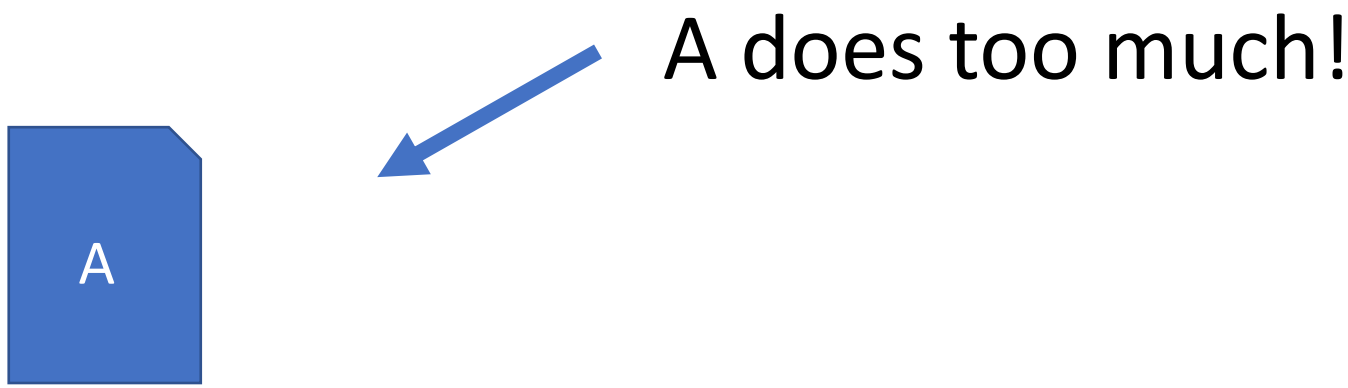
Mock Objects

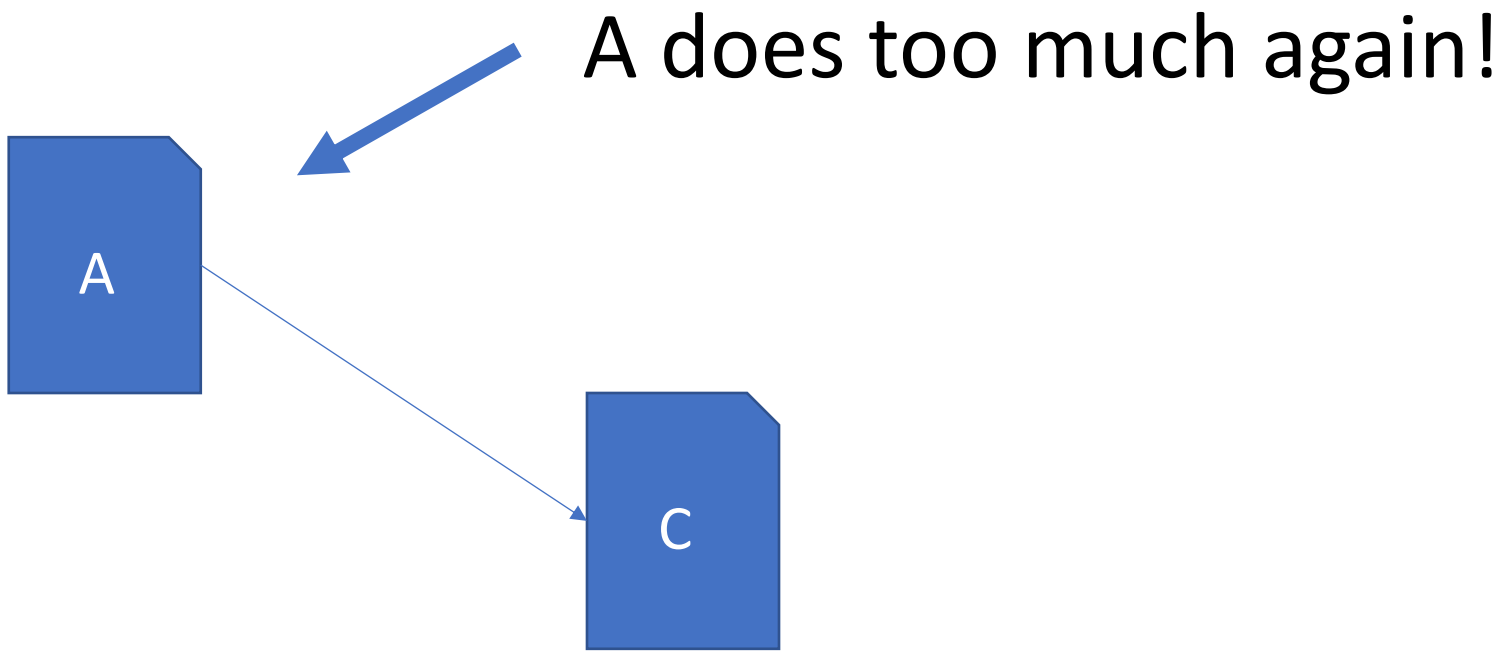
Maurício F. Aniche

M.F.Aniche@tudelft.nl

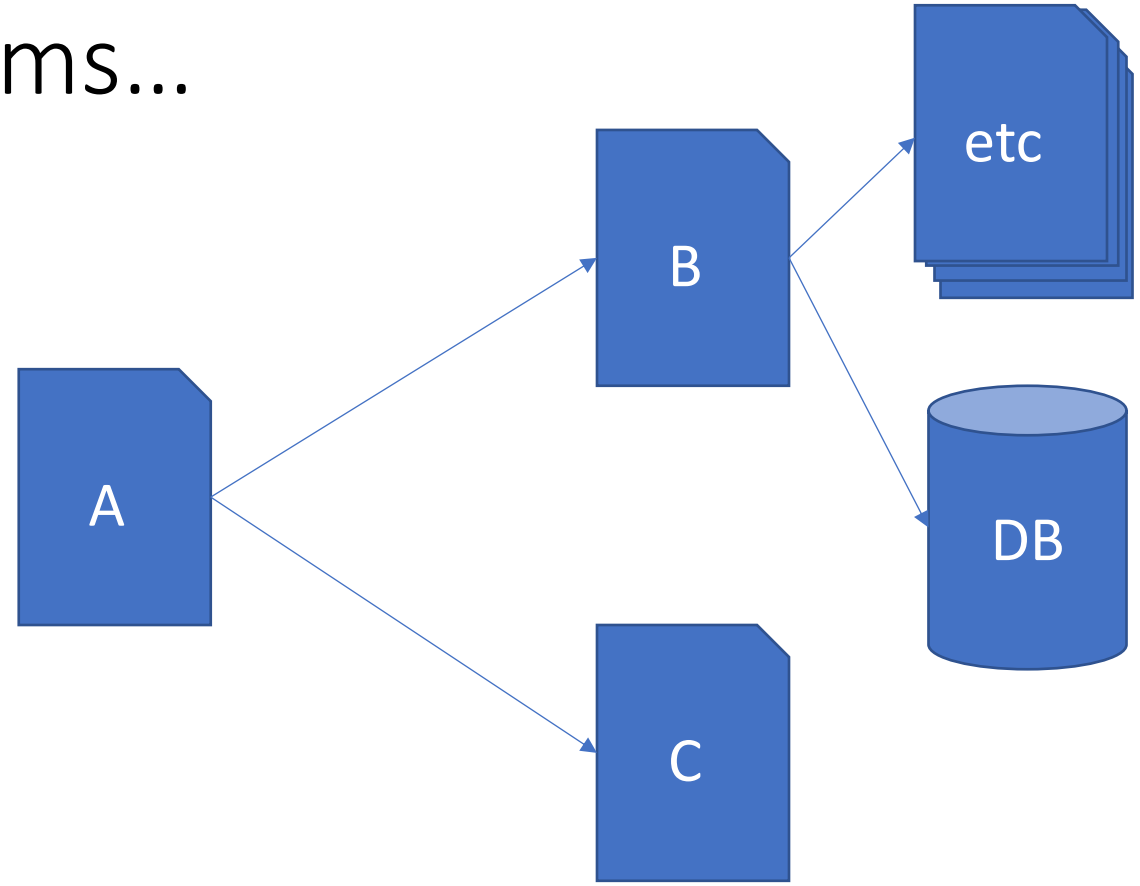
That’s how it is in OO systems…



That’s how it is in OO systems…

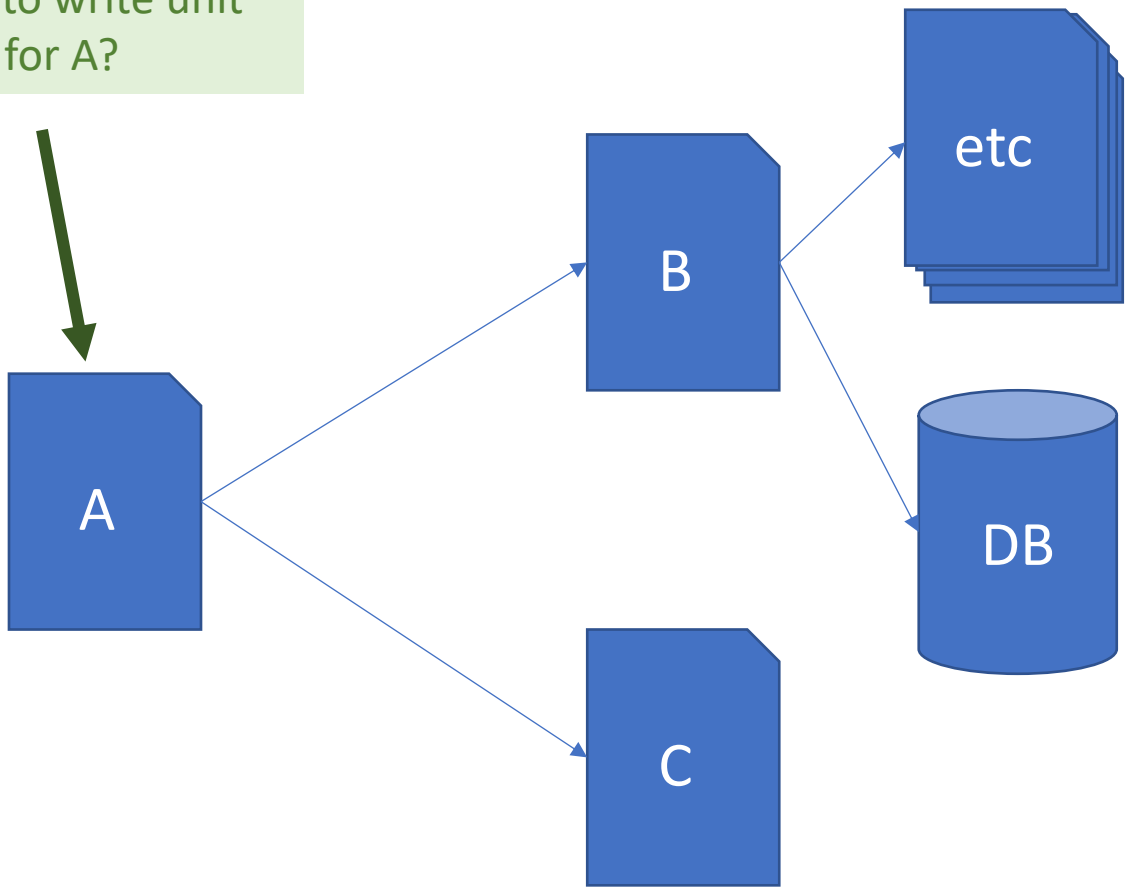


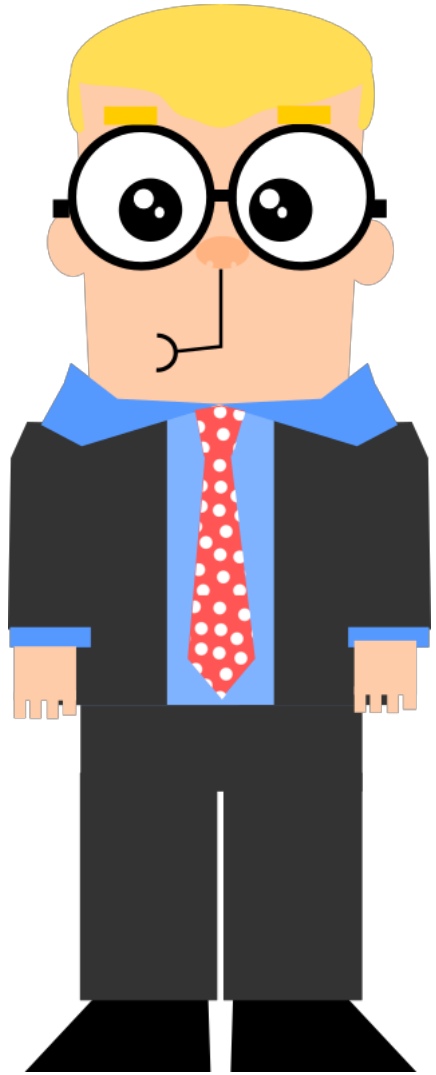
That’s how it is in OO systems…



What should we do to test a class without its dependencies?

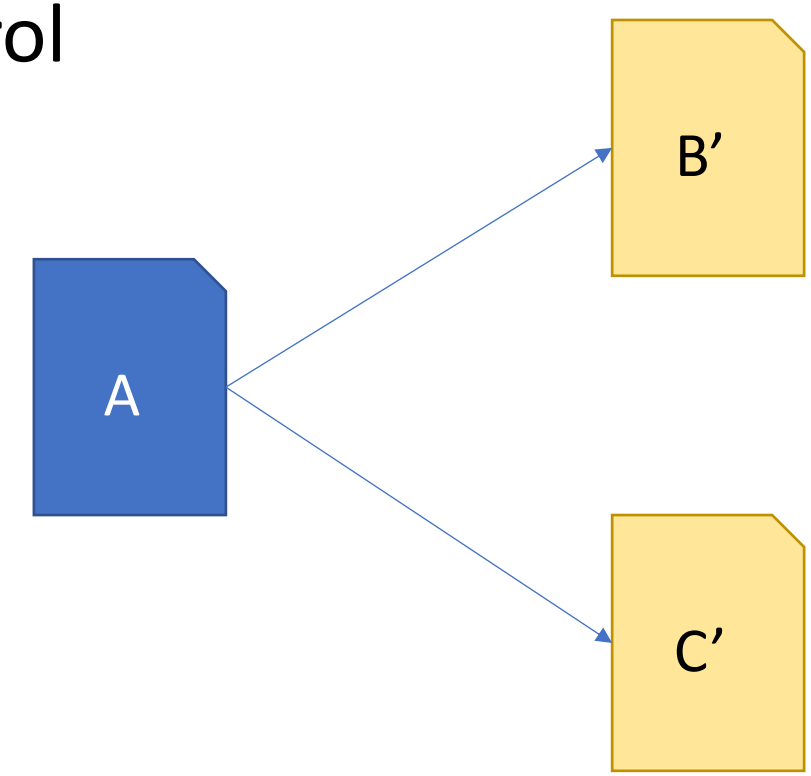
How to write unit tests for A?



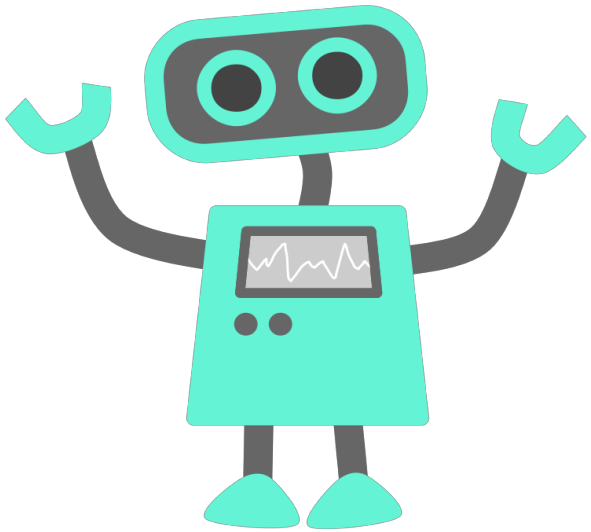


We simulate the dependencies!

- Fast
- Full control



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

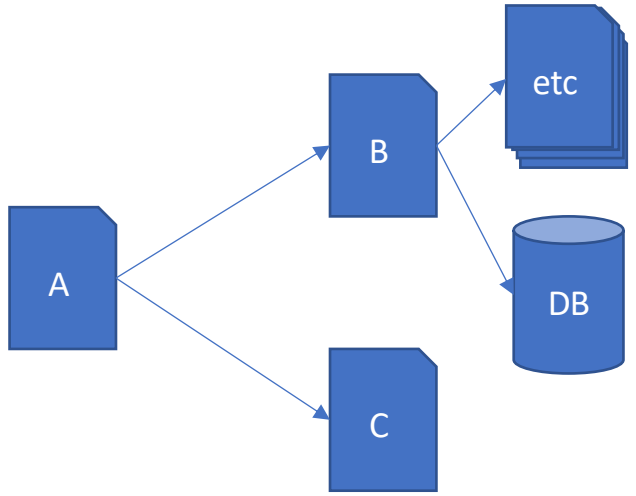


Mock Objects

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.

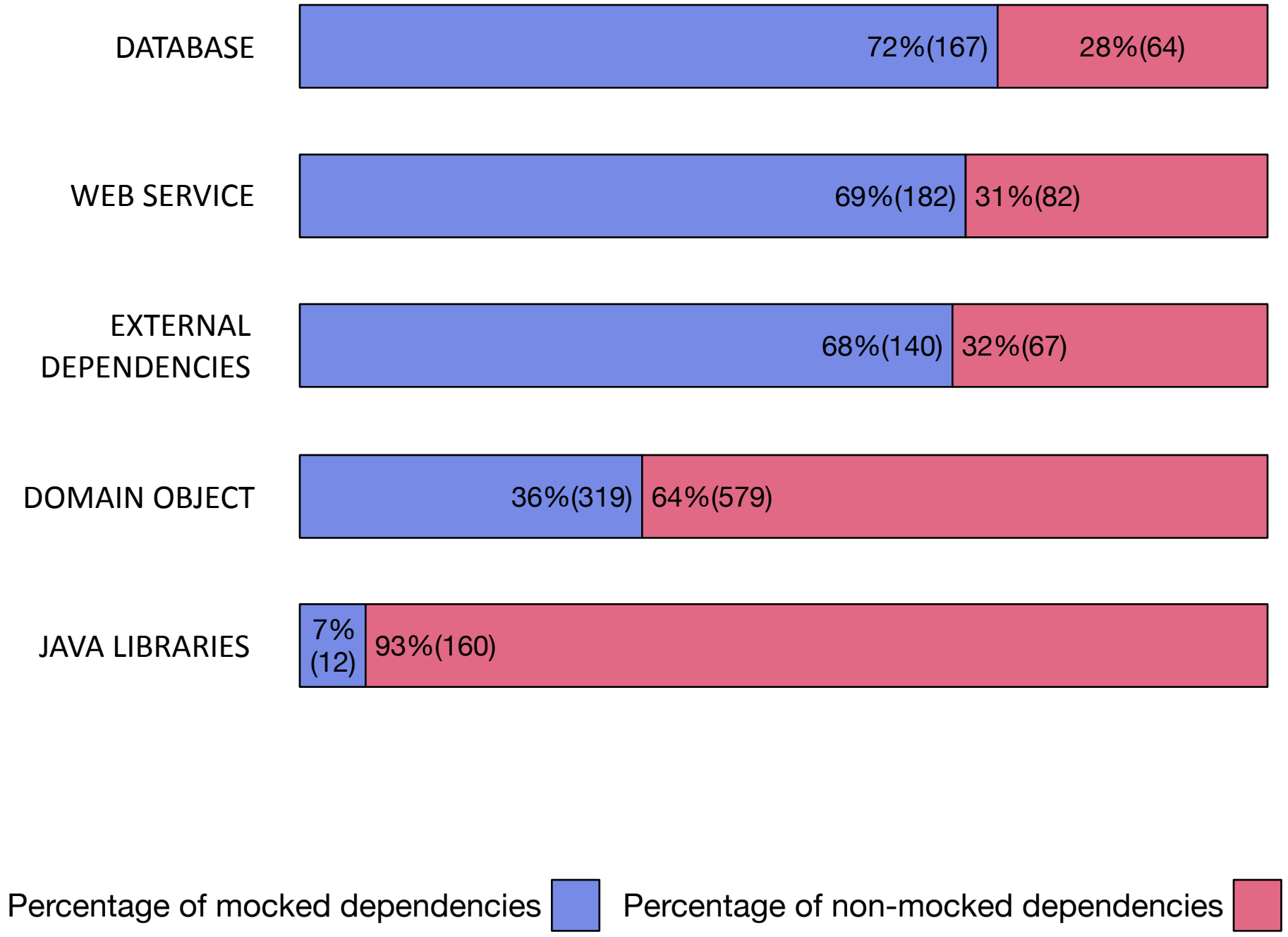
Code: https://gist.github.com/mauricioaniche/03ee12e64d734e7ea370eceb68fe6676

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.


DATABASE
72%(167)
28%(64)
WEB SERVICE
69%(182)
31%(82)
EXTERNAL DEPENDENCIES
68%(140)
32%(67)
DOMAIN OBJECT
36%(319)
64%(579)
JAVA LIBRARIES
7% (12)
93%(160)
Percentage of mocked dependencies
Percentage of non-mocked dependencies

When to mock?

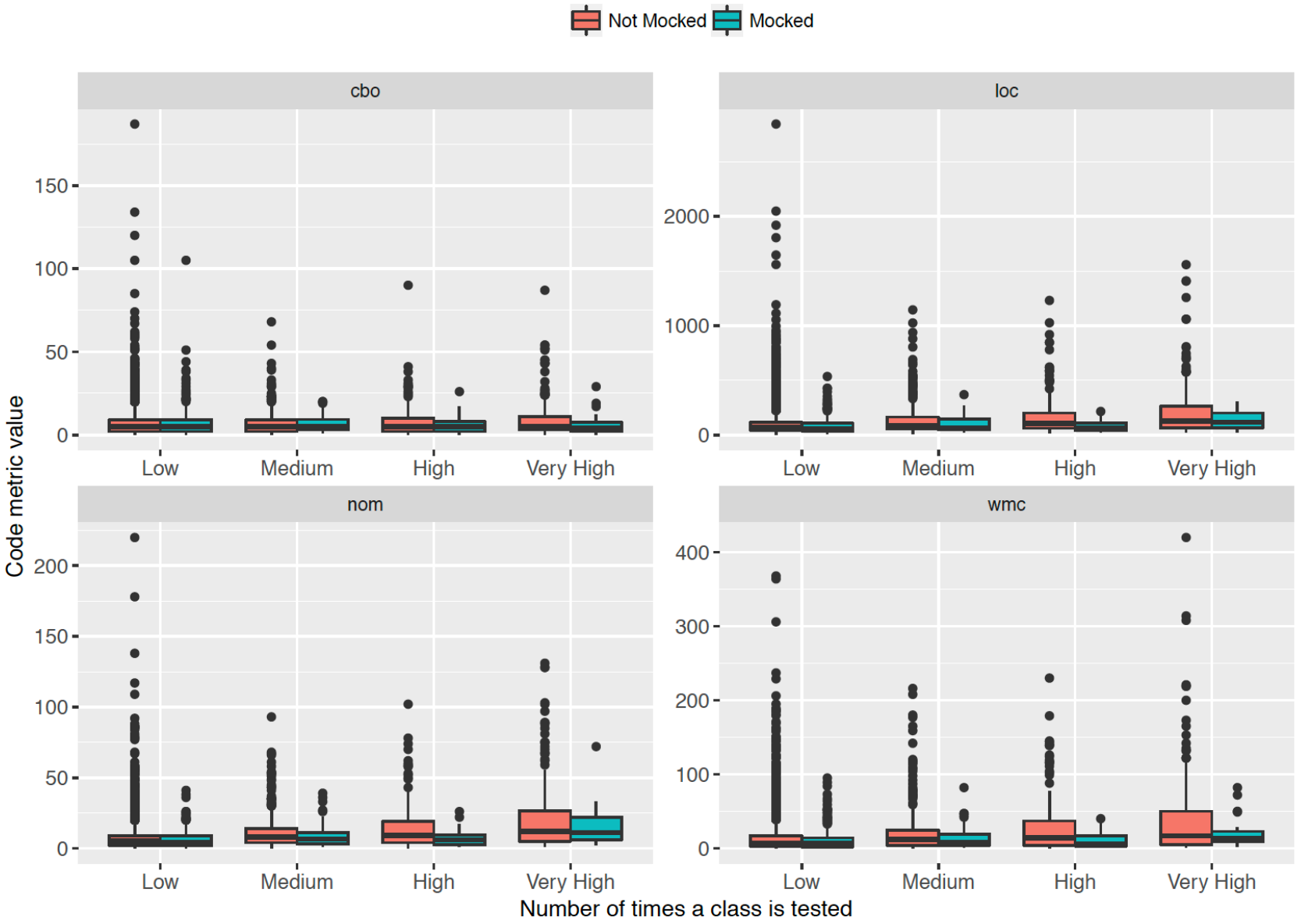
- We empirically see that infrastructure is often mocked.
- There was no clear trend on domain objects.
  - Their practice: Complicated/complex classes are mocked.
- No mocks for libraries (e.g., lists or small util methods).

Mocks are introduced from the very beginning of the test class!

Mockito API	Spring	Sonarqube	VRaptor	Alura	Total
Mocks introduced from the beginning	234 (86%)	1,485 (84%)	177 (94%)	263 (74%)	2,159 (83%)
Mocks introduced later	37 (14%)	293 (16%)	12 (6%)	91 (26%)	433 (17%)
Mocks removed from the tests	59 (22%)	243 (14%)	6 (3%)	35 (10%)	343 (13%)

50% of changes in a mock occur because the production code changed! **Coupling is strong!**

Type of change	Spring-framework	Sonarqube	VRaptor	Alura	Total
Test	57	20	42	45	164
Production API	19	33	36	25	113
Production internal implementation	8	22	15	14	59



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.

Davide Spadini, M. Finavaro Aniche, Magiel Bruntink, Alberto Bacchelli.
To Mock or Not To Mock? An Empirical Study on Mocking Practices. MSR 2017.
Mock Objects For Testing Java Systems: Why and How Developers Use Them, and How They Evolve. EMSE, 2018.

Let’s code!

The remaining invoices should be sent to our Webservice!

Solution

- Final implementation:

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