Tests Too DRY? Make Them DAMP!

The test below follows the DRY principle ("Don't Repeat Yourself"), a best practice that encourages code reuse rather than duplication, e.g., by extracting helper methods or by using loops. But is it a well-written test?

```
def setUp(self):
  self.users = [User('alice'), User('bob')]  # This field can be reused across tests.
  self.forum = Forum()

def testCanRegisterMultipleUsers(self):
  self._RegisterAllUsers()
  for user in self.users:  # Use a for-loop to verify that all users are registered.
    self.assertTrue(self.forum.HasRegisteredUser(user))

def _RegisterAllUsers(self):  # This method can be reused across tests.
  for user in self.users:
    self.forum.Register(user)
```

While the test body above is concise, the reader needs to do some mental computation to understand it, e.g., by following the flow of `self.users` from `setUp()` through `_RegisterAllUsers()`. **Since tests don't have tests, it should be easy for humans to manually inspect them for correctness**, even at the expense of greater code duplication. This means that the DRY principle often isn't a good fit for unit tests, even though it is a best practice for production code.

In tests we can use the DAMP principle ("Descriptive and Meaningful Phrases"), which emphasizes *readability* over *uniqueness*. Applying this principle can introduce code redundancy (e.g., by repeating similar code), but it makes tests more obviously correct. Let's add some DAMP-ness to the above test:

```
def setUp(self):
  self.forum = Forum()

def testCanRegisterMultipleUsers(self):
  # Create the users in the test instead of relying on users created in setUp.
  user1 = User('alice')
  user2 = User('bob')

  # Register the users in the test instead of in a helper method, and don't use a for-loop.
  self.forum.Register(user1)
  self.forum.Register(user2)

  # Assert each user individually instead of using a for-loop.
  self.assertTrue(self.forum.HasRegisteredUser(user1))
  self.assertTrue(self.forum.HasRegisteredUser(user2))
```

Note that **the DRY principle is still relevant in tests**; for example, using a helper function for creating value objects can increase clarity by removing redundant details from the test body. Ideally, test code should be *both* readable and unique, but sometimes there's a trade-off. **When writing unit tests and faced with a choice between the DRY and DAMP principles, lean more heavily toward DAMP.**

More information, discussion, and archives:

testing.googleblog.com