (Toronto's interest group for eXtreme Programmers) and the Kitchener-Waterloo Software Quality Association.

I also co-hosted a session with my colleague Fiona Charles for TASSQ, the Toronto Association of System and Software Quality. We presented a workshop that we called Ask TASSQ, in which attendees at the meeting became clients and consultants to one another. I'll likely write about it in a future newsletter.

As noted above, I visited India in October, and I hope to get there again this year.

My partner Mary and I have been continuing to raise a daughter, now seven months old as of this writing. It's a lot of work. It's tougher when Daddy has pneumonia; I did that too, pretty much all the way through December.

My article on Best Practices (and why I would like to see the term expunged from the lexicon) was published in Better Software (formerly STQE) Magazine this fall. Right around now, I look forward to the publication of an article, also in Better Software, detailing the practice and value of exploratory testing. I've also been honoured with an invitation to become the testing columnist for the magazine, an invitation that I have accepted.

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

---Michael B.