Unit tests

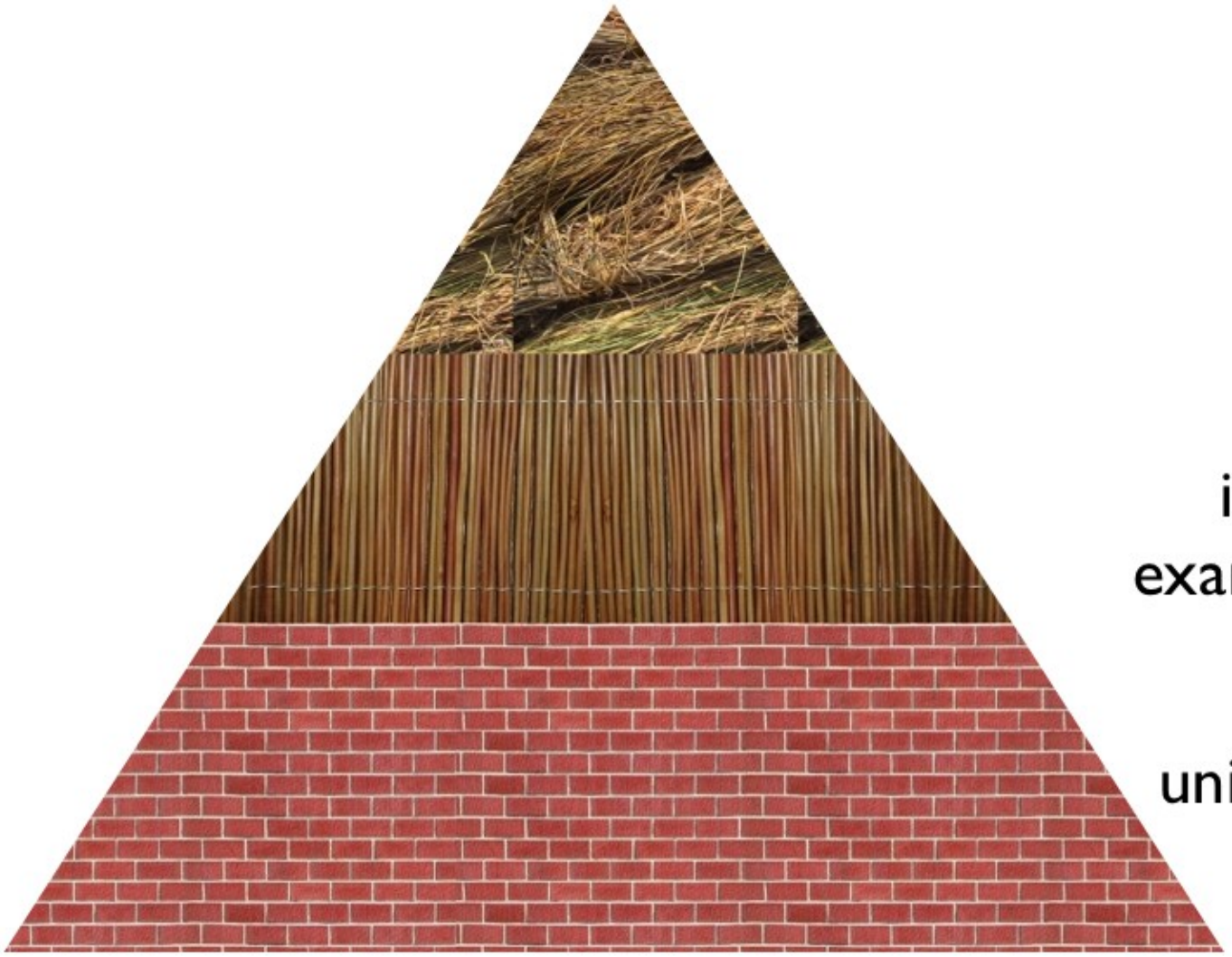
Paulo Cheque

Summer 2009

Types of tests

- Unit: Class/Module or Method/Function
- Tests that focus on features
  - Tests that focus on implementation is a common error in TDD
- Basic test but very important
- Solid and robust test
- Precise: discard debugging

Good Strategy



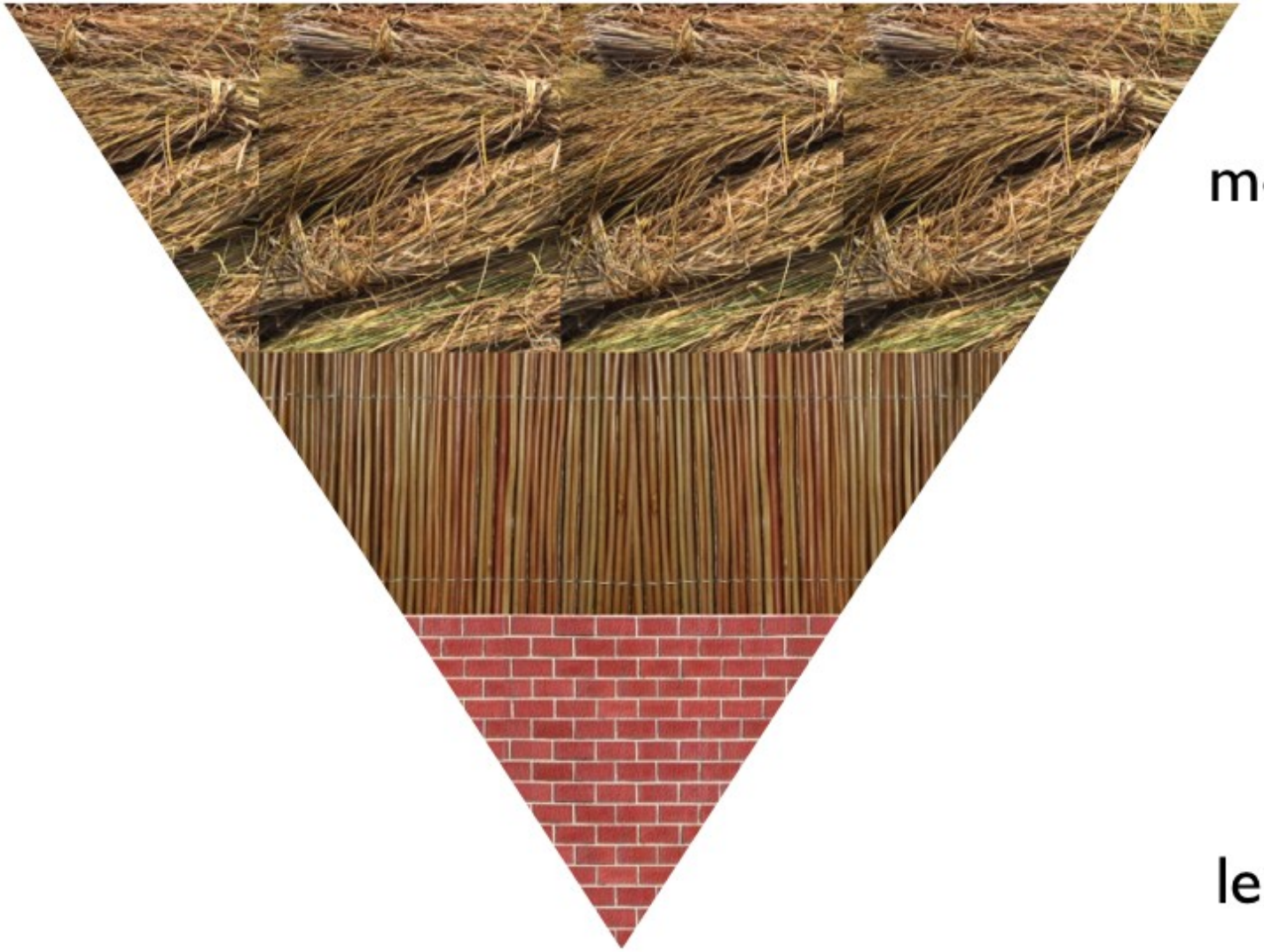
GUI

end-2-end,
integration, story,
example, acceptance

unit/micro/isolation

http://lisacrispin.blogspot.com/2008/08/agile-2008.html

Bad Strategy



bad:
most investment

bad:
least investment

http://lisacrispin.blogspot.com/2008/08/agile-2008.html

Example 1

```
@Test
public void slowAlgorithmSlowTest() {
        // ~ 13 minutes
        assertEquals(2, MathHelper.mdcSlowAlgorithm(8364823434l, 83648234349812388l);
}
```

```
@Test
public void testRestart() {
        GivenPausedTimer();
        long elapsed = timer.elapsed();
        timer.start();
       Thread.yield();
       Thread.sleep(40);  // Thread race
       assertTrue(timer.running());
       assertFalse(timer.stopped());
       assertTrue(timer.elapsed() > elapsed);
}
```

Reason

- Algorithms with high computational complexity:
  - Mathematics
  - Bioinformatics
  - Graphic computing
  - Optimization
- Concurrent algorithms
- Non isolated tests

Fast

- Hundreds of tests per second
- Must be executed tens times a day during development
  - Fast feedback

- Solution: create different test suites:
  - Default suite
  - Suite of slow tests

Example 2

```
@Test
public void testDownload() {
    WebUtils webUtils = new WebUtils(new URL("http://www.agilcoop.org.br"));
    String slide = "/portal/slides/AgilCoop-Verao2009-MetodosAgeisIntro.pdf";
    Flie file = webUtils.download(slide);
    AssertEquals(270107, arquivo.length());
}
```

Isolated

- Do not depend on other test cases ...
- ... and external factors:
  - Web servers, FTP, etc
  - Operational systems
  - Day/Time of execution test
  - ...

Example 3

```
@Test
public static getFormattedDateForInscription() {
    Date date = new Date();
    return new SimpleDateFormat("dd/MM/yyyy").format(date);
}
```

```
@Test
public void testStrictDate() {
    assertEquals("10/02/2009", DateHelper.getFormattedDateForInscription());
}
```

Reproducible

- Execution must be identical
- Bad smell: intermittent tests
- Be careful with:
  - Random
  - Statistics
  - Concurrency (race-condition)
  - etc
- Tips to Refactoring: Isolate the logic (Humble Objects)

Example 4

```
@Test
public void falsePositive1() {
    try {
        new Timer(0, 0, -1);
    } catch (IllegalArgumentException e) {
        assertTrue(true);
    }
}
@Test
public void falsePositive2() {
    try {
        new Timer(0, 0, -1);
        fail("Ops, it was expected an exception");
    } catch (Exception e) {
    }
}
```

Self-Verify

- Must verify/compare something: Success or fail
- Avoid false positives / false negatives
- Reason to exist: Avoid identical tests

Right moment to write

- Concurrently with production source code
  - Or before
- As soon as possible
- Before refactoring
- Before changes in legacy systems
  - Extensive and superficial tests

High-quality tests

- Fast
- Isolated
- Repeatable
- Self-verifying
- Timely
- -- Brett e Tim (Object Mentor)

- + Useful

Example 6

```
@Test
public void uselessTest() {
    User user = new User();
    user.setName("Fulano");
    assertEquals("Fulano", u.getName());
}
```

Useful

- 1 test => 1 source code change
- Must have a reason to exist
- Avoid ambiguous tests

PS: A lot of developers ignore tests involving limit values or some special cases that are similar to other tests. Still, it is worth verifying since the cost to write another test case is low!!!!

Object-Oriented

- Abstract classes
- Class methods
- Protected => same package
- Singletons (dangerous design)

Test the feature

- Classes
  - Anonymous
  - Private
- Methods
  - Private
- Be careful with design if you are trying to test a private method
  - Do not change the visibility to make that method testable!
  - Refactor it to make it testable

Tips

- Verify limit values

```
@Test
public void smallestFullHouseWinsBiggestFlush() {
        assertTrue(smallestFullHouse() > biggestFlush());
        assertTrue(biggestFlush() < smallestFullHouse());
}
```

- Loops, comparisons...
- Identify sets
- Cases of success / failure
- Exceptions
- Functions with inverse: inward and outward tests

+ Tips

- Lists: Full, empty, null
- String: Empty, null, long, white characters (\t, \n, space ...)
- Numbers: 0, negatives, positives, lots of decimals, minimum and maximum values (INT_MAX)
- Regular expression:
  - sequences/repeated characters, characters with diacritics (áàãâ) and special characters (#$%&), punctuation (dots, commas...), brackets, parenthesis ...

Some tools

- CxxTest (C++): http://cxxtest.sourceforge.net
- JUnit (Java): http://www.junit.org
- DUnit (Delphi): http://dunit.sourceforge.net
- VBUnit (Visual Basic): http://www.vbunit.com
- TestNG (Java): http://testng.org
- RSpec (Ruby): http://rspec.info/

- http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

+Tools

- For particular cases:
  - Testing static methods
    - https://jmockit.dev.java.net/
  - Testing private methods
    - http://sourceforge.net/projects/privaccessor

Contact

http://www.agilcoop.org.br
http://ccsl.ime.usp.br
http://qualipso.org

agilcoop@agilcoop.org.br
paulocheque@agilcoop.org.br