Legacy Testing
Cheat Sheet

SOURCE ARTISTS

1 Sprout or Wrap?

Sprouting or wraping would be your two main strategies when a change is needed in your legacy code. Wrapping is safer but sprouting gives a chance for incremental refactoring which leads to reduction of technical debt.

2 Write some additional tests

Usually the part of the code you have extracted, apart from the logic you need to alter, contains additional implementation. Take advantage of that siutation and write some extra tests.

3 Don't think only about reusability

Reusability is one of the things that stops you from improving your design. Of course it is import to search for those reusable components, but reusability is just a feature of OO, not a mandatory point. Extract and test the hell out of it, thats more important.

SPROUT METHOD

1
```
/* A complex legacy method */
public void legacyMethod(){
  /* complex legacy code */

  // this part we need to change
  int discount = user.getDiscount();

 */ complex legacy code */
}
```

2
```
/* Extract code and alter it as required*/
public int resolveDiscount(User user){
  return user.getDiscount()
      + Constants.newBaseDiscount();
}
```

3
```
/* Use the sprout in legacy code */
public void legacyMethod(){
  /* complex legacy code */

  int discount =
      resolveDiscount(user);

 */ complex legacy code */
}
```

4 Unit test *resolveDiscount* method!

Work on a change in isolation without fear of breaking other code.

Makes introducing code to test harness easy thanks to extraction.

Follows the SRP rule.

POWER MOCK

```
/* Add these to you pom.xml */
<groupId>org.powermock
<artifactId>powermock-module-junit4

<groupId>org.powermock
<artifactId>powermock-api-mockito

/* Prepare a runner and the legacy class */
@RunWith(PowerMockRunner.class)
@PrepareForTest(Legacy.class)
public class LegacySutTest {
```

```
@Test
public void legacyMadePossible(){
  /* Stub private or final method */
  doReturn("test")
   .when(legacy, "privateMethod");
  /* Stub static methods */
  mockStatic(LegacyStatic.class);
  when(LegacyStatic.getString())
      .thenReturn("test");
  /* Replace new object with a mock */
  whenNew(LegacyClass.class)
  .withArguments("test")
  .thenReturn(legacyClassMock);
```

This powerful tool lets you mock: private, final, static methods and also objects crated with the usage the "new" keyword.

Remember to declare your legacy classes first in **@PrepareForTest.**

Add: *import static org.powermock.api.mockito.PowerMockito.*;*

4 Do not let customers lose trust in your software

Customers like when we add some new features. When we change or remove something while introducing bugs at the same time, we lose trust. That is why it is paramount to solidify any change with a complete suite of tests.

5 Make some sketches or mindmaps

Working with legacy code is more a process of working around design / structure than using some special unit testing techniques. It is the opposite of TDD where we just start writing tests and a design + solution follow.

6 Boy scout rule

The main idea behind this rule is to make the proximity of your change better. After writing tests, try to add missing comments on interfaces, rename variables etc. Make it a habit and a change will come.