In *The Art of Software Testing*, Glenford Myers asserted that “...the most important considerations in software testing are issues of economics and human psychology” [6]. In fact, the most important considerations of any software development practice are (or should be) issues of economics and human psychology. Particularly psychology.

The challenge in testing systems is that testers are trying to develop a way to find out if they don’t know that they don’t know something. This is equivalent to a group of scientists trying to devise an experiment to reveal something they are not looking for. It is extremely difficult to do. In fact, as Thomas Kuhn pointed out, the image we have of the scientist boldly going into uncharted territory and finding out things we never knew is at odds with the reality [5]. Scientists almost always find out things they already know. The hypothesis must come before the experiment to confirm (or deny) it. In fact, it is almost routine for scientists to ignore results that get in the way of their preconceived notions and carefully constructed intellectual models.

It is not possible to be wholly deterministic about testing since we don’t know what to be deterministic about. Testing, probably more than any other activity in software development, is about *discovery*. In the bad old days, people were sometimes punished for finding defects, since defects were considered bad. In my previous column, I pointed out that even the word “defect” is a little, well, defective [2]. By the time we get around to dynamic testing there may be things we should have found out earlier but didn’t due to some negligence on our part. However, exposing things we didn’t know we didn’t know by dynamically executing the knowledge contained in a system is not itself a bad thing. We’ve stopped punishing testers for finding defects, though rewarding them for the same has its perils.

Sometimes the most effective and efficient way to find certain defects is to test for them. This does not in any way substitute for good engineering practices and feedback mechanisms such as inspections. Indeed, without such processes in place and working, any attempt at dynamic testing quickly becomes overwhelmed and quite ineffective. But there are certain kinds of problems that are very difficult to identify analytically. Some companies I work with perform enormous quantities of integration testing. They must do this because they have enormously integrated systems and it is very difficult to determine their behavior in operation unless you execute them in some controlled fashion. Simulation is taking over traditional testing in some of these areas, but a
close look at what is actually done in an executable simulation shows it is not unlike testing in the way it is set up and run. In fact, I think the boundaries between a "real" system and a simulator and the execution of "tests" against the real system and the execution of a simulation will become progressively blurred. In both cases the real and the simulation system will represent some subset of the necessary knowledge that has been made executable, which is what, of course, software is anyway.

Testing vs. Inspections?

There is no doubt that inspections can be very effective and must be used if we wish to obtain high-quality software. But despite evidence of the “superiority” of inspections and other quality assurance devices over dynamic testing, luckily we don’t have to choose between them. We can have and we need both effective inspections and effective testing. No matter how good we get at managing other aspects of development, it is not likely that execution of software systems under controlled conditions for the purpose of determining their validity will disappear in the near future. So, given that testing is probably here to stay, how do we create tests for things of which we are unaware?

10% of Your Brain

It has been said that we only use 10% of our brains. This is not true. While some physical parts of our brains may be more electrochemically active than others at different times, parts of our brains do not simply shut down. All of our brain is active all of the time. The conscious intentional introspective part of the brain, however, has (and needs) only limited access to what is going on. It would be extremely tedious to require our brains to consciously process, say, our pancreatic function in order to make it work. What we call “consciousness” is a very small part of what is going on inside our heads. Our consciousness is that part of our brain function that is self-aware. This capability is inherent to being human, and there is evidence that it differentiates us from other animals. In fact, it is reflected in the very name of the human race. The Latin label for the current version of the human race is not Homo Sapiens as is often cited—it is Homo Sapiens Sapiens. This means almost literally “man who thinks about thinking” (or perhaps man who thinks twice?).

Other animals undoubtedly think, but they don’t appear to think about thinking as an activity. They don’t introspect. The most obvious and well-known type of thought is this conscious intentional kind, but there are other kinds and other levels. Have you ever puzzled over a problem, or wrestled with a worry while driving a car, and found that you have driven, accident-free, all the way to your destination but were quite (consciously) unaware of performing the activities involved in driving? Obviously when we do this we are not unconscious in the sense of being in a coma, otherwise the journey would not have been accident-free. The complex process of navigating, steering, and avoiding accidents was all the way down there with the pancreatic function. There is an enormous amount of processing that occurs inside our skulls of which we are quite unaware at an intentional level. And this capability can be leveraged. Bertrand Russell, the English mathematician and philosopher, asserted he never attempted to consciously solve a math problem. He would read about it, absorb as much information as he could, and then go to sleep. On awakening, he usually found he had the answer figured out. Brains can do that.

So what does this have to do with software testing?

Testing Strategies

Much of what passes for method in testing involves heuristic strategies. We selectively test complex predicate logic, we create test cases that span the classes of inputs and outputs, we construct combinations of conditions, we press the system to its boundaries both internally and externally, we devise weird combinations of situations that might
never occur in the real world, but which we think might expose a so-far unknown limitation in the system. None of these are guaranteed to throw a defect. In fact, nothing in testing is guaranteed, since we don’t really know what we are looking for. We are just looking for something that tells us we don’t know something. Often this is obvious, as when the system crashes; sometimes it is quite subtle.

**The Dual Hypotheses of Knowledge Discovery**

The Dual Hypotheses of Knowledge Discovery [3] are:

- We can only discover knowledge in an environment that contains that knowledge.
- The only way to assert the validity of any knowledge is to compare it to another source of knowledge.

The first hypothesis shows us why, sometimes, we cannot test for and detect defects in the lab. If we cannot duplicate, in sufficient detail and with sufficient control, the situations that will occur in the customer’s environment when we release the software, we cannot expose these defects. Of course, the customer’s environment, not being subject to this limitation, usually has no difficulty in quite publicly demonstrating our lack of knowledge.

The second hypothesis demonstrates the paradox of testing: if I have sufficient knowledge about what is wrong with my system I can create a robust set of test cases and results that will show if there is anything I don’t know. But if I do have sufficient knowledge about what I don’t know, I must *a priori* know it, which means I have already exposed my ignorance and therefore I don’t need to test at all. Testng, it seems, is effective only if we don’t need to do it, and is not very effective when we do need to do it.

**Paradoctoring the Paradox**

How can we effectively address this situation? Our testing heuristics of boundary value analysis and equivalence partitioning help. They point us to the locations of high-density knowledge within our system. We are most likely to make mistakes where complex knowledge is clustered. Where things are complicated we usually understand them less and our ignorance (read defects) is usually higher.

But there is another aspect to consider. I have found good testers have a “nose” for testing. They experience a kind of intuition that tells them what to test and how. A simple example of this in operation can be shown in the layering or sequencing of tests. At the beginning of *The Art of Software Testing*, Myers suggests a self-test to determine your test effectiveness (his phrase). It involves establishing a set of tests for a trivial program that accepts inputs to be used to predict whether the values, if numeric, will describe the sides of an equilateral, an isosceles, or a scalene triangle. Presumably, the program will also indicate if the input values cannot make a triangle at all for some reason. A standard (correct or valid) input test case might be to use the numbers 3, 4, and 5. These numbers, representing the lengths of the sides of a triangle, would produce a right-angled scalene triangle. Assuming the program works well for this input, would it be better to next execute a test for the number set 3, 5, 4? Or how about the number set 4, 5, 6 or 6, 4, 5?

As shown in the table here, the choices in this case are between changing only the order of the inputs, only their values or changing both order and value at the same time. Is it “better” to...
Nothing in testing is guaranteed, since we don’t really know what we are looking for. We are just looking for something that tells us we don’t know something.

change fewer variables (3,4,5 → 3,5,4 or 3,4,5 → 4,5,6) and if so, would changing the value or the order be more effective at exposing something we don’t know about the program? Or would it be better to change more variables (3,4,5 → 6,4,5)? The answer is—it depends. Good testers, as they acquire confidence in the predictability of the system being tested, will gradually increase the number of changed variables. Increasing the variation too soon may flush out a defect, but make it very difficult to find out what caused it to throw. Increasing the variation too slowly results in many more tests, each with an associated effort.

While this example is quite trivial and we could easily test all combinations in a short while, this is not true for larger systems and the rate of test scaling can be critical. Testing systems is always sample testing. We can only run an infinitesimal percentage of the total possible tests. We must somehow extrapolate from the behavior of this tiny sample to the behavior of the enormous whole. Knowing when and how to scale up testing is crucial to good testing. Good testers know how to do this.

I have found that good testers get a kind of intuitive sense of how, where, and how much to test. The logic behind it can be difficult to explain, though we can usually rationalize a decision even if we don’t know exactly how or why we made it. Below the conscious, intentional part of reasoning, there’s a lot going on, and good testers are able to pick up on little hints that direct their testing approach through a below-conscious reasoning process.

Maybe a hint of remembered experience of a similar system, a dash of knowledge of the customer, a pinch of uneasiness about the rushed design phase, and a dollop of knowledge of the capabilities of the developers overlay the testing heuristics good testers have acquired and lead to a sure sense of what will make this system break and expose our lack of knowledge. Tom DeMarco and Tim Lister devoted an entire (but very brief) chapter of their book Peopleware to a legendary team of testers for an anonymous but large computer company located in upstate New York that seemed to have this skill [4]—it is very valuable.

### Dig Deep

In the preface to the first chapter of *The Timeless Way of Building*, Christopher Alexander said (of architecture) “It is a process which brings order out of nothing but ourselves” [1]. He states that people have an intuitive sense of what is right, orderly, and effective. This sense is not particularly logical, intentional, or easy to understand, but it is there. Good testers are able to employ an intuitive reasoning process that is not easy to explain or codify. The same is true of many creative aspects of what people do. The psychology of testing is both interesting and “the most important consideration.”

Sometimes we don’t quite know how it works, but it works.

### References


**Phillip G. Armour**

(armour@corvusintl.com) is a senior consultant at Corvus International Inc., Deer Park, IL and a research director in the Center for Software Development Innovation at Number Six Software, Arlington, VA.