



November 15, 2005

---

## Techniques for Testers

An interview with Dr. Magdy Hanna  
By Meridith Levinson

Senior Writer Meridith Levinson talked with Dr. Magdy Hanna, chairman of the [International Institute for Software Testing](#) about the discipline of testing software and about techniques testers can use to improve what they do. Here's what he had to say about decision tables, state modeling, requirements based testing, communication between developers and testers, and more.

### **The problem with software testing...**

Test organizations still think that testing is an art and they go about it in a very ad hoc manner—meaning, they don't have a systematic process by which to test software. Testing in fact is very close to being an engineering discipline, and I say this as someone who has an undergraduate degree in petroleum engineering.

### **The first of several techniques testers can use to improve what they do...**

There are a number of things that any IT organization can do. If you look at the literature for software testing, it is full of stuff that people haven't utilized. They haven't utilized one percent of it, though it has been used in certain areas like system analysis and design.

For example, system analysts use data modeling to understand the business. Data modeling is in fact a very precise way of testing applications. A data model tells testers which relationships they need to test. Testers can derive their test cases from a data model. The test cases are formed by counting the number of lines and chicken feet [the relationships between the entities] and multiplying by four because every line has four test cases [two test cases for each end of each line].

Data models bring to light the relationships that may be missing from a requirements document, which won't and can't have all the details because they provide a visual representation of a system. So data models supplement requirements documents and can help test professionals ask questions about how a system should work that aren't in the requirements document. Test professionals need to test every relationship in the data model.

### **On decision tables...**

Another very powerful technique people often ignore is called decision tables. When I ask people in my seminars how many people use them, only one person raises a hand. Interestingly, you can learn about decision tables in any book on system analysis and design. Decision tables have been presented in literature as a program design technique. As a result, testers tend to see decision tables as a technique for programmers, but programmers generally don't use them either.

A decision table is a very simple table of rows and columns that shows the factors that an application will use to make decision and then it shows what the decision is. For example, let's say you're dealing with a reservation system for an airline. If a person is a member of an airline's loyalty program and they've paid for a coach ticket and if a seat is available in first class and they've been waiting for an upgrade, then we'll upgrade them. Or, if they meet all those requirements but they're traveling with a companion, we won't upgrade them. It's a very powerful tool.

Decision tables describe in a very precise manner how an application behaves. The columns in the decision table are straightforward scenarios that must be tested. When one column says a person is a platinum member and they have a companion therefore they will not be upgraded, that's your test case. You develop a test that ensures that a person who meets those criteria does not get upgraded.

People often ask me, "How do we know when we're done testing? How much testing is enough?" The decision table tells you how many scenarios you must test and what those scenarios are. The more scenarios we cover, the more predictable our systems will be.

There are two types of decision tables. I'm more interested in binary decision tables, where factors are yes or no, or true or false. For instance: Did a person pay coach? Are they on a waiting list? Do they have a partner? Those are all yes-or-no questions. You come up with scenarios to put in your decision table based on requirements. Once you've created the decision table with all the scenarios, take it back to the business and have them review it. Often you'll find by doing the decision table that you'll uncover additional requirements that the business people hadn't thought of or articulated. You should do decision tables very early on in the application development process.

### **On state modeling...**

Another technique that I have not seen one single test professional use is called state modeling. This isn't academic stuff. It's very practical. Yet there is no single book that speaks about state models in testing. State models are diagrams with boxes and lines between them. The boxes represent different states a system will be in at any given time. For example, [back to the] reservation system for an airline, the states may represent a new reservation that's made, a paid reservation, a cancelled reservation or a ticketed reservation. The diagram outlines all the possible scenarios that need to be tested. Every path on the diagram represents a scenario. You write test cases for each path. This helps to improve testing by showing every possible scenario that needs to be tested.

We use state models the same way we use data models and decision tables: to go back to the business and ask if it reflects the system. They trigger our thinking. We use the model as

basis to understand, revise and define requirements. If you use these models and give them to developers, there's no chance for a developer to misunderstand them or interpret them differently because they use very precise syntax and semantics.

### **On equivalence class partitioning...**

Equivalence class partitioning is another technique. You can read about this in the first book that appeared on software testing, [The Art of Software Testing](#) by Glenford J. Myers. This technique applies to almost 100 percent of your requirements. Equivalence class helps determine valid and invalid values that a program may get and how a program might react to it. For example, if you're developing a loan approval application for a mortgage company, you can use equivalence class partitioning to determine the valid and invalid values for family income and how the application should react to each of those values.

Equivalence class partitioning stimulates your thinking, helps you generate new questions about requirements and helps you see problems with requirements. It also tells you the number of test cases you need to run and the values you need to use in them.

These techniques help you reach consensus with everyone else about how a system behaves. It also tells you what to test exactly in terms of test scenarios and test cases.

### **On the importance of impact analysis...**

There's more to regression testing than critical path, and that is the impact of changes developers make to code and design. When developers make a change to code—even a simple change like the width of a column in a table—they are often not disciplined enough or willing to take the time after they make a change to ask themselves "What other functionality in the application did I impact?" It's mandatory that developers log impact info into the configuration management system when they put code in.

Developers have to make this an essential part of their process. Analyzing the impact of changes can greatly improve the reliability of software. Also, test managers and test leads need to speak with the development manager and request that information: What was changed? What did developers change? What needs to be retested? The test team should also have access to the configuration management system, but that doesn't happen in many companies. Somehow, configuration management is owned by the development team. The test team should have access to the configuration system if only to read what's there.

### **On communication between developers and business people and modern software development techniques...**

Another problem with software testing is the lack of communication between testers and development teams. I've found throughout the years that when the communications lines between developers and the business is wide open on any project, that's when projects start having problems. This is very contrary to what people might think. Everyone thinks the best thing developers can do is work with the business people. New approaches to application development like agile and extreme programming and rapid application development support that. The principle is not wrong, but the way we go about it is wrong. The problem is when this channel between the business and developers opens, the test team is often left out. When new requirements are generated on the fly during rapid application development, those new requirements are often not communicated to testers. They don't find out until the last minute

and then they don't have enough time to test those requirements. Project managers have to see the test team as part of the project team.

The smartest thing any project manager can do on any project is to increase the interaction between testers and developers, and to go lighter on the communication between developers and the business. If they're going to foster good communication between developers and business users, then at least as communication should be developed on other side, between developers and testers.

### **On requirements based testing...**

Requirements-based testing is known to be a good thing because you're testing what the wants. I can assure you that when we test based on requirements; we don't even test 50% of the code. We test requirements as they're written, as they're given to us, but most test teams don't realize that the requirements doc they received doesn't have 50 percent of what developers put in code. Developers don't update the requirements doc. It's not in the process and it's not their responsibility. In my opinion, your ultimate goal for testing should be covering as much code as possible.

Project managers and test managers should try everything possible to eliminate or minimize the gap between what developers put in code and what testers know based on the requirements doc. One thing they can do is use techniques like data models, decision tables, state models, etc. When they use those techniques up front, all the details of an application are fleshed out and everyone should go into the development effort with a more complete picture of what should happen.

### **On the importance of testers participating in code and design reviews...**

Test managers should also encourage their teams to participate in code reviews and design reviews. Testers are not in code-and-design reviews to find problems with code and design, but simply to learn about what's in code and what's in design that's not in the requirements doc. Testers need to be sharp and recognize when something they're hearing in the meeting is not in the requirements doc. Sometimes it can take a bit for this approach to succeed. It takes a good tester. Sometimes people make the mistake of picking testers who used to be programmers. I've found that [approach] to fail, unless the former-programmers-now-testers are good at controlling themselves. Testers who used to be programmers tend to start criticizing design and that irritates developers. Testers have to go into these meetings with the attitude that their learning from developers. When they go in with that attitude, developers will want testers there and will welcome them. Developers have to accept testers in code-and-design reviews and that takes time because developers don't think testers are capable of understanding anything. Education is the key for testers to improve the image of the testing profession and to be able to take part in design and code and requirements reviews intelligently. They have to contribute and show value. No one wants to see you in a meeting if you don't say anything.