MathWorks' New Methodology Lets *System Engineers* Change Requirements and Verify Design Compliance In Real-Time

This new workflow lets you connect system requirements directly to the design model. Then, you can quickly verify design compliance. . . assess the impact of requirement changes. . . and test multiple architectures and scenarios within minutes!

"Our group produced twice as many requirements, and had 50 times fewer issues per requirement than when we used a traditional methodology"
- Julio Graves, Embraer Commercial Aviation

Dear Reader,
If you're like most system engineers who are developing commercial wireless systems (such as 5G, WLAN, or LTE) . . . Military Comms . . . or Radar & EW . . . you're probably asking these questions every day:

How can I quickly test various design ideas and be sure I have chosen the best ones?
Every idea and scenario you think of must be tested on a physical prototype.
And since each test adds to development time and costs, you end up exploring only a small number of them.

How can I verify design compliance with system requirements throughout the entire development process?

As you are well aware, test and verification are typically done only during the integration phase, which comes later in the development process. This makes it extremely difficult for you to identify and fix errors introduced earlier in the design and coding phases. And finally . . .

How will my design handle a sudden change in requirements?

If even one single requirement changes - your entire system will have to be re-coded, rebuilt, and retested . . . delaying the project by a matter of days, weeks, or even months!
Now, have you stopped to ask yourself why this is so?
The answer is this.

The Flaw in Traditional System Engineering Methodology

You see, these problems are a direct result of working within a flawed system engineering methodology.
It starts with how you are communicating your system requirements to the design teams.

Until recently, system requirements were captured in documents and handed down to the design teams, which led to errors and delays.
(In some cases, there isn't even a specification document, but more of a 'guideline' on what should be developed.)
As a consequence, requirements got "thrown over the wall" - there was no clear or consistent communication between the system engineer and the designers.

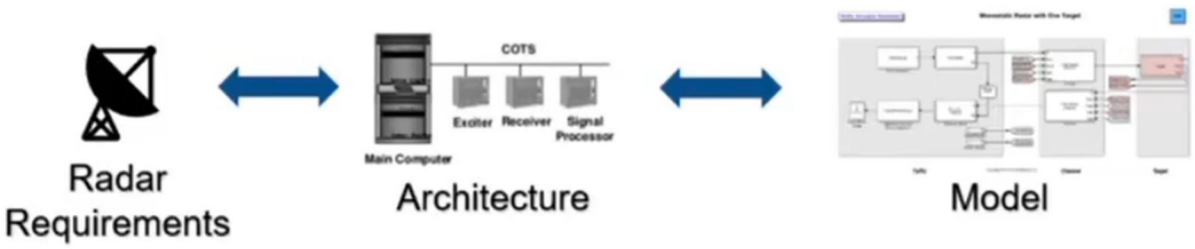
In addition to that, when your SW and HW designers are manually writing code while using document-based requirements, the only way you can verify their code is through a slow, exhaustive verification process.
So here, again, there is no direct connection between system requirements and the desired outcome (in this case - the production code).

Now, At Last, the Problem is Solved

But now, let's talk about a new workflow that's creating an exciting breakthrough for system engineers.

Imagine if, instead of handing down requirement docs to the design teams, you could sketch out a simple block scheme that shows your architecture, and then connect the requirements in the system spec to the relevant part in your block scheme.

Now, imagine the design teams - HW, SW, Algo. . . even Analog and RF - could "pour" their designs into your block scheme, literally transforming it into an actual system model?



What that would allow you to do is this:
First, you could simulate that system model, and get an instant view of your system behavior.

This would allow you to quickly test multiple "what if" scenarios -

without risk, delay, or any reliance on costly hardware - in an early stage in the design process.

Second, since requirements are now connected directly to the system model (through your block scheme), you can *see how the actual system complies with those requirements*, as well as assess the impact of any requirement change on the system - at any given time!

Does this sound too good to be true?

That is what several MathWorks clients believed too until they saw what I'm about to show you now.

MathWorks' New Solution and Methodology

This new workflow I've just described, which is a new part of what MathWorks calls its Model-Based Design (MBD) methodology, lets you write, analyze, and manage system requirements within the system design model.

Furthermore, requirements can also be imported and synchronized from external requirements management tools, such as IBM Doors, Microsoft TFS, or even a simple Word document or Excel spreadsheet.

Then, you can link those imported requirements to the system design, tests, and even final production code.

Once a requirement is linked to the model, designers will get an automatic notification when that requirement has been changed.

This way, they can immediately identify which part of the design or test is directly affected by this change, and quickly assess design performance under this new requirement.
But here's something else this allows you to do.

Explore More Ideas and Scenarios

You can also define and analyze system architectures and scenarios.
This means you now have a way to explore and test any idea, at any time in the design phase, within a matter of minutes.

In fact, you can:

- Quickly evaluate multiple design ideas
- **Explore tradeoffs**
- See how each design change affects your system, and even. . .
- **Investigate design problems and answer integration questions - long before you build expensive hardware or prototypes!**

System requirements. . . HW components. . . SW IP. . . and even test scenarios - are all captured in your model.
Plus, you could (if you wanted to) take that one step further and solve another big design problem.

Trace Requirements to the Final Production Code

Instead of writing thousands of lines of code by hand, what if your design team could also generate production code *directly* from the same system model?

You see, with MathWorks production code generation tools, your design teams can convert the system model into actual code that can be implemented on your production embedded system.

This means that:

1. The automatically generated code can be used for rapid prototyping or production, and . . .
2. Since the production code was generated directly from the design model, it's still linked to system requirements, which means you can trace any requirement all the way down to the lines of code that implement it.

This creates a fast and inexpensive way for you to test algorithms on real hardware, in real-time, and perform design iterations in minutes rather than weeks!

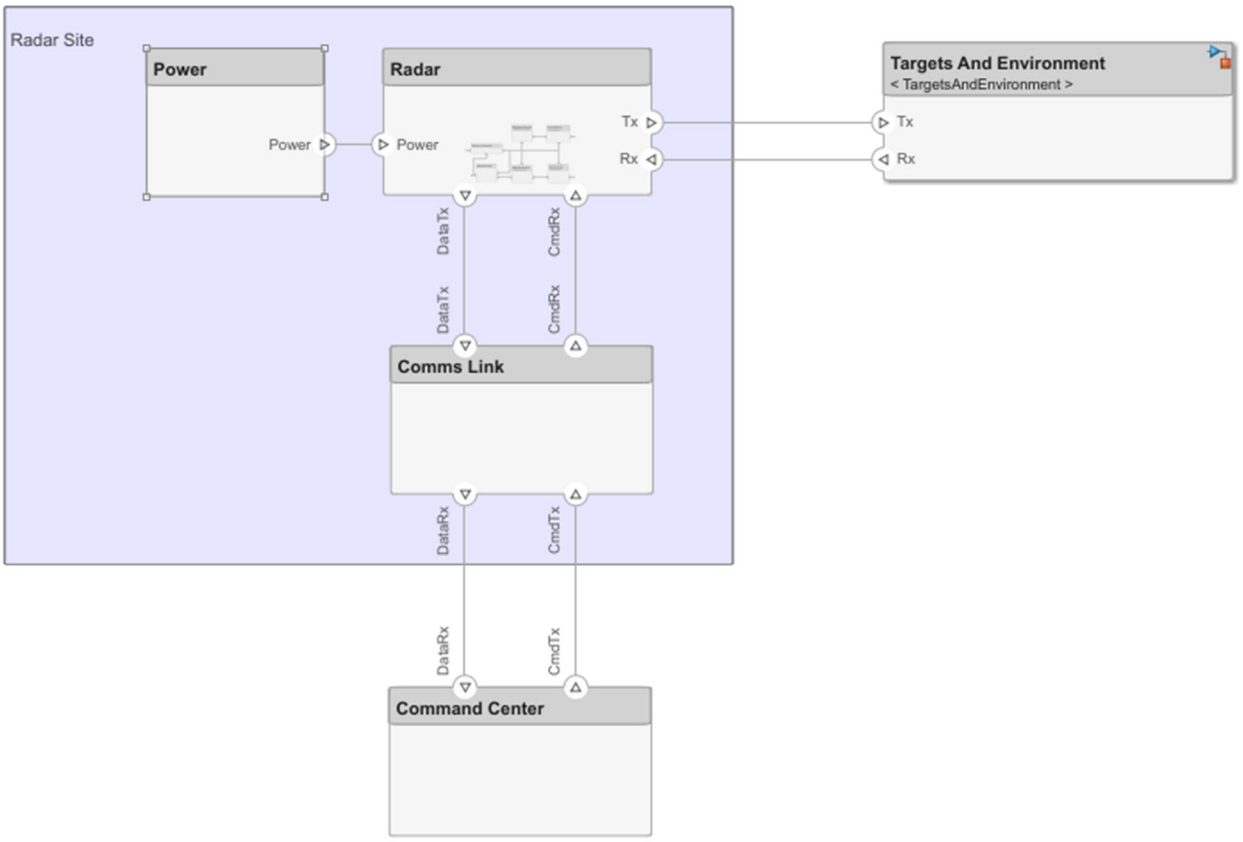
So, let's see how this actually looks like in a real design.

Here's What It Looks Like

For example, let's say you're designing a RADAR system.
(The process is similar for any other type of wireless comms system.)

You begin by building a simple block scheme from system requirements.

So for a radar system, it would look something like this:

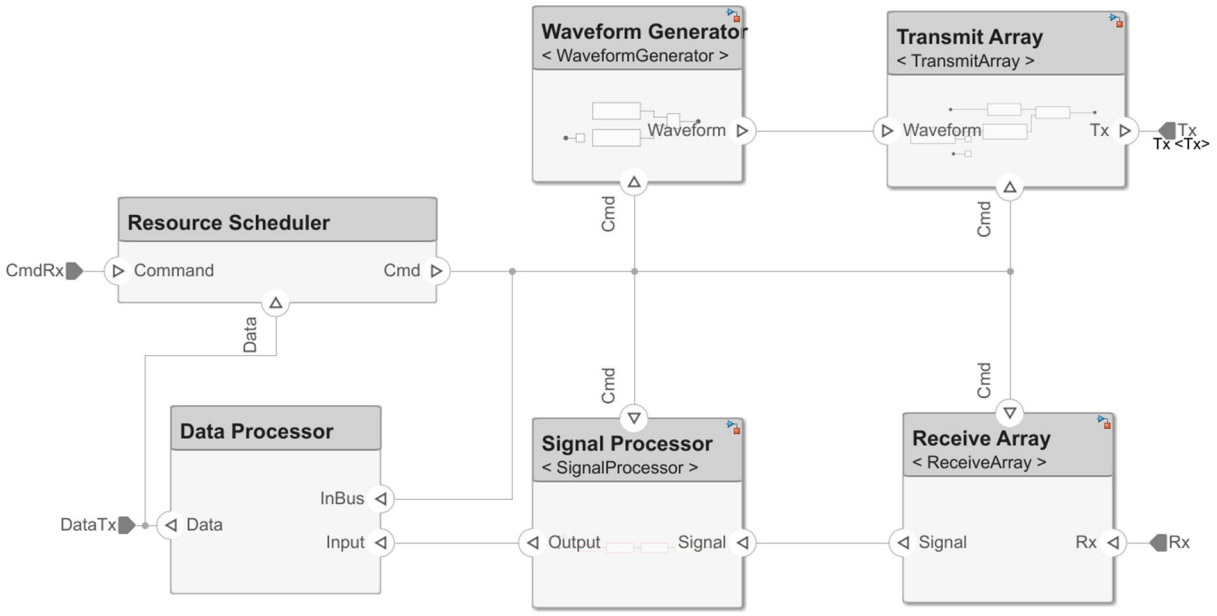


[Caption] ***Step 1 in this new methodology: A simple block scheme that is connected to system requirements and a system model***

As you see, you have a radar transceiver, power, and control circuitry, operating environment (including targets), and link to the command center.
Now let's add some more detail.

Inside each block in the rough sketch above is a high-level, low-fidelity model.

For example, here's what would be inside the radar transceiver block:

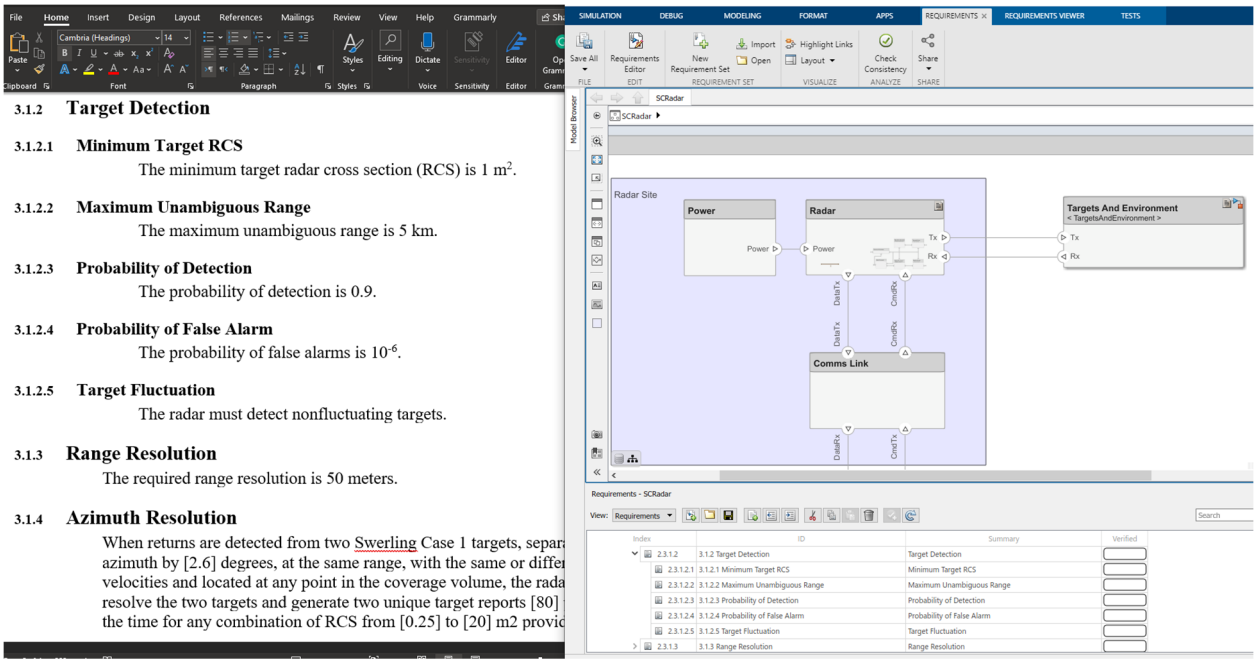


[Caption] ***Step 2: Within each block are additional, lower hierarchy blocks that connect to specific design models***

Each of these blocks is connected to a design model the design team has built, which can contain any (or all) of the following:
MATLAB code, C code, Simulink blocks, Stateflow diagrams, analog and RF designs, and even legacy code.
Now here is where this starts getting really exciting.

Whether you write your system requirements in a Word document, Excel spreadsheet, or an external tool, such as IBM Doors or Microsoft TFS, you can import that to any part of the above system model.

So, if you used a simple Word document for this radar system, the section about target detection would look something like this:



3.1.2 **Target Detection**

3.1.2.1 **Minimum Target RCS**
The minimum target radar cross section (RCS) is 1 m^2.

3.1.2.2 **Maximum Unambiguous Range**
The maximum unambiguous range is 5 km.

3.1.2.3 **Probability of Detection**
The probability of detection is 0.9.

3.1.2.4 **Probability of False Alarm**
The probability of false alarms is 10^{-6}.

3.1.2.5 **Target Fluctuation**
The radar must detect nonfluctuating targets.

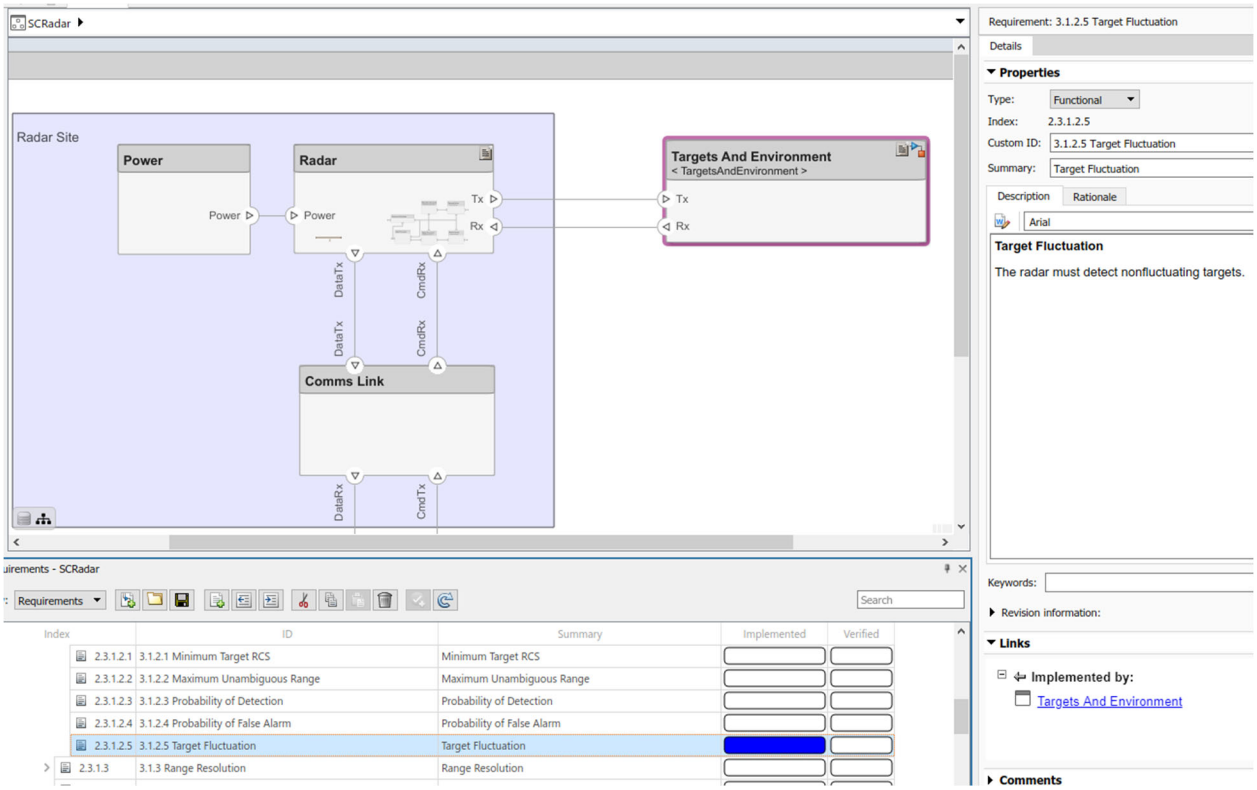
3.1.3 **Range Resolution**
The required range resolution is 50 meters.

3.1.4 **Azimuth Resolution**
When returns are detected from two Swerling Case 1 targets, separ
azimuth by [2.6] degrees, at the same range, with the same or differ
velocities and located at any point in the coverage volume, the rada
resolve the two targets and generate two unique target reports [80]
the time for any combination of RCS from [0.25] to [20] m2 provid

[Caption] ***Step 3: System requirements in a simple Word document format is imported into the system design model***

Now you can link each of these requirements to a block in the system model.

So, for example, requirement 3.1.2.5 concerns target fluctuations.
So, obviously, it should be linked to the Targets & Environment block, as shown below:

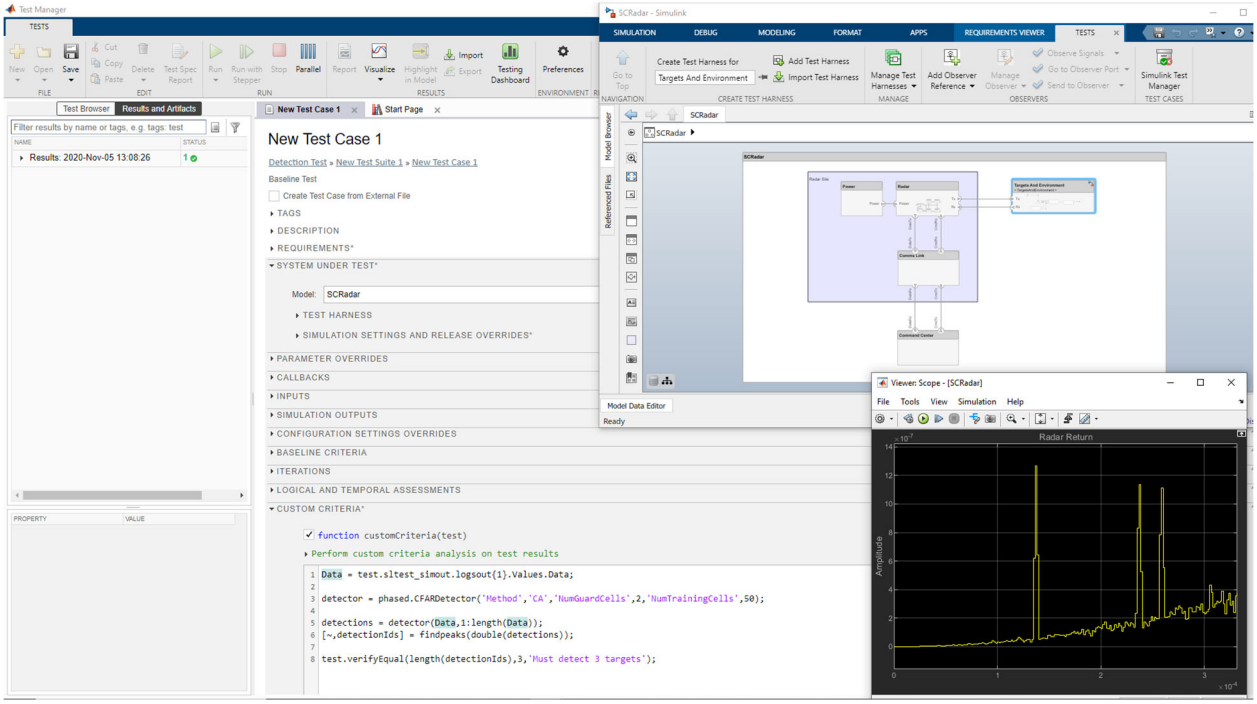


[Caption] ***Step 4: Target Fluctuation requirement is linked to the (highlighted) Targets and Environment block***

Now you can write a test in MATLAB that will define how that requirement will be tested for compliance in the model.

That test can run whenever you want, so you can use it to test for compliance of various architectures and scenarios in a matter of minutes.

Here's an example of a test that finds the number of identified targets in the radar model above:



[Caption] ***Step 5: A test scenario that finds the number of targets identified by the radar runs successfully in MATLAB***

Now, in case there is a change in the requirements doc, you'll get a notification.

So let's see what happens when we change the Maximum Ambiguous Range requirement from 5km to 8km, and the Target Fluctuation requirement from Non-fluctuating to a Swerling Case 2 target:

As you can see in the picture above, the requirement change is highlighted in red.
We update the model's max. Range parameter to 8,000, and the target type to a Swerling type 2.
Now, to see the impact of that change on system performance, we can simply run the test again.

As the project moves forward, the design team members update this model, continuously testing and verifying the system-level behavior according to your requirements, using testing scenarios and criteria you have defined.

Finally, when the model has been thoroughly tested, the HW & SW engineers can generate code directly from it, and download the generated code to the production hardware for quick prototyping or HW-in-the-loop tests.

Here's How You Can Try It Yourself

So, what do you think? Would you like to "test-drive" this demo yourself? I'd love to give you that opportunity, as well as answer any questions you may have regarding this whole workflow.

Simply shoot me an email at shlomis@systematics.co.il and I'll be sure to reply promptly.

I hope to hear from you soon.

Sincerely,

Shlomi Shraga
Applications Engineer, Systematics

P.S. We are doing an online event soon, where I'll be showcasing this workflow, along with other new MathWorks tools for your industry.
Would you (or any of your peers) like to join me?

If so, <link> go here to sign up now for the event <link>.

P.P.S. Feel free to share this article with your peers.