

Testing code

Code coverage



Learn. To Change.

```
// persisted properties <html> <p style="font-weight:bold;">HTML font code is done</p>
<html> <body style="background-color:yellowgreen;color:white;text-align:center;text-align:justify">
<html>text - :200px;> <.todolistid = data.todoId; todoListId = data.todoId>
// Non - text - :200px;>persisted properties<html> <errorMessage = ko , observable() ; >
<p style="color:orange;">HTML font code is done</p>
function todoitem(data) {
  var self = this <html> <errorMessage = ko , observable() ; >
  data = data || <html> <errorMessage = ko , observable() ; >
  // Non - persisted propertie function todoitem<html> <errorMessage = text - :200px;">ko , observable() ; >
  <p style="font-weight:bold;">HTML font code is done</p>
  <body style="background-color:yellowgreen;color:white;text-align:center;text-align:justify">
  text - :200px;> <.todolistid = data.todoId; todoListId = data.todoId>
  - text - :200px;>persisted properties<html> <errorMessage = ko , observable() ; >
```

Objective



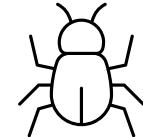
Is to explore:

- What is code coverage
- How it is measured and
- Why it is important

What is Code Coverage?

Is to measure how much of the code is tested

if your tests only cover 50% of your code, there could be bugs hiding in the untested parts.



```
public int divide(int a, int b) {  
    if (b == 0)  
        throw new IllegalArgumentException ("Cannot divide by zero");  
    return a / b;  
}
```

Just testing **divide(4,2)** means you have partial coverage

Test **divide(4,0)** also to ensure all paths are covered.

Higher code coverage reduces bugs.

What is Code Coverage?

How well do the tests cover the code base?

- **White box tests** – you should cover every line of code
- **Tools:** Clover and Istanbul
- **Acceptable percentage of coverage**
 - **Ordinary development:** struggle to get to 90%
 - **TDD:** should naturally be at least 90%

Aggregate Packages	Packages	Average Method Complexity	TOTAL Coverage	
org.easymock.tests	1	1,46	91%	<div style="width: 91%; background-color: green; height: 10px;"></div> <div style="width: 9%; background-color: red; height: 10px;"></div>
org.easymock	2	1,46	92,4%	<div style="width: 92,4%; background-color: green; height: 10px;"></div> <div style="width: 7,6%; background-color: red; height: 10px;"></div>
org	2	1,46	92,4%	<div style="width: 92,4%; background-color: green; height: 10px;"></div> <div style="width: 7,6%; background-color: red; height: 10px;"></div>

COVERAGE METRICS



1. Method coverage

- % of **methods** covered
- Very coarse-grained, i.e. inaccurate

2. Symbol coverage

- % of sequence points (**statements**) that have been exercised
- doesn't show if both sides to an 'if' condition have been covered.

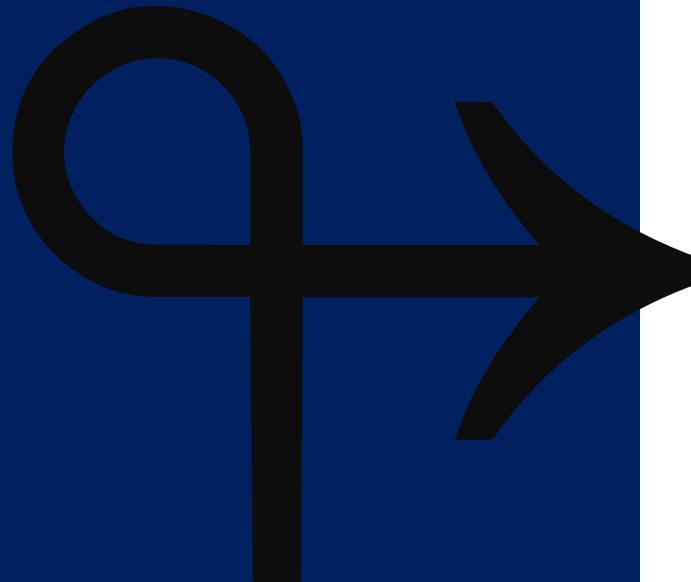
3. Branch coverage

- % of completely executed blocks in a method
- Both side of an IF statement is covered

4. Cyclomatic complexity

- How many linearly independent paths through the code
- Reflects the number of tests to get 100% coverage
- Refactor If over 15 in a method

percentage isn't everything



'100% test coverage doesn't guarantee that your app is 100% tested'

– Massol

- **Test generation tools** which auto-generate unit tests with 100% coverage, but the tests are worthless
- Detects regions of code which are not tested
- Test class **full of trivial Get / Set methods** which make it hard to see where the tests with real value are.
your tests must challenge your code to be meaningful!
- **Failing test** counts towards coverage the same as passing test

Quality over quantity



'The more tests you have, the better' – FALSE!

- Jester – JUnit test tester (<http://jester.sourceforge.net/>)

The screenshot shows a software interface for running JUnit tests. At the top, there's a blue header bar with the word "Progress" and a red progress bar. Below the bar is a code editor window containing Java code. The code is part of a Person class and includes a compareTo method. The code is partially visible, showing:

```
qa\tdd\junitintro\Person.java - changed source on line 20 (char index=570) from if ( to if (false &&
compareTo(Object other_) {
    >>>if (! (other_ instanceof Person)) throw new C
```

- May produce false positives

Lab

