One of my former colleagues once revealed the fact that she had no less than 70 pairs of shoes. That's right, seventy! She had been very good at her job and by no means had any plans to start her own shoe business so I asked myself why on earth would anyone need seventy pairs of shoes? Aren't 5 enough?

Well, apparently all this has to do with the art of matching the right shoe for every occasion. Now what's all this have to do with verification, one might ask. Well, it's the matching thing. I admit that matching the right shoe for a particular occasion is something I'm less experienced in, but more often than not I find myself trying to match the right testing approach with a particular feature that I want to verify, and in that sense the more shoes I have at my disposal the more effective my test would be. More specifically, I'm talking about where you position your testing method on a scale whose one end represents the fully random coverage-driven approach and the other - the fully directed scenario-driven approach.

Coverage Driven Verification has become a sacred term over the recent years and everybody wants to go random. Now don't get me wrong here, the coverage-driven approach is a powerful thing and is nothing less than essential in today's complex designs. It's just that sometimes we forget that there are alternatives. Let me give you an example: In the early stages of block verification you would typically look for zero order bugs, right? Just about every scenario you could think of would probably reveal a bug. Let's also assume that your environment supports random generation. Now, would you start off by bombarding your DUT with the heavyweight random tests or by scratching the surface gently with a bunch of predetermined scenarios? Let's think about it for a moment - random testing right from the start could definitely do the job. But wouldn't it be easier to start off with several basic tests with much less randomization in them? Wouldn't it be more efficient in terms of debug time? Probably yes.

You might feel discouraged to maintain a set of directed tests as well as a set of random ones but remember the only cost is redundancy while the gain is a nice little mini-regression suite that you can run whenever you need a quick status on your DUT. Moreover, modern verification environments allow you to generate predetermined scenarios simply by applying sophisticated constraints. So you don't even have to worry about extra maintenance for your directed tests. In fact, when I use the term directed test or scenario driven test I mean exactly that - a heavily constrained test such that will produce a predetermined scenario.

Ok, now that we got that out of the way let's take a look at a case where you want to verify the behavior of the DUT during a specific scenario. The classical coverage driven approach will tell you to go define that scenario in terms of coverage, which might not be an easy thing to do, and then run your random test suite until you fill your coverage goal. But assuming you can generate this scenario with a fairly straightforward directed test, wouldn't it make sense to have a designated test for it? You don't even have to give up on the beloved coverage item - simply define it as a successful run of the designated test (can be done automatically using smart base-classes / prototype units).

Another example relates to the growing number of projects that rely on FPGA emulation in addition to simulation based verification. While this gives extra confidence in the correctness of the design, for us verifiers it also means that our verification environment should be capable of reproducing scenarios that failed in emulation. Now this might be time-consuming, especially if you designed your environment with only random generation in mind. As a matter of fact, even when you're trying to reproduce an old bug found in simulation, sometimes running the same test with the same seed might produce a different scenario because both the environment and the DUT had evolved. So a set of directed tests can come in handy in cases like these. What's more, you will also be able to let a designer take any of your directed tests and fiddle with it, or just use it as a template to add his own directed tests.

I talked before about a virtual scale which represents the level of randomization and the previous examples suggested where you may want to consider using a low level of randomization, if at all. But obviously a lot of times you'll want to use a much higher level. Still, where you position your test on that scale could mean a lot when it comes to debug time. For example, when you generate random packets as stimuli, is it really necessary to randomize the payload? Wouldn't it be enough to constrain the payload to be some sort of pseudo-random arithmetic sequence and benefit from the ease of debug? Probably yes. (You may still want to let a few tests fully randomize the payload as well just to be on the safe side). So we see that sometimes cutting down on randomization may not necessarily have a negative effect on coverage as long as smart generation is applied.

To sum things up, the real trick is to use just the right amount of randomization to fully cover the feature that you want to verify taking into account the maturity of the design and the current phase of the verification process. Typically you will want your verification process to gradually evolve from scenario-driven testing to coverage-driven testing while maintaining a set of semi-random or directed test-cases all along the way even at the price of redundancy. These will serve as a mini-regression suite or cover specific scenarios and corner cases (and it's the only place where you may go wild and have some fun with aspect oriented programming!)

I think we would all agree that random generation is our true power as verifiers. But sometimes we can achieve better and faster results with directed tests. Most of the times we'll just settle for a semi-random approach where only a small set of parameters are being randomized while the others remain constant. But exactly how much randomization should be applied? What is the right amount? Don't worry, as long as you keep an open mind you'll know the answer