TAME Turns Better Specs into More Productive Tests

Many individual test cases are needed to properly validate a large software system. Yet writing these tests has always been one of the most tedious and error-prone software lifecycle tasks.

"Even when you automate test execution with tools like Selenium or TestComplete, you're still left with the job of creating those test cases," said Marc Balcer, CTO of Model Compilers and chief architect of TAME. "Automation, in fact, has created a 'hungry beast' that needs to be fed with lots of test cases."

TAME—Test Authoring Made Easy—produces comprehensive test cases and scripts from simple yet detailed functional specifications. Supporting behavior-driven and keyword-driven approaches, it can create tests for single functions (a single UI page or function in code), sequences of functions (use cases and user stories), complete business processes, and whole systems. Using TAME, a tester creates specifications in simple spreadsheets, defining how individual situations can lead to different outcomes. The TAME processor then weaves together these various bits and pieces into complete test cases.

TAME is available as a cloud offering or as a Windows application. A limited TAME Personal edition is free for evaluation and education purposes; more capable Professional and Enterprise versions offer capabilities for test tracking, collaboration, and enterprise reporting.

"It's simple to use—if you know what your software is supposed to do and can fill out spreadsheets, you can use TAME to produce test cases," Balcer said.

Take the boring out of building tests

Test case development is still a largely *ad hoc* manual process that relies upon testers who can interpret wordy and unstructured specification documents.

"When I conduct training courses and help customers to implement test automation, I see that learning the tools and APIs is easy. The hard part is in knowing what to test." He noted that while most testers can easily plan the main success scenarios, the many alternate and exception scenarios are much more troublesome. "Part of it was in having a way to scope out all of the alternates. But even when you can identify lots of alternates, the process of turning those into executable tests can be very repetitive, error-prone, and just plain boring."

> "I wanted to make the process of using TAME so obvious that testers would say, 'why haven't we been doing this all along?'"
> —Marc Balcer

Balcer developed the principal analysis techniques as a graduate student. "We called it the 'Category-Partition Method' because, for each function to test, you'd categorize different inputs and environment conditions, and then partition each category into different choices of values that would cause different results from the function." A tool then formed combinations of those choices in order to create many test cases.

That process of forming combinations and then matching those to expected results is at the heart of building comprehensive test cases.

Model complex behavior simply

Comprehensive testing requires identifying and exercising different paths through the software. Even the simplest functions can have many different such paths, and bugs slip through when some of those paths aren't tested.

"Think about a login screen. What does it take for a user to log into an application? The user ID and password have to be specified—not blank. They have to be in the right form. The user ID has to identify a real user and the password has to be right for the user. Anything else and you'll get an error message."

The complexity, of course, is that there can be different messages for different errors, and sometimes a user might get more than one error message. "But, of course, if the user hits the 'cancel' button, it doesn't matter what was in the user ID and password fields; the user will just go back to the homepage."

Using TAME, the tester identifies two different inputs, the user ID and the password fields. He identifies two different actions, clicking OK or clicking Cancel. Each of the possible outcomes is a different result, such as "the user is logged in" or "the user gets a 'bad log in' message." Environment conditions, such as whether there is a real user with the given user ID, round out the set of categories.

Those categories are then partitioned into different choices, such as whether the user ID is empty, of bad syntax, or of good syntax. Finally the tester identifies which sets of choices lead to which results. "To log on correctly, the user ID and password have to be of good syntax, there has to be a user in good standing with that user ID, and the password has to be correct for the user." Specifying that in TAME is as easy as putting an "x" in the cells at the intersection of the choices' rows and the results' columns.

"I call it 'X marks the spot,'" Balcer said. "I wanted to make the process of using TAME so obvious that testers would say, 'why haven't we been doing this all along?' The notation has to be something that doesn't require complex training to be able to read. When students come into our training course, I have them read a TAME spec and tell me what it says even before we start the first lesson. They're rather amazed at the TAME's simplicity but also impressed by the sophistication of what they can express."

Get the most from test automation

TAME makes combinations of inputs, applies different preconditions to those combinations, and matches them to expected results to form test cases. TAME creates test cases systematically and includes features for detecting and eliminating redundant tests.

Since TAME is a specification-based "black-box" testing method, test development can be done in parallel with or even prior to coding. TAME tests are not biased by knowledge of the implementation and are thus more likely to find errors resulting from misinterpretation of the original software requirements.

"Back when we first published the Category-Partition Method, automated test execution tools really didn't exist." Test execution was often limited by the availability of people to do the testing, and so testing exhaustive permutations of inputs and environment conditions was out of the question. "Now that test execution is automated, there's far less of a barrier to running lots of tests—today the difficulty is in creating those tests."

TAME gets your team writing better specifications because it organizes key knowledge into a well-structured, easily-understood form. The result is a set of test cases—exhaustive enough to give your software a good workout—but intelligent enough to not waste time and effort.

Learn more at www.tametest.com.