Mock Objects

Maurício F. Aniche
M.F.Aniche@tudelft.nl
That’s how it is in OO systems...

A does too much!
That’s how it is in OO systems...

A does too much again!

A

C
That’s how it is in OO systems...
What should we do to test a class without its dependencies?

How to write unit tests for A?
We simulate the dependencies!

- Fast
- Full control

B’ and C’ are (lightweight) simulations of B and C, respectively.
Mock Objects

mock objects are objects that mimic the behavior of real objects/dependencies, easing their controllability and observability.
Why do I want to control my dependencies?

• To easily simulate exceptions
• To easily simulate database access
• To easily simulate the interaction with any other infrastructure
• To avoid building complex objects
• To control third-party libraries that I do not own
Let’s code!

I wanna filter all the invoices where their value are smaller than 100.0. Invoices come from the database.

Code: https://gist.github.com/mauricioaniche/03ee12e64d734e7ea370eceb68fe6676
To mock or not to mock?

- Developers have mixed feelings about the usage of mock objects.

- Can you see the advantages and the disadvantages of using mocks?
  - Adv: Easy to control dependencies, easy to automate test cases.
  - Disadv: Not very real, integration problems might pass.

- At the end of the day, it’s about using the right tool at the right moment.
When to mock?

• We empirically see that infrastructure is often mocked.
• There was no clear trend on domain objects.
  • Their practice: Complicated/complex classes are mocked.
• No mocks for libraries (e.g., lists or small util methods).
Mocks are introduced from the very beginning of the test class!

<table>
<thead>
<tr>
<th>Mockito API</th>
<th>Spring</th>
<th>Sonarqube</th>
<th>VRaptor</th>
<th>Alura</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mocks introduced from the beginning</td>
<td>234 (86%)</td>
<td>1,485 (84%)</td>
<td>177 (94%)</td>
<td>263 (74%)</td>
<td>2,159 (83%)</td>
</tr>
<tr>
<td>Mocks introduced later</td>
<td>37 (14%)</td>
<td>293 (16%)</td>
<td>12 (6%)</td>
<td>91 (26%)</td>
<td>433 (17%)</td>
</tr>
<tr>
<td>Mocks removed from the tests</td>
<td>59 (22%)</td>
<td>243 (14%)</td>
<td>6 (3%)</td>
<td>35 (10%)</td>
<td>343 (13%)</td>
</tr>
</tbody>
</table>
50% of changes in a mock occur because the production code changed! **Coupling is strong!**

<table>
<thead>
<tr>
<th>Type of change</th>
<th>Spring-framework</th>
<th>Sonarqube</th>
<th>VRaptor</th>
<th>Alura</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>57</td>
<td>20</td>
<td>42</td>
<td>45</td>
<td>164</td>
</tr>
<tr>
<td>Production API</td>
<td>19</td>
<td>33</td>
<td>36</td>
<td>25</td>
<td>113</td>
</tr>
<tr>
<td>Production internal implementation</td>
<td>8</td>
<td>22</td>
<td>15</td>
<td>14</td>
<td>59</td>
</tr>
</tbody>
</table>
No single metric explains why a class is mocked.
Developers are aware of the trade-offs!

- They mock databases, but then later write integration tests.
- Added “complexity” in exchange of testability.
- They understand how coupled they are when they use mocks.
Let’s code!

The remaining invoices should be sent to our Webservice!
Solution

• Final implementation:

https://gist.github.com/mauricioaniche/ca143c74f7a788e7e42644af74b472de