Increase Test Fidelity By Avoiding Mocks

Tuesday, February 27, 2024

This article was adapted from a Google Testing on the Toilet (TotT) episode. You can download a printer-friendly version of this TotT episode and post it in your office.

By Andrew Trenk and Dillon Bly

Replacing your code's dependencies with mocks can make unit tests easier to write and faster to run. However, among other problems, using mocks can lead to tests that are less effective at catching bugs.

The *fidelity* of a test refers to how closely the behavior of the test resembles the behavior of the production code. A test with higher fidelity gives you higher confidence that your code will work properly.

When specifying a dependency to use in a test, prefer the highest-fidelity option. Learn more in the Test Doubles chapter of the Software Engineering at Google book.

1. **Try to use the real implementation**. This provides the most fidelity, because the code in the implementation will be executed in the test. There may be tradeoffs when using a real implementation: they can be slow, non-deterministic, or difficult to instantiate (e.g., it connects to an external server). Use your judgment to decide if a real implementation is the right choice.
2. **Use a fake if you can't use the real implementation**. A fake is a lightweight implementation of an API that behaves similarly to the real implementation, e.g., an in-memory database. A fake ensures a test has high fidelity, but takes effort to write and maintain; e.g., it needs its own tests to ensure that it conforms to the behavior of the real implementation. Typically, the owner of the real implementation creates and maintains the fake.
3. **Use a mock if you can't use the real implementation or a fake**. A mock reduces fidelity, since it doesn't execute *any* of the actual implementation of a dependency; its behavior is specified inline in a test (a technique known as *stubbing*), so it may diverge from the behavior of the real implementation. Mocks provide a basic level of confidence that your code works properly, and can be especially useful when testing a code path that is hard to trigger (e.g., an error condition such as a timeout).

   (*Note: Although "mocks" are objects created using mocking frameworks such as Mockito or unittest.mock, the same problems will occur if you manually create your own implementation within tests.)*

***A low-fidelity test**: Dependencies are replaced with mocks. Try to avoid this.*	***A high-fidelity test:** Dependencies use real implementations or fakes. Prefer this.*

`@Mock OrderValidator validator;`
`@Mock PaymentProcessor processor;`
 	
`...`
 	
`ShoppingCart cart =`
`  new ShoppingCart(`
`  validator, processor);`
	`OrderValidator validator =`
	`  createValidator();`
	`PaymentProcessor processor =`
	`  new FakeProcessor();`
	
	`...`
	
	`ShoppingCart cart =`
	`  new ShoppingCart(`
	`  validator, processor);`

Aim for as much fidelity as you can achieve without increasing the size of a test. At Google, tests are classified by size. Most tests should be small: they must run in a single process and must not wait on a system or event outside of their process. Increasing the fidelity of a small test is often a good choice if the test stays within these constraints. A healthy test suite also includes medium and large tests, which have higher fidelity since they can use heavyweight dependencies that aren't feasible to use in small tests, e.g., dependencies that increase execution times or call other processes.

 

Labels: Andrew Trenk , Dillon Bly , TotT