There's been a lot of buzz about the UVM lately and for a reason. The Universal Verification Methodology is about to change the rules of the game pretty soon, if not already. That is interesting because not too long ago verification engineers questioned the need for adopting any verification methodology… Today, however, the question is entirely different – it is WHICH methodology to adopt. So as the UVM is gradually taking form and drawing the attention of verification stakeholders I think it's a great time to stop and remind ourselves why we need a verification methodology in the first place, and more importantly - what makes a verification methodology so good that the only question about it would be "how quickly can I learn it"? I have identified 3 elements of a good verification methodology that work together, each on a separate track, to make it do its magic. The first element is Guidance, the second is Efficiency, and the third one – perhaps the most exciting one – is what I call The Real Added Value.

Guidance

To benefit from a good verification methodology you have to pay a price. You must take the time to learn it. But as with all successful investments, you will see return on your invested time and energy once the learning curve is over. Nevertheless, a good verification methodology should be compiled of a collection of relatively simple (yet powerful) principles. Those principles, that should be fairly easy for you to understand, will guide you and provide you with the basic concepts and approaches that form the foundation of your verification environment. For example, code encapsulation rules and topology guidelines. A good methodology should also provide guidance on more mundane issues such as naming convention, common terminology, file names, etc. These will help you keep a good level of uniformity and consistency across your verification environment. Guidance should be given also on the best known verification methods and how they can be implemented. Now the most effective way to deliver all that information to prospective users, is by providing them with many examples and tutorials. Reference manuals are great too, but most of the times a good example or a code snippet can do the job much quicker. A good methodology should also be good at educating its target users.

Efficiency

Obviously you adopt a verification methodology with the hope that it will help you become more efficient. Why else would you bother? Don't worry. A good verification methodology will help you save precious time. How? That's easy. Verification methodologies provide "off-the-shelf" constructs and components that help you avoid "reinventing the wheel". Need a generator? No problem. We've got one for you, just instantiate it and you're ready to go. But not only that, code integration becomes ten times quicker when everybody uses standard elements who all share a common API. This is a powerful thing. Another example of how a verification methodology can help you save time is the Register Package. This kind of code package – that usually ships with any leading methodology today - will help you model and simulate the entire register activity in your chip and save you tremendous amount of time by simply introducing automation to an area where it didn't exist before. Today register packages provide a powerful API and can also run automatic tests for you. So less code to develop, less bugs in the environment and less time spent on code integration all translate into quicker time to market. Another nice thing about sticking to an industry standard methodology is that it's out there, available to everybody else as well. And so, 3rd party developers are encouraged to develop extensions and utilities that can sometime help you become even more efficient. For example - a utility that further automates register modeling, straight from MS WORD.

The Real Added Value

You might not be able to sense it upfront, but adopting an industry standard methodology comes with more value than meets the eye. For instance, If your verification environment is kept XYZ-compliant, then it implicitly promotes reuse! Much has been said about the "price" you have to pay to make your code reusable, but the truth is that if you stick to your chosen methodology, reusability will take care of its own. The components you develop will be reusable in the context of your current project, and possibly in the context of other ones. Another great thing about methodology is that it helps you think about your environment in a top-down manner rather than a bottom-up manner. Your natural habit, especially if you're less experienced in verification, might be to construct your environment from the bottom up which might turn into trouble at the code integration phase. Unless you have a solid methodology, that is. The thing is, verification methodologies help you visualize the overall verification architecture that's needed for your project by showing you how low level components will fit perfectly into the top level puzzle.

Counting on methodology to do its magic, you can safely split the work among different people, some will implement the low level parts, and others will be able to focus on the big picture. You are virtually guaranteed that everything is going to fall into place during integration. Thinking big upfront helps in bringing fundamental verification problems onto the surface very early in the verification process, which is when you want to deal with them. A good verification methodology also provides implicit protection. You don't (and not required to) know how much stuff is going on behind the scene - arbitration, thread synchronization, debug utilities, warning messages - just to name a few. All of that good stuff is usually taken care of for you to help you focus on your most important tasks, and provide you with a robust verification platform. There's yet another big added value to adopting an industry standard verification methodology - and this is one of my favorites - inherent flexibility. You see, the nature of verification projects is such that the more flexible an environment is (i.e. promoting future extensions and tweaks) – the better. Unfortunately, building a robust environment that's also very flexible is not so easy. The good news is that the folks who develop verification methodologies (UVM being the perfect example) are well aware of that. In fact, a great deal of attention is given to this subject in all popular methodologies so once you nail the concepts of your chosen methodology and stick to them, flexibility is pretty much guaranteed. You don't have to worry about it too much. Good examples are the increasingly popular TLM ports and software paradigms such as class factories and callback functions.

Overall – a methodology is a good thing and you should adopt one. The UVM might turn into the perfect choice for you as it is being driven and developed by the people who understand verification better than anyone else on the planet. UVM is all about bringing together the best verification concepts currently available today and unifying them into a single entity.