Industry-standard methodologies are great, really. It would be so nice if our entire verification environment (VE) were OVM-e (eRM) compliant, wouldn't it? But what if there are legacy components in our env that don't follow any specific methodology and we still want to make them speak OVM-e?

What we often want to do is take our legacy driver and monitor (or whatever TCMs that provide this functionality) and make them a nice self-contained *Interface eVC* with a Sequence mechanism and all that stuff. Now the good news is that usually this is not a big problem. So here are Think Verification's 5 easy steps to start using sequences in your env:

1. Redefine your transaction - go back to your original transaction definition and make it inherit from **any_sequence_item.**

2. Get hold of another eVC that's good for your application and extract its skeleton (delete all protocol-specific code but leave all units, struct method headers, interconnection). Alternatively, use a script such as vBuilder to build a skeleton for you. Think Verification strongly recommends changing all file and struct names to be consistent and add a unique prefix to all objects and types (very useful, also required by eRM rules). A small perl utility can help, try this .

3. Fill your monitor unit with protocol-specific content (e.g. implement the *while TRUE* loop in the main tcm).

4. Fill your BFM unit with protocol-specific content (e.g. implement the drive_transaction tcm).

5. Prepare a sample test file which extends the MAIN sequence and generates one or more items to test your new eVC.

And one final tip for writing new eVCs - Always have a *known-to-be-good* eVC nearby as reference!

Voila! Now you have a new eVC and you can send your legacy componets away to the National Museum of Verification History. It’s now time to develop your Sequence Library. Happy Sequencing!