More tests.
Not more work.

tame Test Authoring Made Easy

Welcome!

You can be waterfall or agile. You can be big or small. You can be legacy or cutting-edge.

No matter your approach, if you build systems, you know you have to test. And you need a lot of test cases to do it right.

But where do you get all of those test cases? Someone still has to write 'em—and that can be a boring, repetitive slog.

TAME is Test Authoring Made Easy.

Fill out a simple table and TAME produces dozens of useful test cases.

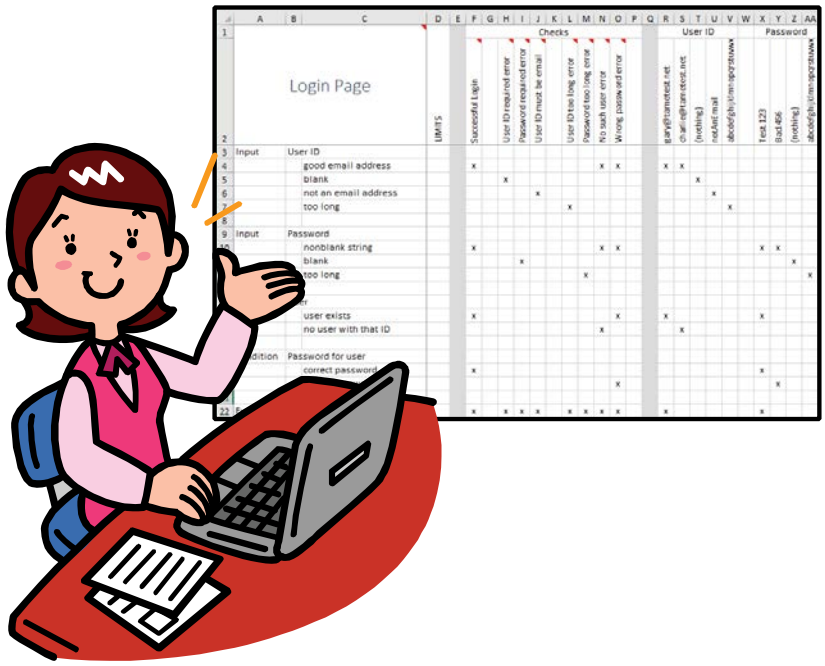
Enhance your software then update your test base in minutes.

Easily maintain a consistent, reliable suite of tests.

TAME the software testing beast!

Marc J. Balcer
Chief Architect, Model Compilers

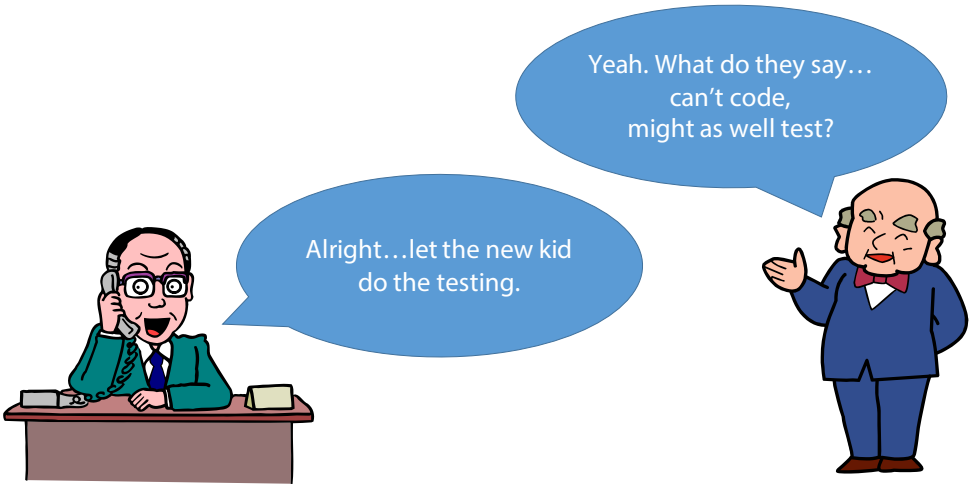
For more information contact sales@tametest.com
https://www.tametest.com
+1 (415) 931-3132



Contents

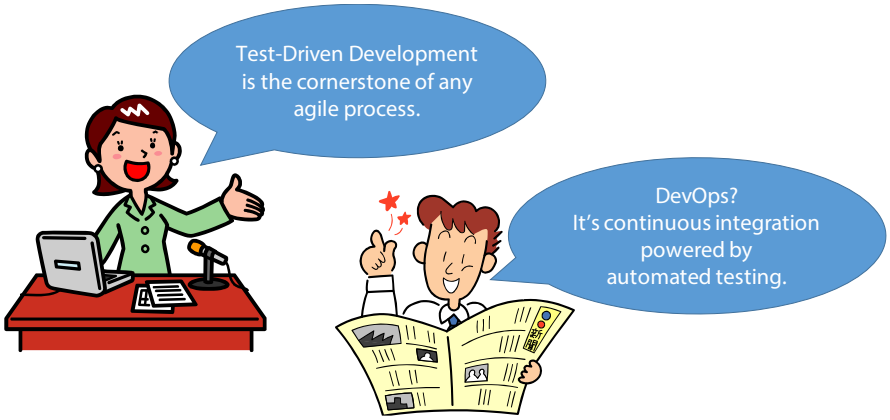
Testing Gets Respect

It wasn't too long ago that testing was a sort of backwater in software development. Perhaps you can still remember when the general attitude was, "let the new kid do the testing." Or testing was what you did when you weren't seen as good enough to do real development.



Testing was somewhere between a necessary evil and "anyone can do it."

Then a funny thing happened. The short cycle times and focus on working code introduced by agile methods put a new emphasis on testing.



New techniques like Extreme Programming required tests for every function to be delivered. Today, test-driven development is the cornerstone of agile processes. DevOps stresses the importance of continuous integration powered by automated testing.

But while automated test execution is essential to keep the pace for agile development, you're still left with the time-consuming problem of designing many test cases.

The Hungry Beast

"Agile development
and automated testing
have created a 'hungry beast'
that needs to be fed
with lots of test cases."

It's been lost to history who first referred to agile teams as "hungry beasts," but if you've ever been a product owner or tester on an agile team you know right away what that means.

It's a huge challenge to build all the tests you need in order to keep up with all of the new and changing features. Maybe it's the short two-week iteration cycles, maybe it's the way features are divided into small units called user stories, maybe it's the insistence on testing, maybe it's all three.

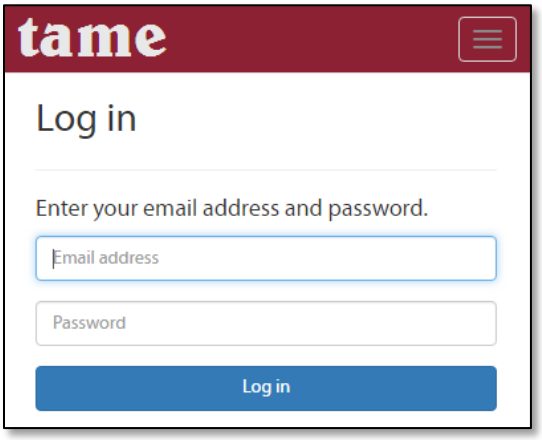
Now in the interest of full disclosure, the hungry beast doesn't just stalk agile projects. Have you ever faced this situation?

- The business wants a new feature or change.
- The developers think they know what "it" is—and so they build it quickly.
- QA is backlogged and won't get to testing "it" for a few weeks—or longer.
- When the business gets the new feature it's not what they want.

The solution? You need to have the tests to get agreement on the scope of work. You need lots of tests—and you need them quickly.

Don't Be a Serial Tester

Consider an application login page, such as the one on the TAME website.



The typical way to write the tests for a login page might be to start with the main success scenario or "happy path:"

1. Start on the login page
2. Enter a good user ID (gary@tametest.net)
3. Enter a good password (Test.123)
4. Click the blue Log in button
5. Verify that the user has been granted access to the application.

Another scenario would have the user enter a bad password. Then the user will get an error message and not be granted access to the application.

Still another scenario would leave the user ID blank. This would result in a different error message.

How many more of these scenarios need to be created? Whether it's a small or a large number, writing them one at a time is going to be painful.

Writing tests one by one—serial testing—can take up a lot of time. It's really boring and repetitive and it often results in incomplete and unmaintainable suites of tests. Under these circumstances, it's not uncommon for comprehensive testing to fall seriously behind.

So here comes the cliché: let's tame that testing beast.

We want to build tests quickly and efficiently, but also ensure that our tests cover the required functionality.

What about TDD, BDD, and Cucumber?

Test-Driven Development (TDD) and Behavior-Driven Development (BDD) invert the typical development and test paradigm: you develop the tests before software design and coding.

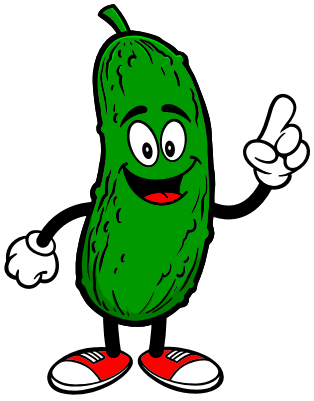
The advantage of developing tests first is in the specificity: each test is an example of necessary functionality. The people with the requirements can say, "yes, that's what I need" or "no, that's not it." The developers have a clear sense of scope: they build what's needed, they are less likely to miss something important, and they're strongly discouraged from gold-plating (building what's not needed).

Cucumber is a popular BDD framework. Its language, Gherkin, provides many teams with a common syntax and structure for writing tests:

```
Scenario: Successful Login
    Given I'm on the application's login page
     When I enter 'gary@tametest.net' into the User ID
      And I enter 'Test.123' into the Password
      And I click the blue Log in button
     Then the login is successful. I'm taken to the My Projects page
```

This Given-When-Then language lets teams define scenarios in natural language. Tools in the Cucumber family can generate code to enable automated unit and UI testing.

But a popular common syntax does not address the problem of how to devise those many scenarios. Testers still have to create them one at a time.



Fortunately, as you'll see on page 17, TAME can create all the Gherkin BDD scenarios from a single test workbook.

More Tests. Not More Work.

TAME is based on the idea of combinatorial testing—an approach in which test cases are formed from different combinations of inputs and environment conditions.

Of course, you can't reasonably test every different user ID and password value. You don't need to. You just need to identify different "characteristic values"—simply known as "choices"—different kinds of values that produce different results.

What are some different choices for the user ID?

Input	User ID
	good email address
	blank
	not an email address
	too long
Input	Password
	nonblank string
	blank
	long blank string
	too long

- It could be a good email address.
- It could be empty.
- It could be bad syntax: something that doesn't look like an email address.
- It could be too long. You might never try to enter a 500-character user ID, but it's just something like that that could break the application.

There are a similar set of choices for the password: Check for a good value, an empty value, a value that's all whitespace characters, and a value that's too long.

The simplest way to form test cases would be to blindly form combinations by taking one choice from each of the inputs. Two inputs with four choices for each produces 16 test cases. This simple combination approach produces many test cases, but many of these are redundant and not terribly useful.

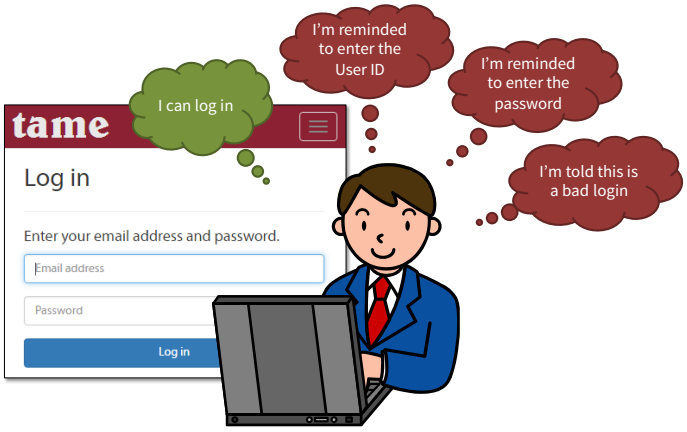
Combinatorial Explosion

More inputs and more choices only make the redundancy problem worse.

Seven inputs and five choices for each would require 7^5 (16,807) combinations. While seemingly comprehensive, many of these tests are redundant, testing the same situation or error over and over again.

TAME has figured out how to form combinations without all the redundancy.

How is that? In TAME, define the set of expected results and then identify the choices that lead to those results.



A successful login is one result. Other results include error messages shown when the user ID or the password are missing. Another result is a general message that says that you can't login with the given user ID and password.

List the results across the top of the sheet in the "Checks" section.

Login Page		LIMITS	Checks								
			Successful Login		User ID required error	Password required error	User ID must be email		User ID too long error	Password too long error	Blank password error
Input	User ID										
	good email address		x								
	blank				x						
	not an email address						x				
	too long								x		
Input	Password										
	nonblank string		x								
	blank					x					
	long blank string										x
	too long									x	
Event	Log in		x		x	x	x		x	x	x

Finally, identify what causes each of those results.

Do this by marking the cells at the intersection of the input choices and the results.

As we like to say,
"X marks the spot."

Notice the Event row. This identifies the behavior that actually triggers the specified result, such as a button click.

We can read the Successful Login column like this:

```
Scenario: Successful Login
    Given I'm on the application's login page
     When I enter a good email address into the User ID
      And I enter a nonblank string into the Password
      And I click the blue Log in button
     Then the login is successful.
```

Well,
not exactly…
see the next page

In some cases, a single choice is all that's required in order to yield a particular outcome. For example, if the user ID is missing when the login button is clicked, that error message will be produced.

Environment Conditions

Sometimes a function's results depend upon more than just the inputs. Just having a good syntax user ID and password won't guarantee that the login will be successful.

Login Page		LIMITS		Checks											
				Successful Login		User ID required error	Password required error	User ID must be email		User ID too long error	Password too long error	Blank password error		No such user error	Wrong password error
Input	User ID														
	good email address			x										x	x
	blank					x									
	not an email address							x							
	too long									x					
Input	Password														
	nonblank string			x										x	x
	blank						x								
	long blank string											x			
	too long										x				
Condition	User														
	user exists			x											x
	no user with that ID													x	
Condition	Password for user														
	correct password			x											
	wrong password														x
Event	Log in			x		x	x	x		x	x	x		x	x

The solution is to add environment conditions. Sometimes called "preconditions," these are properties of the system that are expected to be true before the test runs.

For example, a good syntax email address and good syntax password will result in a "no such user" error if there is no user with the given user ID.

Another condition produces a "wrong password" error when the user exists but the password is wrong.

Only when the test has a good email address that identifies a real user and a good syntax password that is correct for the user will the login be successful.

Lots of tests!

Upload this spreadsheet to TAME and get nine different tests—the happy path plus eight error conditions.

tame

Test Case	Name
1	Successful Login
2	User ID required error
3	Password required error
4	User ID must be email
5	User ID too long error
6	Password too long error
7	Blank password error
8	No such user error
9	Wrong password error

What do these tests look like? Here's one of the tests that TAME produces:

Case 4: User ID must be email

Inputs

User ID : not an email address

Procedure

Start:
Login Page
Input:
Enter 'not an email address' into User ID
Do this:
Log in
Check:
User ID must be email

The test case is written in terms of a starting point, the behavior to be performed ("input" and "do this"), and a final check.

The test case title is taken from the expected results

There's nothing magic to this: everything that is in the test case can be found on the worksheet.

While the test cases produced by TAME appear fairly detailed, they don't actually define what's meant by a good syntax user ID or a password that's too long. The language has a stilted, "template" feel to it.

Fortunately, with a little more work you can get TAME to produce much more precise tests that say exactly what to do.

Defining Exact Values

TAME provides a simple but sophisticated method for defining the actual values to be used in the test cases. List the different values across the top, and mark choices to indicate which value is used for which combination of choices.

Login Page		LIMITS	Checks											User ID						Password					
			Successful Login		User ID required error	Password required error	User ID must be email		User ID too long error	Password too long error	Blank password error		No such user error	Wrong password error	gary@tametest.net	charlie@tametest.net	(nothing)	notAnEmail	abcdefghijklmnopqrstuvwx		Test.123	Bad.456	(nothing)	"	abcdefghijklmnopqrstuvwx
Input	User ID																								
	good email address		x										x	x	x	x									
	blank				x												x								
	not an email address						x											x							
	too long								x										x						
Input	Password																								
	nonblank string		x										x	x							x	x			
	blank					x																	x		
	long blank string										x													x	
	too long									x															x

Values are defined separately from the inputs simply because the actual values often depend upon *both* the input choice and environment conditions. Both gary@tametest.net and charlie@tametest.net are good email addresses, but only gary@tametest.net names a real user.

Marking a value on an Event row selects the default value to be used on that event when the test case treats the input as "don't care."

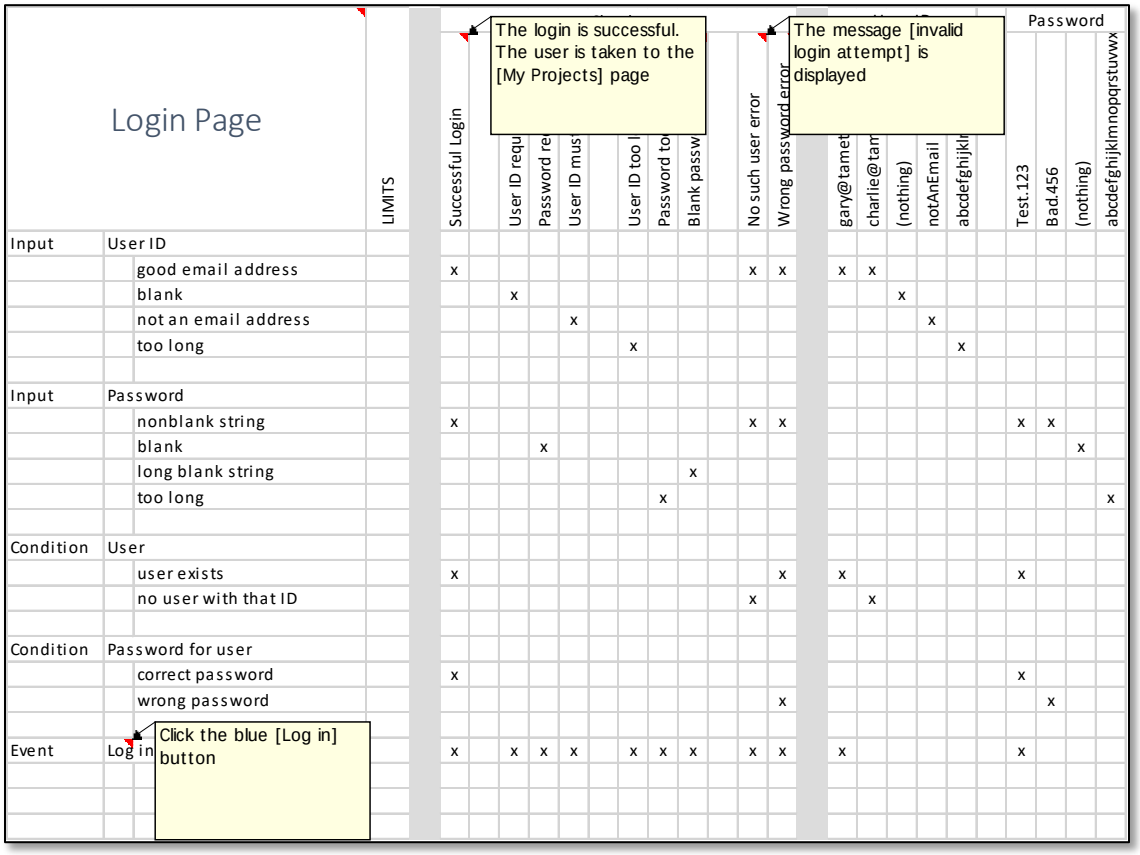
Computed Values

Values are often interdependent or dependent upon changeable conditions. Because a TAME workbook is a spreadsheet, value definitions can make use of all of the spreadsheet's formula and function capabilities.

As an example: a medical software company used TAME to produce test cases that chose random test values based upon non-linear distributions (e.g. a bell curve). They wrote the random function in Excel and TAME produced tests with random values that met their exact needs.

Descriptive Instructions

To make your tests accurate and easy to follow, add comments to the result and event cells. Such comments describe in detail what to do and what to check.



Case 4: User ID must be email

Inputs

User ID : not an email address
Password : default

Procedure

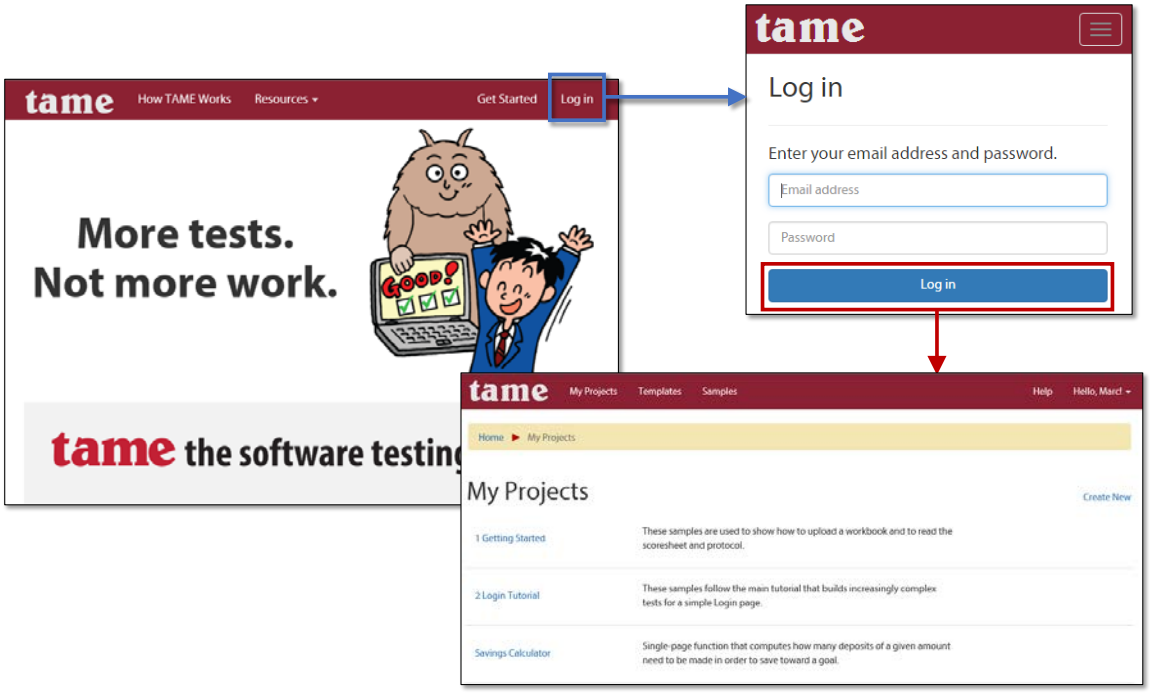
Start:
Navigate to the application's homepage and click the Login link in the upper right corner
Input:
Enter 'notAnEmail' into the User ID
Enter 'Test.123' into the Password
Do this:
Click the blue [Log in] button
Check:
The error message [The user ID must be in the form of an email address] is displayed

These are written into the test protocol as detailed instructions to the tester, making the tests almost indistinguishable from hand-written tests.

Sequences—More Extensive and Realistic Tests

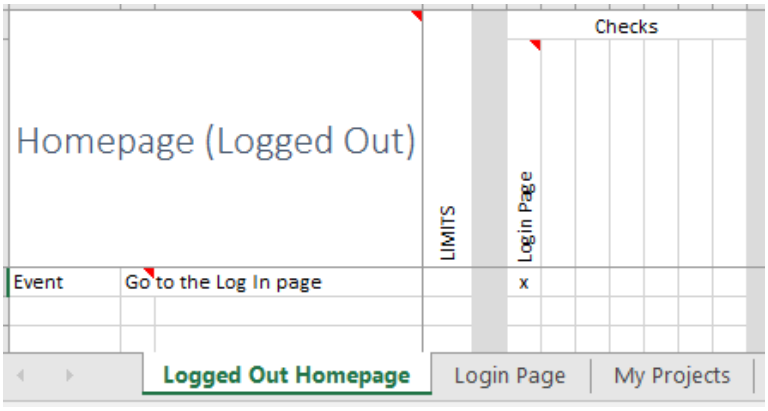
TAME can do a lot more than just build tests for single pages or functions. TAME can also build tests for whole sequences of steps.

Consider the whole sequence of logging in to TAME. You start on the TAME homepage and click the Log In menu item. That takes you to the login page, where, of course, you need to log in correctly in order to proceed to your "My Projects" page.



Writing test sequences in TAME is very easy. Just create one worksheet—one tab in a spreadsheet—for each page in the sequence.

To show how clicking the login button takes the user to the login page make the result on the homepage match the name of the tab for the login page.



TAME produces nine test cases. Note that each test case now shows the two or three step sequences in terms of the starting page, the event, and the resulting page. Here's an example:

Case 4: User ID must be email

Step 1

Go to the Log In page

Procedure

Start:
The web browser is opened to the main application homepage
Do this:
Click the [Log in] button in the upper right corner
Check:
The login page is displayed

Step 2

Log in: User ID must be email

Inputs

User ID : not an email address
Password : default

Procedure

Input:
Enter 'notAnEmail' into the User ID
Enter 'Test.123' into the Password
Do this:
Click the blue [Log in] button
Check:
The error message [The user ID must be in the form of an email address] is displayed

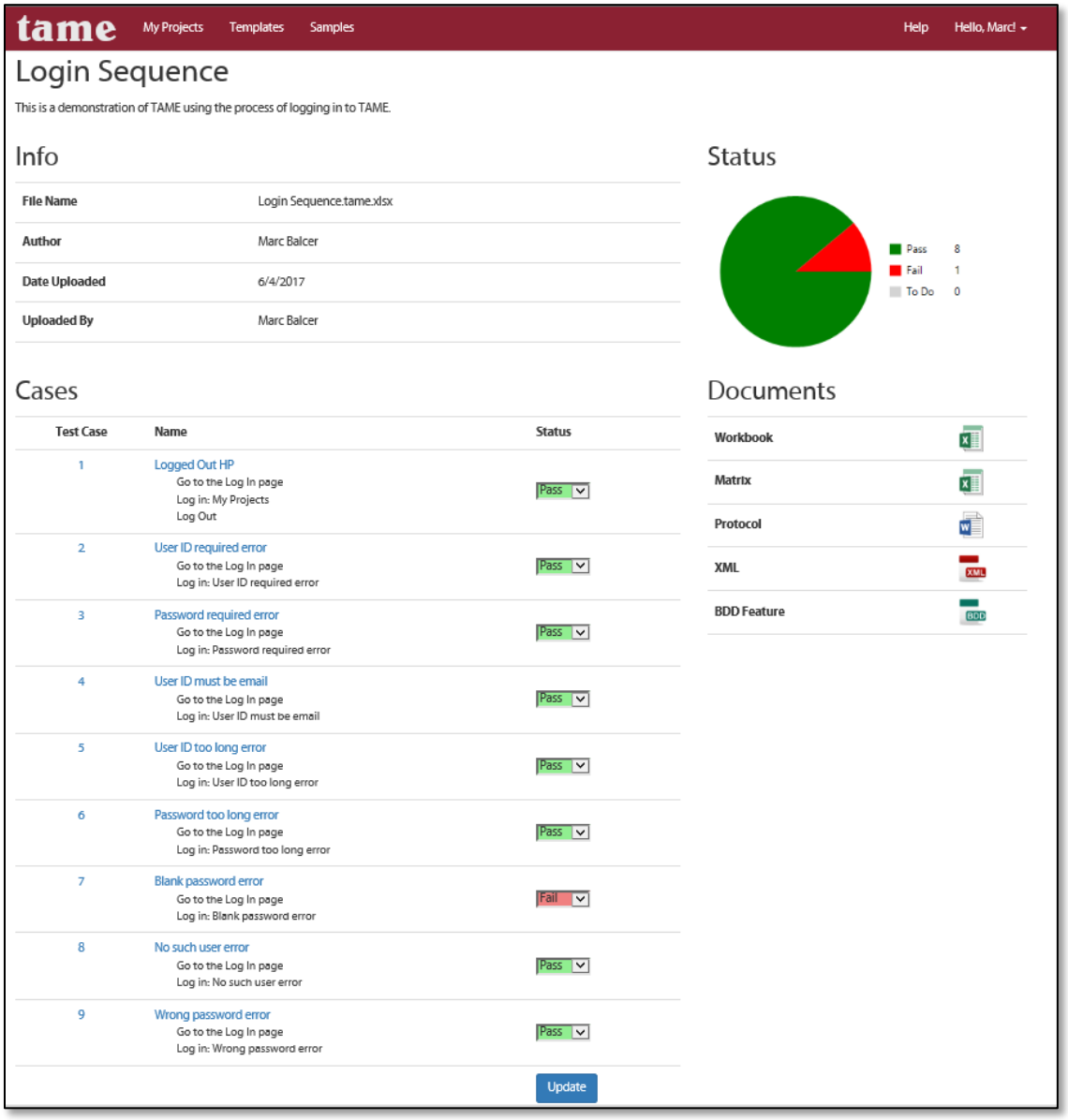
They're as precise as something you'd write by hand, but produced a lot faster than writing them by hand.

In practice when testers have to create large numbers of test cases without something like TAME, testers often have to do a lot of copying and pasting. That's a pretty boring and repetitive job and just the kind of thing that's better done by an automated tool like TAME.

Got Tests. Now What?

Now that TAME has generated tests, you or your testers can use them to manually test your application. Just follow the instructions.

TAME also provides a simple but effective test tracking capability. On the test page, mark the tests “passed” or “failed” to indicate the current status of those tests.



You can even run TAME on your smartphone—follow the instructions to test on your computer or device and then mark the test passed or failed.

Everyone on your team can see your test progress.

BDD Fast

Earlier we said that TAME can create all the Gherkin TDD scenarios from a single test workbook.

Here's the same example we've been following through this paper:

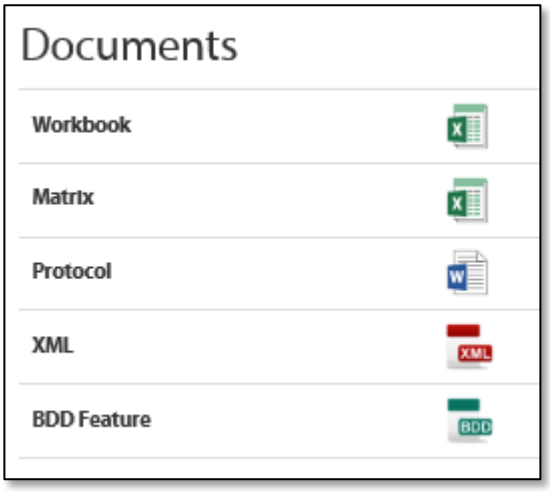
```
Scenario: User ID must be email
    Given the web browser is opened to the main application homepage
     When I click the [Log in] button in the upper right corner
     Then the login page is displayed

     When I enter 'notAnEmail' into the User ID
      And I enter 'Test.123' into the Password
      And I click the blue [Log in] button
     Then the error message [The user ID must be in the form of an email
          address] is displayed
```

This test looks very much like something you'd write by hand—but you didn't have to write each test one at a time.

What can you create with TAME?

TAME actually creates tests in several different forms.



- The matrix is a spreadsheet[1] that lists each of the test cases and test data values.
- The protocol is a single document containing all of the test cases.
- The BDD Feature file contains the test cases in Gherkin BDD format.
- The XML file contains the test cases in a format usable by automated testing tools

And you're not limited to just these forms: with a little bit of XML Stylesheet (xslt), you can create test documents specific to your needs.

[1] TAME is not limited to Microsoft Excel spreadsheets. Apache Open Office spreadsheets will work as well. If you upload an Open Office Calc spreadsheet, the protocol will be produced an Open Office Writer document.

No Bull Agile

Some have suggested that TAME means "Testing Agile Made Easy."

One of the most common and effective ways to define when development on a software feature is complete is to define a set of test cases (often referred to as "acceptance tests") that must all pass in order to declare the development work "done."

While this is a laudable goal, it's rarely achieved because of the slow pace of creating these test cases. When a team is considering a feature for inclusion in an iteration or release, the entire team can't afford to get bogged down in the process of defining these tests one at a time.

However, with TAME, a team can quickly identify the different inputs, environment conditions, and expected results. From these, TAME generates many tests in just moments. The team can then review the generated tests to determine if the scenarios need to be constructed and if it's possible to do that work in the current iteration. Test development is no longer detached and isolated—it's a key collaborative activity.

The tests define the product.

As a developer if you know what the tests will be,

- there will be no surprises when the software is tested
- you will spend time getting the right features built and avoid working on things that don't count ("gold plating")

As an analyst/product owner, you are evaluating and delivering specifics, not the "glittering generalities" of the typical requirements document[2].

[2] Of course, diehard use case mentors will point out that well written requirements in the use case form were always about testable specifics.

Summary

In this paper, we've introduced you to TAME's capabilities for defining test cases. You've seen how TAME does the boring and repetitive work of forming combinations of inputs and environment conditions and matching those up to results. This technique lets you think and plan broadly as opposed to having to grind out one scenario at a time.

In fact, the process of building TAME specs encourages analysts, developers, and testers to constantly ask "what about" throughout the whole process. It doesn't matter if you're traditional waterfall or highly agile and incremental. TAME gets you thinking better, and that better thinking leads to better, more productive tests, and better software overall.

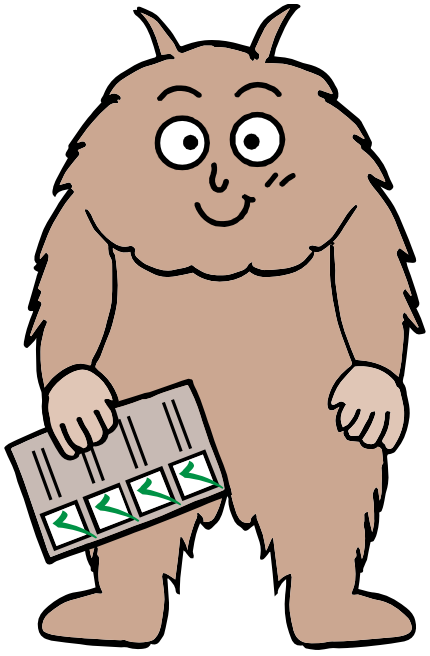
You can use TAME to produce lists of things to test and protocol documents for your testers. TAME also includes a basic test management feature that keeps track of your testing progress.

Finally, while TAME is not a system for running tests, nor for coding up automated tests, we have partnerships with test automation tool providers. TAME tests can be easily transformed into automated tests for desktop, web, and mobile applications.

The best way to see if TAME is right for you is to try applying it to your own real projects.

Go to tametest.com,
schedule an informative tutorial,
and start using TAME.

You can work through the demos
and try your hand at building
your own tests.



tame

Test Authoring Made Easy