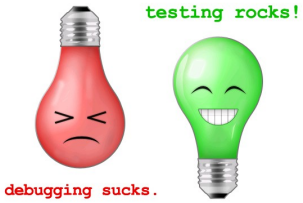

Testing on the Toilet

Keep Your Fakes Simple

Jan 22, 2009

When scientists in California tried to raise condors in captivity, they ran into a problem. The chicks wouldn't eat from the researchers' hands; they wanted a mother condor to feed them. So the scientists got a puppet. To the chicks, it looked like their mother's head was feeding them—but inside was the same scientist's hand.

Consider a contrived example based on that:

```
TEST_F(BabyCondorTest, EatsCarrion) {
  FakeCondor mother;  scoped_ptr<Carrion> carrion;  BabyCondor* pchick = &chick_;
  mother.Imprint(vector<BabyCondor*>(&pchick, &pchick + 1));  // just one chick
   while(!chick_.HasFood()) {
    mother.Eat();  // disposes of any food the mother kept for herself
    mother.Scavenge(carrion.reset(new FakeCarrion));  // finds new food
    mother.RandomlyDistributeFoodAmongYoungAndSelf();  // feeds baby or mom
  }
  chick_.Eat();
  EXPECT_TRUE(carrion->WasEaten());
}
```

Something is wrong here—that was **a lot of setup!** The general-purpose FakeCondor **replicates too much functionality** from the full class. The researchers' puppet didn't scavenge its own carrion, so why should ours? We just want to test that the baby Eats. We condense various motherhood behaviors, such as giving food, into single method calls by **extracting a role interface**. (If we couldn't change Condor, we would also write an adapter.)

```
class CondorMotherhoodRoleInterface {
 public:
  virtual Carrion* GiveFood() = 0;
  virtual SomeReturnTypes* OtherMomBehaviors() = 0;
};
```

Then we write a **single-use fake** which provides **only behaviors we need for this particular test.**

```
class CondorFeedingPuppet: public CondorMotherhoodRoleInterface {
 public:
  virtual Carrion* GiveFood() { return test_carrion_; }
  virtual SomeReturnTypes* OtherMomBehaviors() { return NULL; } // Dummy Impl.s
  Carrion* test_carrion_;  // public variable is tolerable in a one-off object
};
TEST_F(BabyCondorTest, EatsCarrion) {
  CondorFeedingPuppet mother;  FakeCarrion test_carrion;
  mother.test_carrion_ = &test_carrion;
  chick_.ReceiveFood(&mother); // inline: this->beak_->food_ = mother.GiveFood();
  chick_.Eat();
  EXPECT_TRUE(test_carrion.WasEaten());
}
```

This **highly-focused fake** is **easy and quick to write**, and makes the test much **simpler** and **more readable.** Don't overestimate the complexity of your dependencies! Often a very simple fake is the best.

More information, discussion, and archives:
http://googletesting.blogspot.com