Shigeo Shingo

- Two kinds of tests
  - tests that find defects after they occur
    (a waste of time)
  - tests to prevent defects
    (the only kind to create)
# Agile Testing Principles

- All code is tested code!
  - Teams get no credit for delivering functionality that has been coded but not tested.
- Tests are written before, or concurrently with, the code itself.
- Testing is a team effort. All developers write tests.
- Automation the the rule, not the exception.

---

# Agile Testing Strategy

<table>
<thead>
<tr>
<th>Agile Testing Strategy</th>
<th>Unit Test</th>
<th>Acceptance Test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Developers write unit tests for every class and method.</strong></td>
<td>Each unit test returns a “pass” or “fail” against the developers build.</td>
<td>All unit tests must pass before code can be checked in.</td>
</tr>
<tr>
<td>Automated unit tests are run frequently (or continuously) against an integrated build of the system.</td>
<td><strong>Testers / product owners write functional or acceptance tests for each new user story.</strong></td>
<td>Acceptance tests are run during the iteration and serve as acceptance checkpoints for the iteration’s stories.</td>
</tr>
<tr>
<td>Acceptance tests are automated whenever possible and are added to the regression test suite at each iteration.</td>
<td>Acceptance tests are elaborated and written during the iteration planning and execution.</td>
<td></td>
</tr>
</tbody>
</table>

---

Dean Leffingwell, Scaling Software Agility, 2007
Test Driven Development

• Write a test - RED
  – fails because the code doesn’t yet exist
• Make it run - GREEN
  – quickly do the clean, simple, obvious solution
  – something that works
• Make it right - GREEN
  – refactor to something you can live with
  – clean code

public void testGreatCircleDistance() {
  assertEquals(466, greatCircleDistance( 37, -102, 41, -109 ));
  assertEquals(466, greatCircleDistance( 37, -109, 41, -102 ));
}

int greatCircleDistance( double lat1, double lon1, double lat2, double lon2 ) {
  return 466;
}
Test Driven Development - Step 2

public void testGreatCircleDistance() {
    assertEquals(466, greatCircleDistance(37, -102, 41, -109));
    assertEquals(466, greatCircleDistance(37, -109, 41, -102));
    ...
}

int greatCircleDistance(double lat1, double lon1, double lat2, double lon2) {
    double distance;
    ...
    // code to compute in miles
    return Math.round(distance);
}

Test Driven Development - Step 3

public void testGreatCircleDistance() {
    // test in miles
    ...
    assertEquals(466, greatCircleDistance(37, -102, 41, -109));
    assertEquals(466, greatCircleDistance(37, -109, 41, -102));
    ...
    // test in km
    assertEquals(750, greatCircleDistance(37, -102, 41, -109));
    assertEquals(750, greatCircleDistance(37, -109, 41, -102));
    ...
}

int greatCircleDistance(double lat1, double lon1, double lat2, double lon2) {
    double distance;
    ...
    // code to compute in miles or km
    return Math.round(distance);
}
Test Driven Development in Scrum

- First we figure out what it should do
  - Write declarations / signatures
  - Write Javadoc for them
  - Write tests that describe correct behavior
- Now we know what it should do
  - Write empty definition, run tests, FAIL
  - Write simplest code to make it work, run tests, PASS
  - Refactor / Clean the code, run tests, PASS