December 2006 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

LET'S DO THE NUMBERS
The annual salary survey results are in!

SEMPER FI
Lead like a US Marine

Changing The Hand You're Dealt

Better Designs Through Problem Redefinition

Rock, Paper, Scissors

by Michael Bolton

The specification was clear: "Use simple rounding for transactions where the calculated exchange rate results in a difference of exactly one-half cent." Yet for certain values—$410.2050, for instance—the software rounded the result down to $410.20, instead of up to $410.21 as I would have expected. I wrote myself a note. At lunch, I chatted with Chris, a tester of financial systems. "Rounding down on a half-cent? Look up 'significance arithmetic' on Wikipedia.com to get started," he said and grinned.

After lunch, I looked up "significance arithmetic" and found the "round-to-even" rule, also known as "bankers' rounding." Banks use it because rounding up tends to favor one party over another. The code was probably right, and the spec was probably wrong. At the next morning's Scrum, I explained my observation, asked if anyone had heard of bankers' rounding, and received blank looks. "I think the foreign exchange module uses it—could someone check that out?" Eric, the development lead, agreed to look. He didn't see anything unusual in the code—it used a normal rounding function. Later that day, he spoke to Dale, a developer in another group who had worked on the original module. "Oh yes," she said. "Visual Basic's round() function uses bankers' rounding. That's intentional. Look it up in Microsoft's knowledgebase." At the following day's Scrum, our customer representative agreed to check with our three largest customers. The day after that, he confirmed that the software was right. We updated the functional specifications.

In all my consulting work, I've never heard someone say, "We're completely happy with our requirements documents, and we would do nothing to change them." I do hear that documents are incomplete, outdated, self-contradictory, or just plain wrong. In many of the testing organizations I visit, people ask for help in improving their test design, saying that they've been relying on requirements documents, functional specifications, or use cases as the sole sources of their testing ideas. At least, that's what they say. In fact, I observe that they use other sources of information, too, but not necessarily consciously.

Skilled testers strive to recognize these other sources and use them effectively for test design, because the requirements aren't the same as the requirement documents.

James Bach says that requirements are a conversation between what we want and what we can have. He says we develop our understanding of requirements by considering *reference*, *inference*, and *conference*. The process isn't linear; we use references, draw inferences, and join conferences in no particular order. At any time, just as in a game of Rock, Paper, Scissors, an inference can challenge a reference, which can be confirmed by a conference and then overruled by another reference.

Reference

References are things that we can point to—documents, mock-ups, prototypes, or models. They may contain tables, pictures, diagrams, maps, and so on. References can refer to other references ("This product shall use the protocols identified in RFC 123666") or to other artifacts ("Version 16 of this product shall perform in a manner substantially similar to Version 15, except where specified here").

References may be helpful, but they're never perfect. They're heuristic devices—fallible methods for solving a problem—that represent an attempt by some person to capture in some medium some ideas that someone had about what someone's requirements are or were. Requirements have changed on every project I've worked on, while requirements documents changed later, if ever.

References are always incomplete to some degree. A complete reference to a product would have to describe the system—everything germane to the use of the program in its business context. Some testers and managers demand a lot of reference material, perhaps believing that the testers are inexperienced or unable to use observation and judgment. This is insulting to the testers if the belief is wrong and a serious risk if the belief is right. Experience, training, and mentorship can help to address the problem, but thicker specifications won't go far to make up for poor testing skills.

Inference

Skilled testers build on references by using inferences to refine requirements, aid test design, choose oracles, and identify risks. Inferences come from experience, systems thinking, heuristic models, and critical reasoning. From experience, we infer that clicking on a close button will close a window. Our general models help us to infer that there will be numerous users with different goals and temperaments and that there is risk in failing to fulfill the goals. When we read the references critically, we can infer that an author might have left out something important to a constituency other than his own or that he might have been mistaken or misinterpreted. Paradoxically, inexperience can be helpful; naïve inferences can lead to interesting test ideas since many of the product's users may be naïve about it too.

Diagrams present great opportunities to build inferences about risk. When we see a diagram, we can certainly pay attention to what it tells us, but we should infer that its author is intentionally leaving out far more information than the diagram shows. Point at any diagram and ask, "What if the data coming in here were bad? What would happen if this connection weren't available? What error checking happens in this process? What would happen if this component were replaced by a 'compatible' update? How many of these transactions can this unit handle over a given time? What if we overloaded it?"

When we test, we compare the program, our references, and our inferences. As we operate, observe, and evaluate the program, we generate more inferences that aren't discussed in the reference or that identify possible errors in the reference. Perhaps since the reference was produced, some specified requirement is now insufficient to satisfy the customer. We may observe that the program's behavior disagrees with an inference, but then check the reference and decide to reject the inference. Maybe the program's behavior disagrees with the reference. That's a bug, assuming that our references and our inferences agree—but maybe they don't. So how do we decide whether there's a bug or not?

Conference

We work out the decision in conferences. Conference is the process of exchanging and refining understanding about the system between two or more parties. We discuss inferences, references, and observations, generally with some person in authority—a program manager, customer representative, development manager, or business manager (some organizations may combine these roles). Conference is a conversation—a chat face to face, on the phone, in an instant-messaging system, or in an email. Like inference, the flow of conference is typically exploratory; we learn outside of a predefined formal structure. Agile processes take extra advantage of conference by minimizing written documentation in favor of working software negotiated with an onsite customer. That reduces the need for reference, exploits inferences, and makes feedback time short.

Unlike a game of Rock, Paper, Scissors, though, one option ultimately prevails: conference with the person who matters most. A reference is simply a stand-in for some person, and an inference is some person's conclusion. We may disagree on what constitutes a bug and whether it should be fixed, but the game is decided by the project owners. That's why they're paid the big bucks. **{end}**

Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries as part of James Bach's Rapid Software Testing course. He is also program chair for the Toronto Association of System and Software Quality. Michael is a regular contributor to Better Software *magazine. Contact Michael at mb@developsense.com.*