Maximize Effectiveness of Model-based Testing

The journey to efficient software testing starts with a mindset and process shift – embracing a model-based combinatorial methodology. Traditional test design approaches often lead to the following problems:

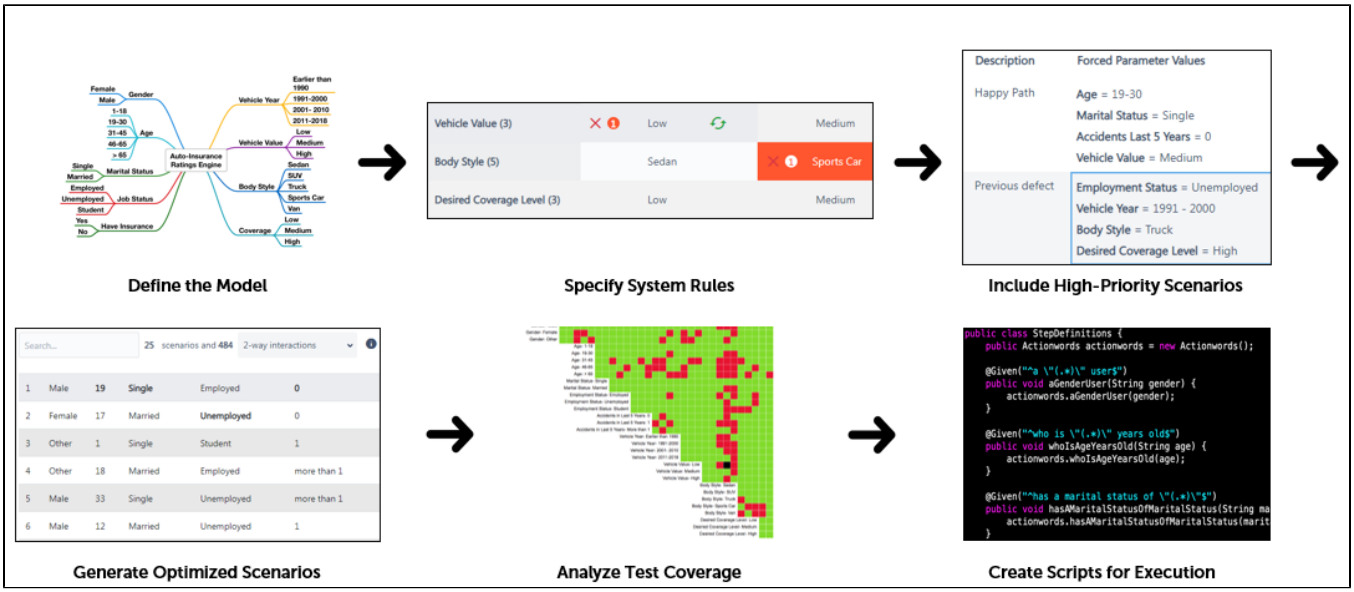
- Direct duplicate tests due to inconsistent formatting or spelling errors
- “Hidden”/Contextual duplicates (meaningful typos; same instructions written by different people with varied styles);
- Tests specifying some values, leaving others as default (when several scenarios can be tested in the same execution run).

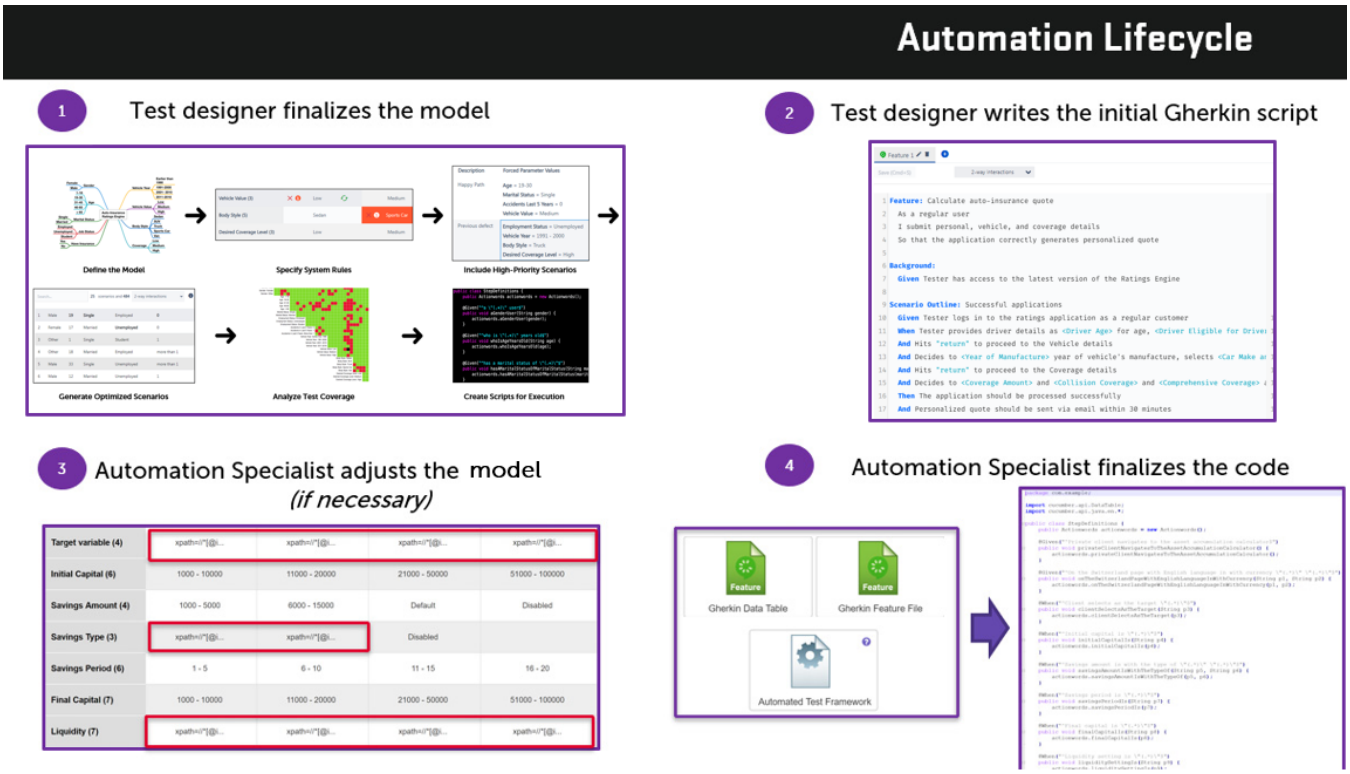
Adopting a more scientific strategy, as confirmed in multiple Test Case Designer case studies, helps eliminate large amounts of inefficiency – reducing the number of tests and making the test suites more easily maintainable.

Approach Summary

Such combinatorial, model-based testing can be applied beyond BAU regression optimization to scenarios like new feature releases or applications undergoing a redesign.

The “best practice” process to follow is shown in these flow diagrams:





It can be aggregated into the “cheat sheet” with four key stages.

Stage 1: Begin test design efforts focusing on potential variation in the system under test

Instead of focusing **only** on details specified in requirements documentation, it is crucial to take a step back and evaluate what matters for the system as a whole. It is necessary to analyze which steps a user could take in an application and which choices they could have at each step.

Then it is important to consider external elements that could affect user behavior (dependencies with other applications, environments, etc.).

Finally, testers should organize that information appropriately, like parameter/value tables, to review this information with stakeholders. Test Case Designer mind maps (first image on the top diagram above) can also be useful to facilitate such discussions.

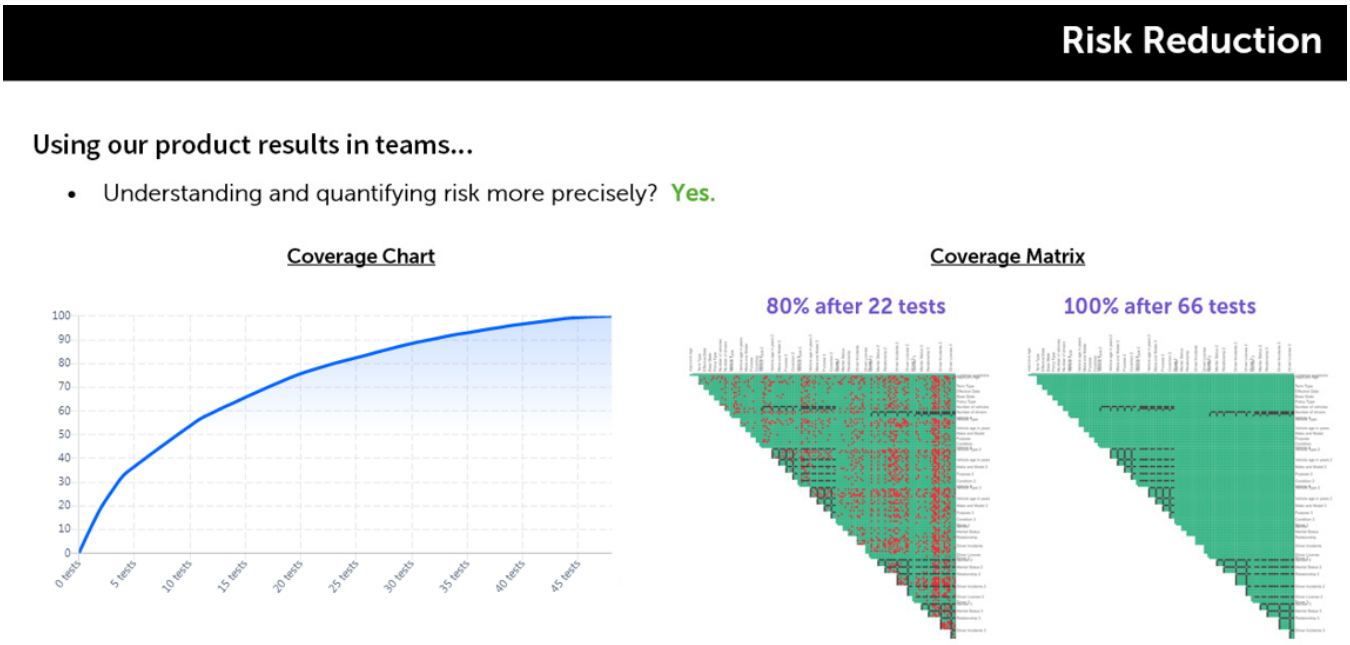
Once all the key input parts of the model have been agreed upon, the team should analyze the constraints (i.e., how inputs can and can’t interact with each other) and requirements (both formal and informal).

Stage 2: Create systematically-varied, data-driven test scenarios

Next, test designers should generate efficient and thorough test scenarios based on the inputs defined in the model.

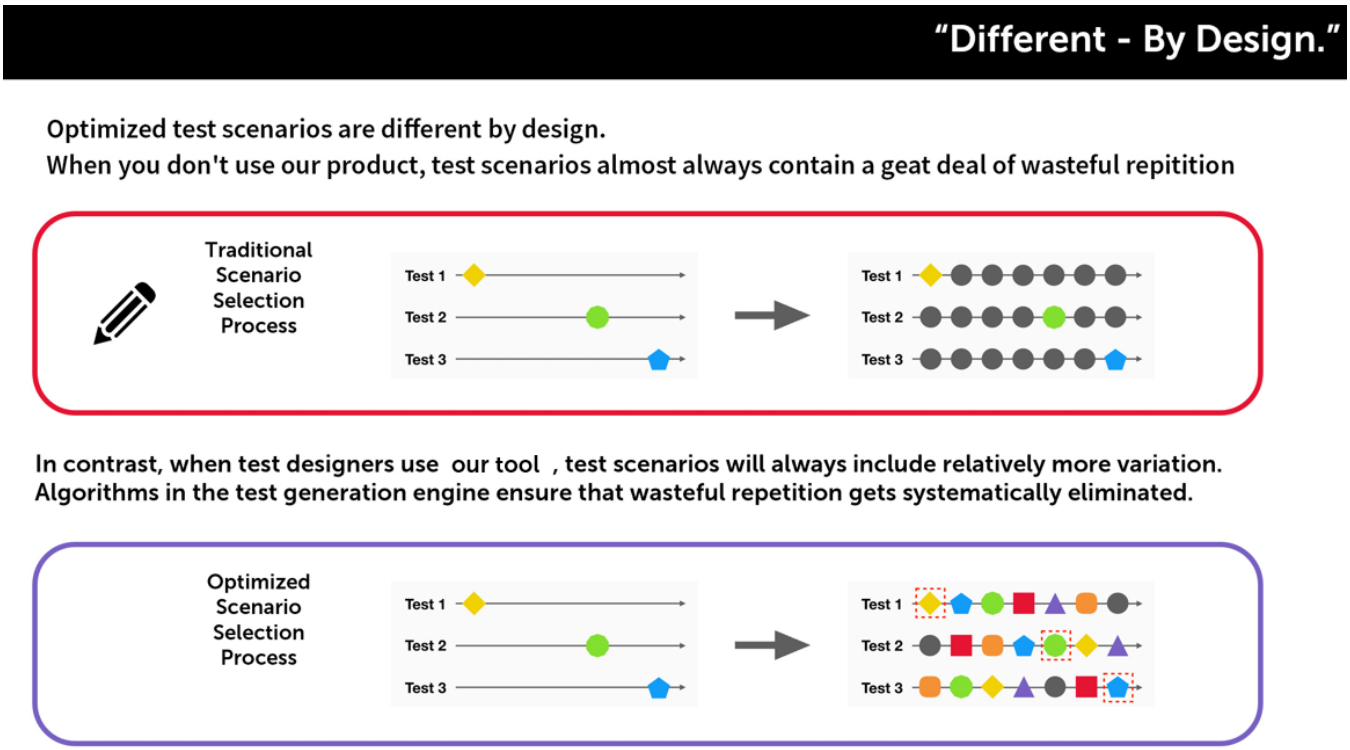
Using Combinatorial Test Design methodologies at this step is beneficial as they apply selection algorithms to identify the scenario components that guarantee maximum variation in the minimum necessary number of tests.

This process can be difficult and time-consuming to accomplish manually, but a proprietary combinatorial algorithm helps TCD simplify and accelerate this part of the process.



Further, it's crucial to understand the level of interaction coverage of the test suite as, on average, 84% of defects in production are caused by 1 or 2 system elements acting together in a certain way.

Evaluating the test suite at the model level leads to risk reduction through a clear understanding of what is covered in each set of tests (for example, by leveraging the coverage graph & matrix in Test Case Designer).



Stage 3: Create consistent and accurate documentation

Increasing the consistency and reducing the ambiguity of both script steps and expected results improves execution time and effort and make automation more straightforward. When considering the expected results, it is important to identify which combination of inputs triggers each desired outcome precisely.

TCD allows you to create a single script for the model that is automatically applied to each test scenario. This allows for the best practice of creating data-driven test scripts and ensures the script creation effort does not increase linearly with each test scenario. The tool allows users to incorporate the appropriate logic for expected result generation.

Stage 4: Simplify test case maintenance over time

One of the key advantages of data-driven testing is the ability to make a single update to the model's inputs and have that update automatically applied to all test scenarios. This saves significant time and effort in maintaining a test suite over time and helps ensure consistency and accuracy of all test cases. TCD also allows you to export the updated artifacts in various formats with test case management and test automation tools.