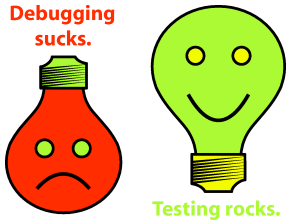

Testing on the Toilet

June 12, 2008

Friends you can depend on

When you want to test code that depends on something that is too difficult or slow to use in a test environment, swap in a **test double** for the dependency.

A **Dummy** passes bogus input values around to satisfy an API.

```
Item item = new Item(ITEM_NAME);
ShoppingCart cart = new ShoppingCart();
cart.add(item, QUANTITY);
assertEquals(QUANTITY, cart.getItem(ITEM_NAME));
```

A **Stub** overrides the real object and returns hard-coded values. Testing with stubs only is state-based testing; you exercise the system and then assert that the system is in an expected state.

```
ItemPricer pricer = new ItemPricer(){ public BigDecimal getPrice(String name){ return
PRICE; }};
ShoppingCart cart = new ShoppingCart(pricer);
cart.add(dummyItem, QUANTITY);
assertEquals(QUANTITY*PRICE, cart.getCost(ITEM_NAME));
```

A **Mock** can return values, but it also cares about the way its methods are called ("strict mocks" care about the order of method calls, whereas "lenient mocks" do not.) Testing with mocks is interaction-based testing; you set expectations on the mock, and the mock verifies the expectations as it is exercised. This example uses JMock to generate the mock (EasyMock is similar):

```
Mockery context = new Mockery();
final ItemPricer pricer = context.mock(ItemPricer.class);
context.checking(new Expectations(){{one(pricer).getPrice(ITEM_NAME);
will(returnValue(PRICE));}});
ShoppingCart cart = new ShoppingCart(pricer);
cart.add(dummyItem, QUANTITY);
cart.getCost(ITEM_NAME);
context.assertIsSatisfied();
```

A **Spy** serves the same purpose as a mock: returning values and recording calls to its methods. However, tests with spies are state-based rather than interaction-based, so the tests look more like stub style tests.

```
TransactionLog log = new TransactionLogSpy();
ShoppingCart cart = new ShoppingCart(log);
cart.add(dummyItem, QUANTITY);
assertEquals(1, logSpy.getNumberOfTransactionsLogged());
assertEquals(QUANTITY*PRICE, log.getTransactionSubTotal(1));
```

A **Fake** swaps out a real implementation with a simpler, fake implementation. The classic example is implementing an in-memory database, or using a fake BigTable.

```
Repository<Transaction> repo = new InMemoryRepository<Transaction>();
ShoppingCart cart = new ShoppingCart(repo);
cart.add(dummyItem, QUANTITY);
assertEquals(1, repo.getTransactions(cart).Count);
assertEquals(QUANTITY, repo.getById(cart.id()).getQuantity(ITEM_NAME));
```

While this episode used Java for its examples, all of the above "friends" certainly exist in C++, Python, JavaScript, and probably in YOUR favorite language as well.

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