Keep Tests Focused

What scenario does the following code test?

```
TEST_F(BankAccountTest, WithdrawFromAccount) {
  Transaction transaction = account_.Deposit(Usd(5));
  clock_.AdvanceTime(MIN_TIME_TO_SETTLE);
  account_.Settle(transaction);

  EXPECT_THAT(account_.Withdraw(Usd(5)), IsOk());
  EXPECT_THAT(account_.Withdraw(Usd(1)), IsRejected());
  account_.SetOverdraftLimit(Usd(1));
  EXPECT_THAT(account_.Withdraw(Usd(1)), IsOk());
}
```

Translated to English: "***(1)*** *I had $5 and was able to withdraw $5;* ***(2)*** *then got rejected when overdrawing $1;* ***(3)*** *but if I enable overdraft with a $1 limit, I can withdraw $1."* If that sounds a little hard to track, it is: **it is testing three scenarios, not one.**

A better approach is to **exercise each scenario in its own test**:

```
TEST_F(BankAccountTest, CanWithdrawWithinBalance) {
  DepositAndSettle(Usd(5));  // Common setup code is extracted into a helper method.
  EXPECT_THAT(account_.Withdraw(Usd(5)), IsOk());
}
TEST_F(BankAccountTest, CannotOverdraw) {
  DepositAndSettle(Usd(5));
  EXPECT_THAT(account_.Withdraw(Usd(6)), IsRejected());
}
TEST_F(BankAccountTest, CanOverdrawUpToOverdraftLimit) {
  DepositAndSettle(Usd(5));
  account_.SetOverdraftLimit(Usd(1));
  EXPECT_THAT(account_.Withdraw(Usd(6)), IsOk());
}
```

Writing tests this way provides many benefits:

- Logic is easier to understand because there is less code to read in each test method.
- Setup code in each test is simpler because it only needs to serve a single scenario.
- Side effects of one scenario will not accidentally invalidate or mask a later scenario's assumptions.
- If a scenario in one test fails, other scenarios will still run since they are unaffected by the failure.
- Test names clearly describe each scenario, which makes it easier to learn which scenarios exist.

One sign that you might be testing more than one scenario: after asserting the output of one call to the system under test, the test makes another call to the system under test.

While a scenario for a unit test often consists of a single call to the system under test, its scope can be larger for integration and end-to-end tests. For example, a test that a web UI can send email might open the inbox, click the *compose* button, write some text, and press the *send* button.

More information, discussion, and archives:

testing.googleblog.com