Design for Testability

Maurício F. Aniche

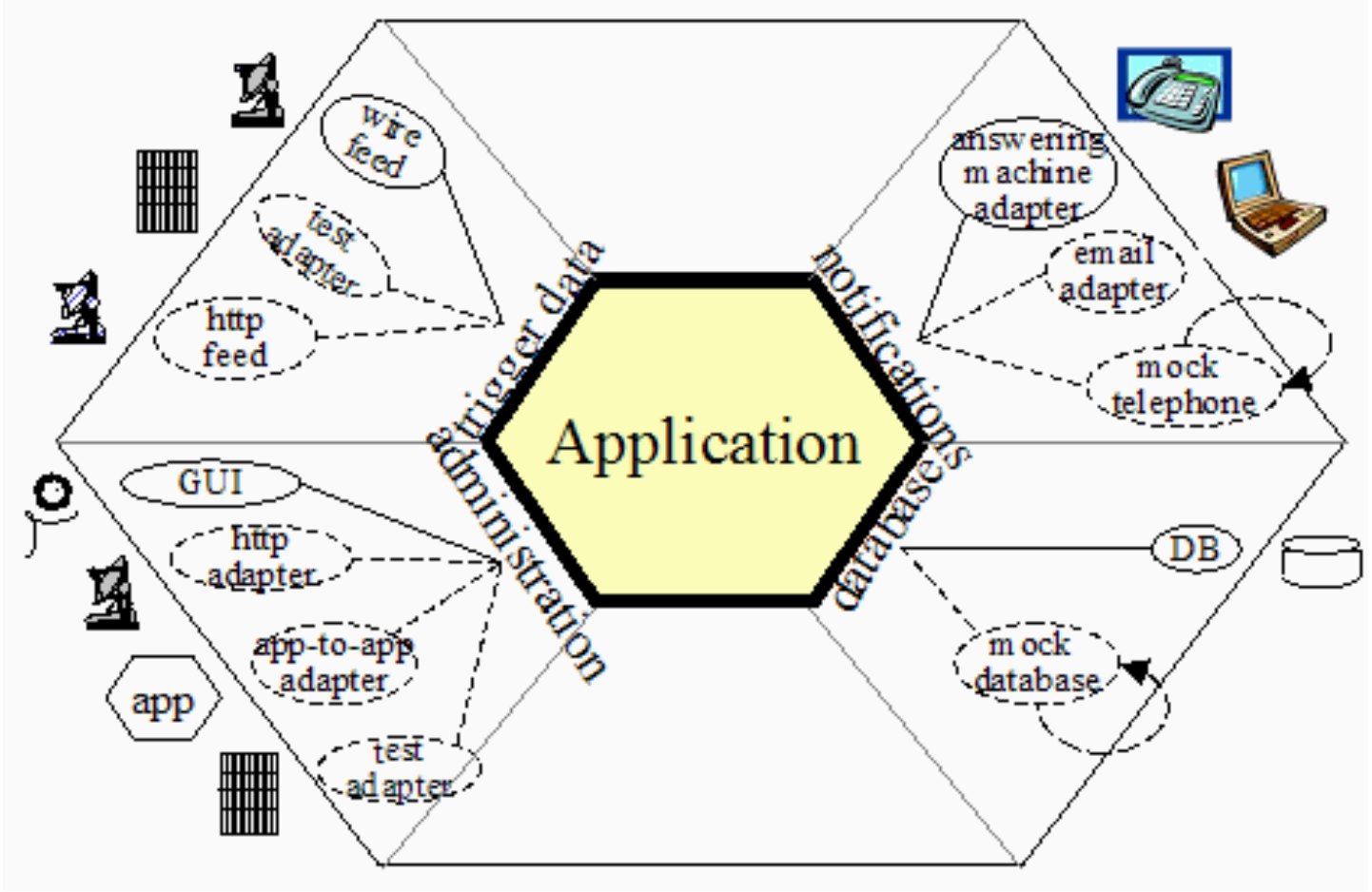
M.F.Aniche@tudelft.nl

Controllability and Observability

- **Controllability** determines the work it takes to set up and run test cases and the extent to which individual functions and features of the system under test (SUT) can be made to respond to test cases.
- **Observability** determines the work it takes to set up and run test cases and the extent to which the response of the system under test (SUT) to test cases can be verified.

- See Robert Binder's post on Testability: https://robertvbinder.com/software-testability-part-2-controllability-and-observability/

Hexagonal architecture
(aka ports and adapters)



Extracted from http://alistair.cockburn.us/Hexagonal+architecture

Harder than it seems!

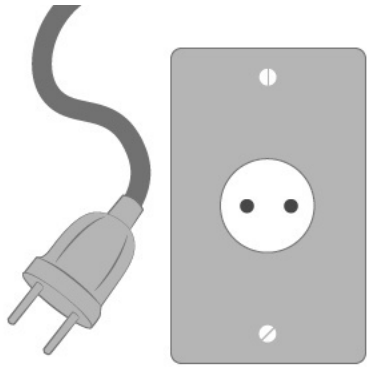
Clear and well-defined interfaces

Encapsulation plays a big role here!

Dependency Inversion

- Make dependencies injectable.

```
public class InvoiceFilter {

    private InvoiceDao invoiceDao;
    private Webservice webservice;

    public InvoiceFilter(InvoiceDao invoiceDao, Webservice webservice) {
        this.invoiceDao = invoiceDao;
        this.webservice = webservice;
    }

    public List<Invoice> filter() {

        List<Invoice> filtered = new ArrayList<>();

        List<Invoice> allInvoices = invoiceDao.all();

        for(Invoice inv : allInvoices) {
            if(inv.getValue() < 100.0) {
                filtered.add(inv);
                webservice.send(inv);
            }
        }

        return filtered;

    }
}
```



Don’t Ask

```
double tax = 0;
if(invoice.getValue() < 100)
    tax = invoice.getValue() * 0.01;
else
    tax = invoice.getValue() * 0.02;
```

```
double tax = invoice.calculateTax();
```

```
public double calculateTax() {
    double tax = 0;
    if(this.getValue() < 100)
        tax = this.getValue() * 0.01;
    else
        tax = this.getValue() * 0.02;

    return tax;
}
```

Tell!

Encapsulation

- Behavior should be in the right place.
  - Otherwise, test gets too complicated.
  - Indirect testing smell.
- Law of Demeter
  - Avoid *a.getB().getC().getD().doSomething();*
  - Much harder to arrange the test, i.e., you have to arrange states in A, B, C, and D.

Should I test a private method?

- Probably not.
- You should test the class from its public interface.
- If you want to test the private method in an isolated way, that’s the test telling you that your code is not cohesive.

Static methods

- Now that you know mock, what can we do with static methods?
- Static methods can’t be mocked.
  - Therefore, **AVOID THEM**.

Don’t be afraid of creating layers

- How can I test date/time related things?
- How can I test environment-dependent things?

- Create a layer on top of APIs
- These layers are easily mockable

- The Restfulie.NET case:
- https://github.com/mauricioaniche/restfulie.net/blob/master/Restfulie.Server/Marshalling/UrlGenerators/AspNetMvcUrlGenerator.cs

Let’s code

- The *Clock abstraction* example.
- https://gist.github.com/mauricioaniche/312249471a71dddeeb74991774529b7b

Solution

- We added a layer on top of Calendar.
  - Again, do not be afraid of adding layers to facilitate testability.

- Solution: https://gist.github.com/mauricioaniche/2200d8c1dcab41e7c4dddcc46211f4c0

Guide: Writing Testable Code

To keep our code at Google in the best possible shape we provided our software engineers with these constant reminders. Now, we are happy to share them with the world.

Many thanks to these folks for inspiration and hours of hard work getting this guide done:

- Jonathan Wolter
- Russ Ruffer
- Miško Hevery

Available online at: http://misko.hevery.com/code-reviewers-guide/

http://misko.hevery.com/attachments/Guide-Writing%20Testable%20Code.pdf

Some of the flaws listed in the Testability Guide, by Hevery

- Flaw: Constructor does Real Work
- Flaw: Digging into Collaborators
- Flaw: Brittle Global State & Singletons
- Flaw: Class Does Too Much

- His tool, Testability Explorer (https://github.com/mhevery/testability-explorer) never became a hit.
  - I'm looking for BSc and MSc students who want to revive this idea!

References

- **Compulsory:** Fowler. M. Mocks aren't Stubs. https://martinfowler.com/articles/mocksArentStubs.html
- Freeman, S., & Pryce, N. (2009). *Growing object-oriented software, guided by tests*. Pearson Education.
- Martin, R. C. (2002). *Agile software development: principles, patterns, and practices*. Prentice Hall.
- **Compulsory:** Cockburn, A. Hexagonal architecture. http://alistair.cockburn.us/Hexagonal+architecture
- **Compulsory:** Hevery, M. *Guide: Writing testable code*. http://misko.hevery.com/attachments/Guide-Writing%20Testable%20Code.pdf