# Test Creation Best Practices

Now that we have established the sharing mechanism, collaboration workflows, and test execution methods, we can explore some tips on how to best use SOAtest to efficiently create effective and maintainable tests.

## Use SOAtest Test Creation Wizards

SOAtest can create a new Web service test project from a WSDL, OpenAPI/Swagger or RAML definition, a UDDI registry, a WADL, a WSIL file, monitoring ESBs, or a traffic log file. Web UI tests are created by recording using a browser. These features are useful in jumpstarting work to create the initial building blocks for your use case scenario tests.

## Use Copy, Cut, and Paste

Leverage existing tests and tests suites to create new ones instead of creating each one from scratch. Available .tst files, test suites, tests, and chained tools can be copied, cut, and pasted with right-click commands as well as keyboard commands (CTRL+C, CTRL+X, and CTRL+V, respectively).

## Split Tests into Many Test Suite (.tst) Files

Do not lump all your tests into very few .tst files. Having too many tests organized into too few large .tst files could make the product run slower because more tests are open and in memory. Moreover, the benefits of using revision controls in a repository are diminished and remote users over a network could experience slower performance (since they would end up transferring large .tst file sizes instead of smaller ones).

SOAtest helps you maintain files into projects and directories to facilitate the management of large numbers of test cases. This results in a more balanced tree of resources that helps you get more out of the organized hierarchy of projects, folders, test suite files, and test suites.

## Split Test Suite (.tst) Files According to a Consistent Pattern

Depending on the scale of your web services, the splitting pattern may be “Each test file is associated with a WSDL” or “Each test file is associated with an operation in the WSDL.” For Web applications, tests for certain areas of the Web site may be divided into .tst files based on the area name.

The ideal splitting strategy will balance the number of files and the included tests in a way that makes it easy to find the test you are looking for and easy to understand the reports (since the reports reflect your test organization).

## Separate Service Definition Tests, Service Unit Tests, and Functional Tests into Multiple Test Suites

We recommend having three major test suites in each test suite file:

- WSDL/OpenAPI/Swagger/RAML/WADL tests
- Unit tests
- Functional tests

If you organize tests in this way, it’s easy to assess whether all these validation aspects are covered.

## Organize Service Unit Tests into Positive and Negative Test Suites

Choosing the “Organize as positive and negative unit tests” layout in the wizards that generate tests from a WSDL is an ideal way to create a test suite with the proper layout for sufficient and well-organized unit tests. Creating both types of tests is important for achieving sufficient test coverage, and this layout helps you quickly assess whether additional positive or negative tests may be needed.

## Name Tests and Test Suites Properly

Be sure that each test and test suite has a name that is meaningful to you and your team.

By default, SOAtest uses the WSDL binding name for Web service test suite names, the operation name for SOAP Client test names, and browser actions for Browser Playback test names. However, these names can be changed to better reflect what is being tested. To modify the default names (e.g., “Test Suite,” “Test 1,” “Test 2,” etc.), enter a new value in the test or test suite’s configuration panel.

Using a meaningful name not only helps with test maintenance, but it also improves report appearance and readability. Another helpful strategy is to add further details and comments about each test or test suite in its Notes panel.

## Add Regression Controls, Assertions, or Other Success Indicators to All Tests

It is critical that success criteria are configured on the tests. Without such criteria, SOAtest does not know what constitutes a test failure… and a regression error could go uncaught.

To accurately identify test failures, tests need regression controls or success indicators beyond getting a SOAP Fault response. XML Assertor, JSON Assertor, Diff, Extension tool (formerly known as “Method tool”), Search tool, and so on can be chained and combined to add the proper level and type of validation on the XML messages or HTML content.

## Use Existing Tools for Repeated Tool Configurations or Tests

SOAtest allows you to add global tools that can be accessed at the test suite file level. If you think you will be using a validation tool for multiple tests (such as a Diff tool with a reusable regression control), add the tool to the test suite, then reference it as an existing tool. This way, you don’t need to redefine it time and time again for multiple tests.

## Use Shared Properties for Repeated Configurations

SOAtest allows you to add global properties so that settings such as ignored Diff tool XPaths, JMS connection settings, or database connection settings are configured in a single location at a test suite file level, then referenced and reused. This makes it easier to maintain and modify the tests in the future.

## Use Test Suite References

You can create a reference from one test suite to another test suite file by right-clicking on a test suite and choosing Add Test> Test Suite Reference. This allows you to define a set of tests that should be executed in all project files (such as common log on/authorization tests) and have them referenced in multiple files for reuse.

## Use Environments

Parasoft SOAtest allows you to define sets of variables and values, then reference those variables in almost any user-editable field with the tests. This is typically useful for configuring hostnames and URLs so the same tests can be executed against different environments. Environment configurations can be created and shared in a single test suite file or across multiple test suite files by referencing environment configuration files.