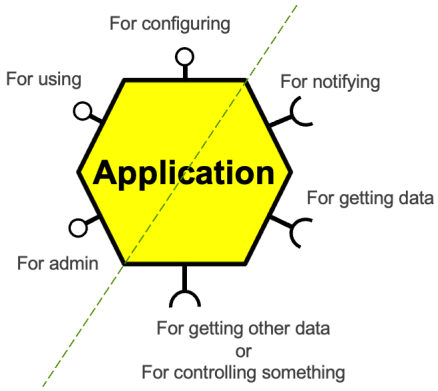
Hexagonal Architecture
(Ports & Adapters)

Alistair Cockburn

with thanks to
Juan Manuel Garrido de Paz



Your challenge:
Program a simple Ports & Adapters app during this talk!

Choose your own app, or
Bios of German poets

Please

- *ATDD or at least have & show the tests*
- *OK just 1 user service (small is ok, the structure is key)*
- *At least 1 driven actor (secondary actor in use case terms)*
- *Easy to read – the structure is what we are after*
- *Folder structure is important (for readability)*

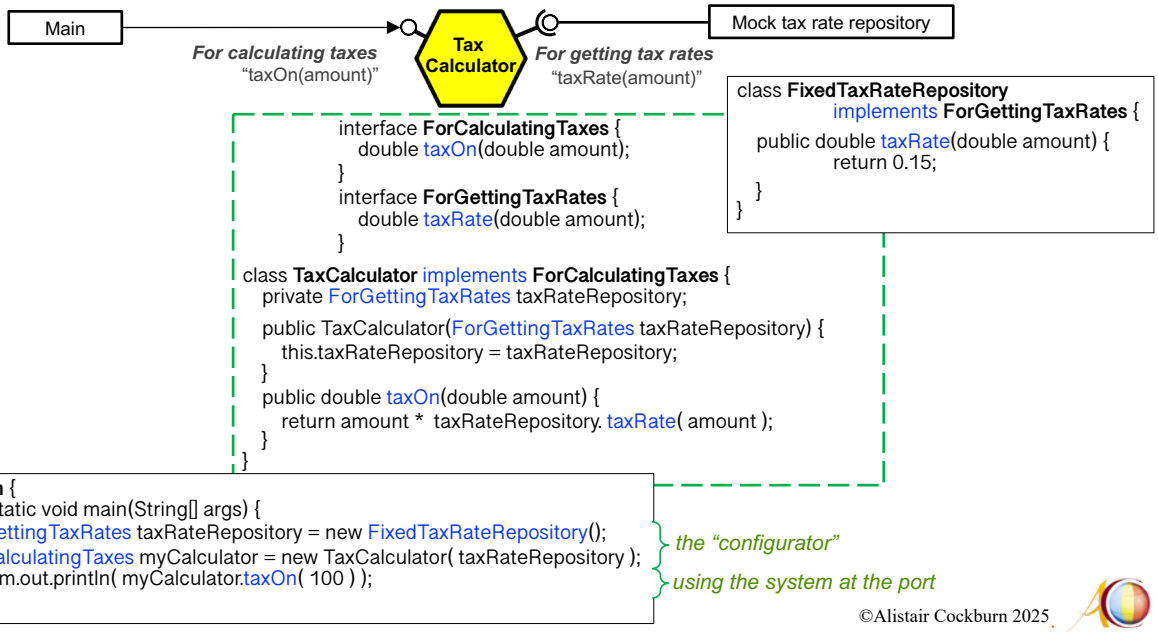
and show us the code at the end of this talk

What is the point?

Create your application to work without either a UI or a database so that

- *you can run automated regression-tests against it,*
- *work when the database becomes unavailable,*
- *upgrade to new technology,*
- *link applications together, and*
- *protect your code from leakage between business and I/O.*

tl;dr: Just write this code (Java)



Benefits

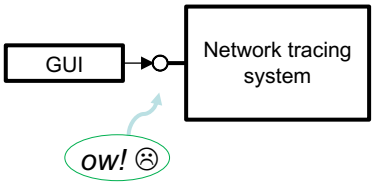
1. You get to set the app's driven actors at initialization,
   or change them in real time while it's running,
   or over a period of years as technologies shift.
2. You get to replace production connections with test harnesses, and back again, without changing the source code.
3. You get to avoid having to change the source code and then rebuild the system every time you make these shifts.
4. You can prevent leaks of business logic into the UI or data services, & vice versa, leaks of UI or data service logic into the business logic.

Costs

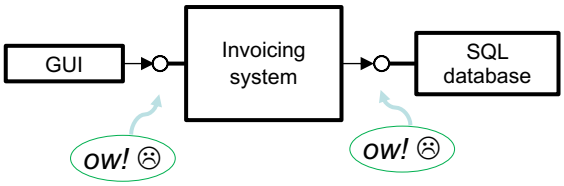
1. You must add an instance var to hold each driven actor,
   or get it every time.
2. You must add a constructor parameter for each driven actor,
   or a setter function,
   or a call to the configurator to get it.
3. You must design and add a configurator.
4. *(Type-checked languages)* You must declare the *"required"* interfaces.
5. *(Type-checked languages)* You must add folder structure for the port declarations.

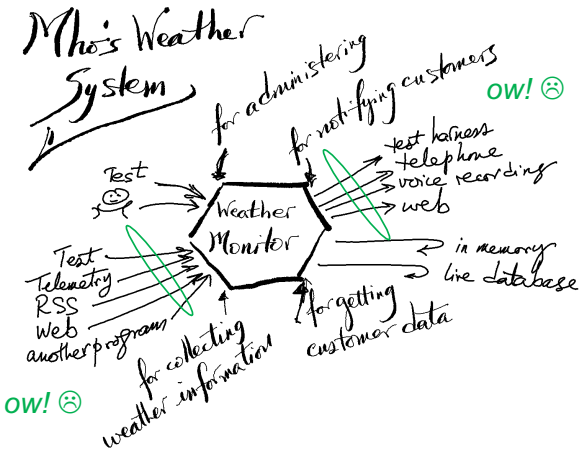
When would it have been worth it?

Network tracing program

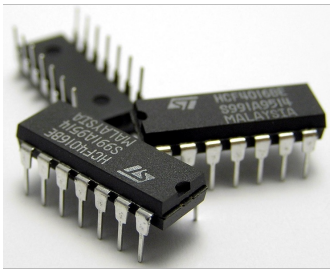


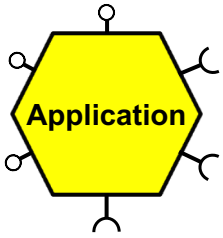
Invoicing system





The app is a "component"

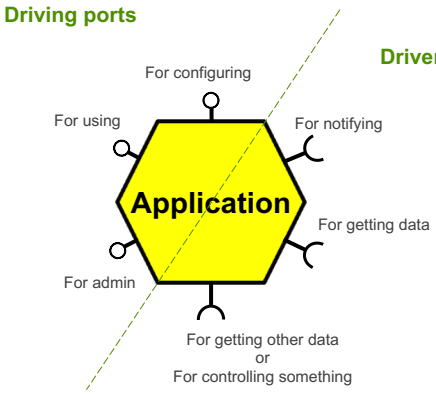




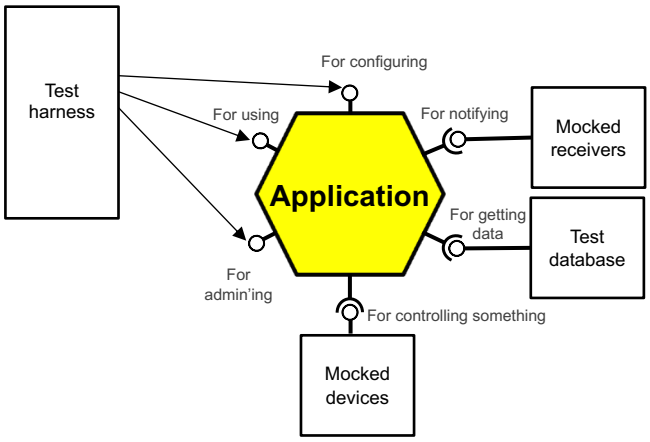
The app is a component, with ports

We name the ports for their purpose: "For_doing_something".

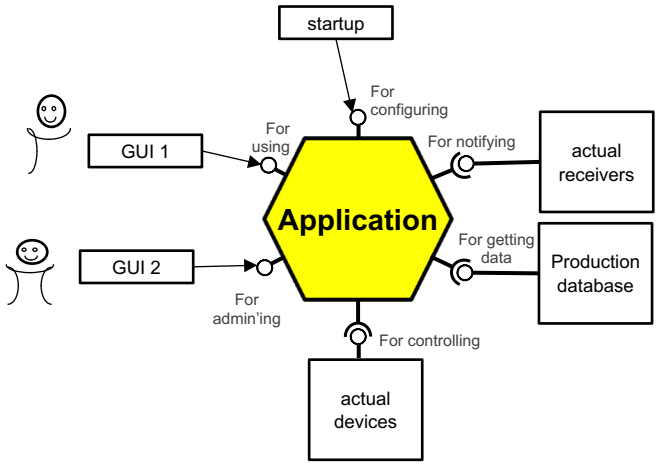
Each port can have multiple function calls.



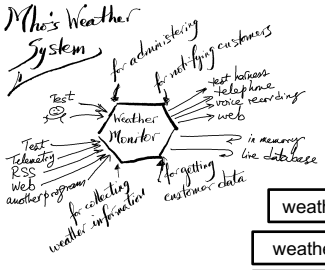
Hooking up the component for testing

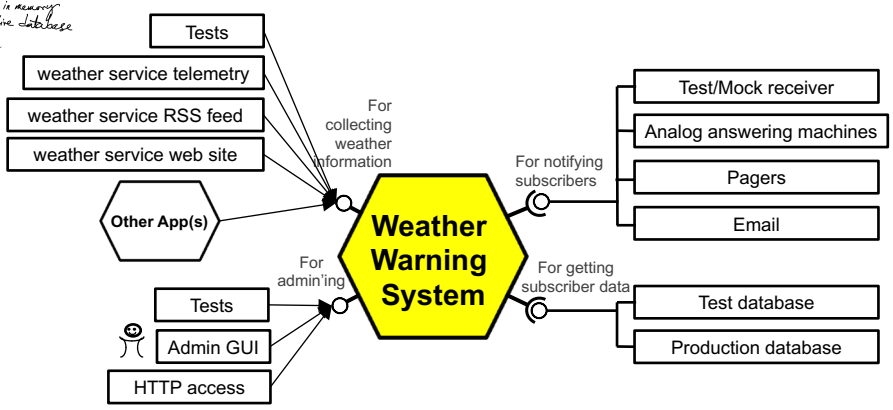


Hooking up the component for production

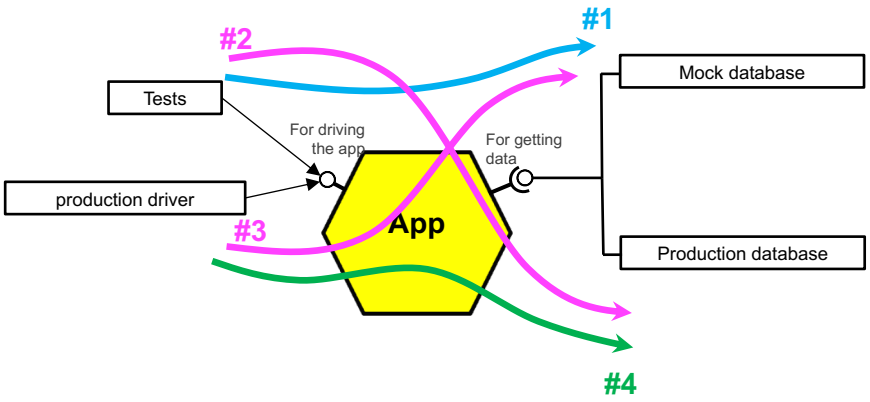


Mho's weather warning system as components

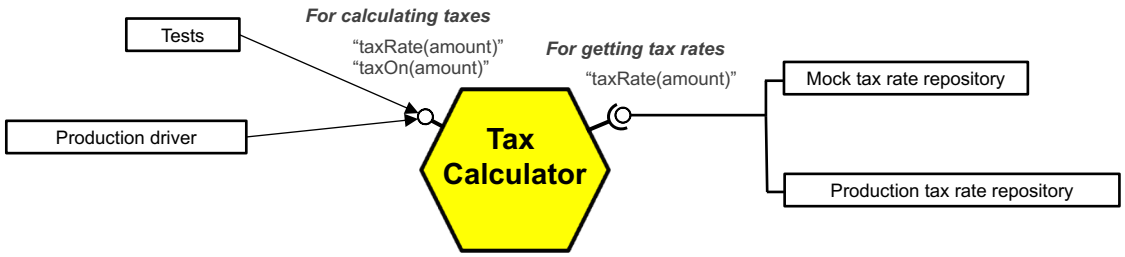




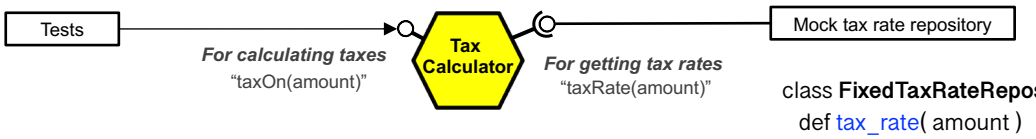
Development Sequence:
tests + mocks first



Simplest example: tax calculator



Code for Tax Calculator (Ruby)




```
class TaxCalculator
  def initialize( tax_rate_repository )
    @tax_rate_repository = tax_rate_repository
  end
  def tax_on( amount )
    amount * @tax_rate_repository.tax_rate( amount )
  end
end
```

```
class FixedTaxRateRepository
  def tax_rate( amount )
    0.15
  end
end
```

(Note: no interface declarations)

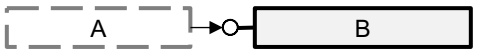
```
tax_rate_repository = FixedTaxRateRepository.new
my_calculator = TaxCalculator.new( tax_rate_repository )
puts my_calculator.tax_rate( 100 )
```

This is the work of the "configurator" (aka Composition Root)

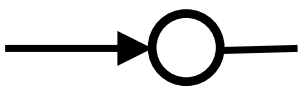
This is using the system at the port

Detour for type-checked languages: *Provided* & *Required* interfaces

B provides services.
A uses them.

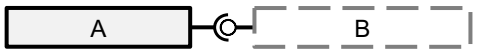


B "implements" the interface.

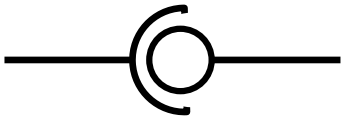


Useful for defining a public API

A "requires" this interface.
B implements it.



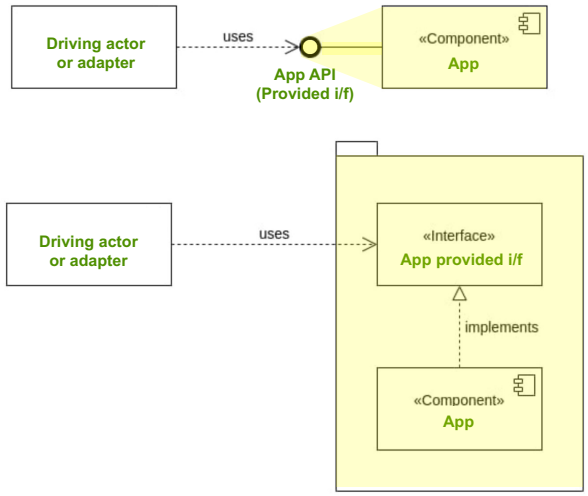
A owns the interface definition.



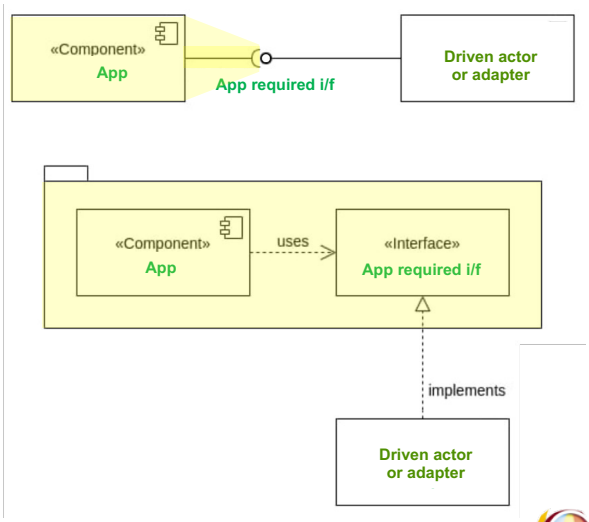
Useful for decoupling a component from its receivers

The source-code dependencies

Driving ports: The app owns the interface



Driven ports: The app owns the interface



What a required interface looks like in code (Java)

For getting tax rates
"taxRate(amount)"

taxCalculator — FixedTaxRateRepository

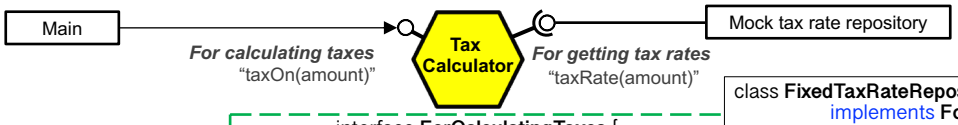
```
interface ForGettingTaxRates {
    double taxRate(double amount); }
```

Make sure the required interface's definition belongs to the calculator, not to the repository!

```
class TaxCalculator {
    private ForGettingTaxRates taxRateRepository;
    public TaxCalculator(ForGettingTaxRates taxRateRepository) {
        this.taxRateRepository = taxRateRepository;
    }
    public double taxOn(double amount) {
        return amount * taxRateRepository. taxRate( amount );
    }
}
```

```
class FixedTaxRateRepository
        implements ForGettingTaxRates {
    public double taxRate(double amount) {
        return 0.15; }
}
```


Code for Tax Calculator (Java)




```
interface ForCalculatingTaxes {
    double taxOn(double amount);
}
interface ForGettingTaxRates {
    double taxRate(double amount);
}

class TaxCalculator implements ForCalculatingTaxes {
    private ForGettingTaxRates taxRateRepository;

    public TaxCalculator(ForGettingTaxRates taxRateRepository) {
        this.taxRateRepository = taxRateRepository;
    }
    public double taxOn(double amount) {
        return amount *  taxRateRepository. taxRate( amount );
    }
}
```

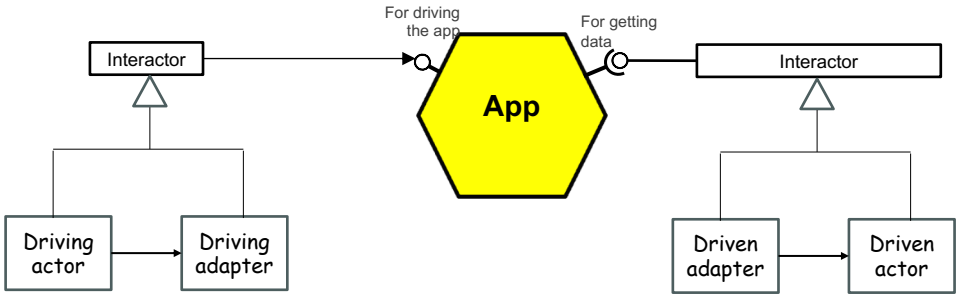
```
class FixedTaxRateRepository
        implements ForGettingTaxRates {
    public double taxRate(double amount) {
        return 0.15;
    }
}
```

```
class Main {
    public static void main(String[] args) {
        ForGettingTaxRates taxRateRepository = new FixedTaxRateRepository();
        ForCalculatingTaxes myCalculator = new TaxCalculator( taxRateRepository );
        System.out.println( myCalculator.taxOn( 100 ) );
    }
```

the "configurator"

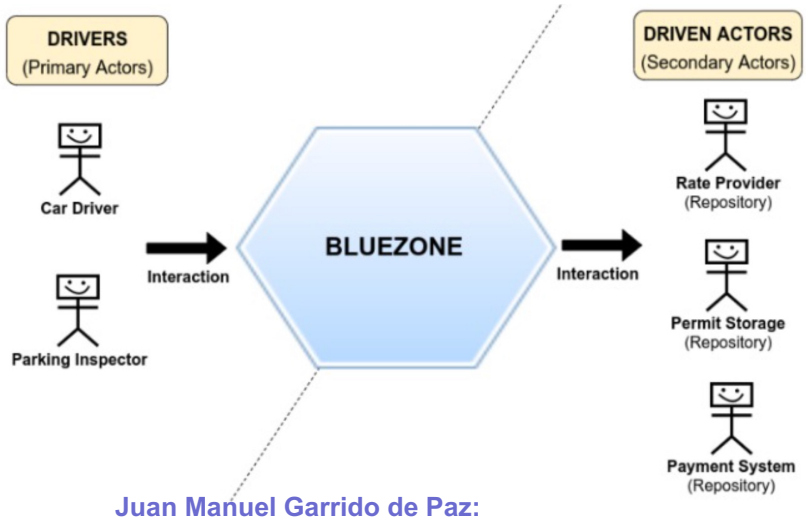
using the system at the port

what is an interactor?
(UML / C++ / Java)



A more complex example: Juan's "Blue Zone"

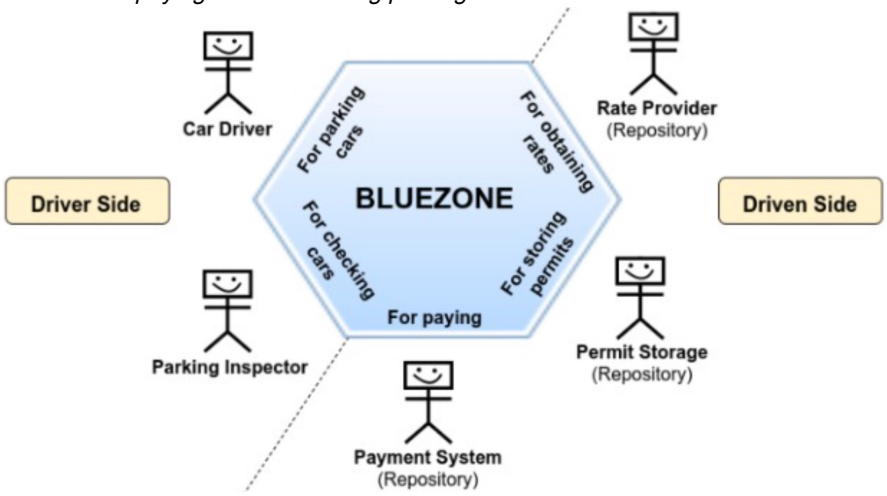
BlueZone allows car drivers to pay remotely for parking cars at zones in a city, instead of paying with coins using parking meters



Juan Manuel Garrido de Paz:
https://github.com/jmgarridopaz/bluezone

Juan's "Blue Zone" example

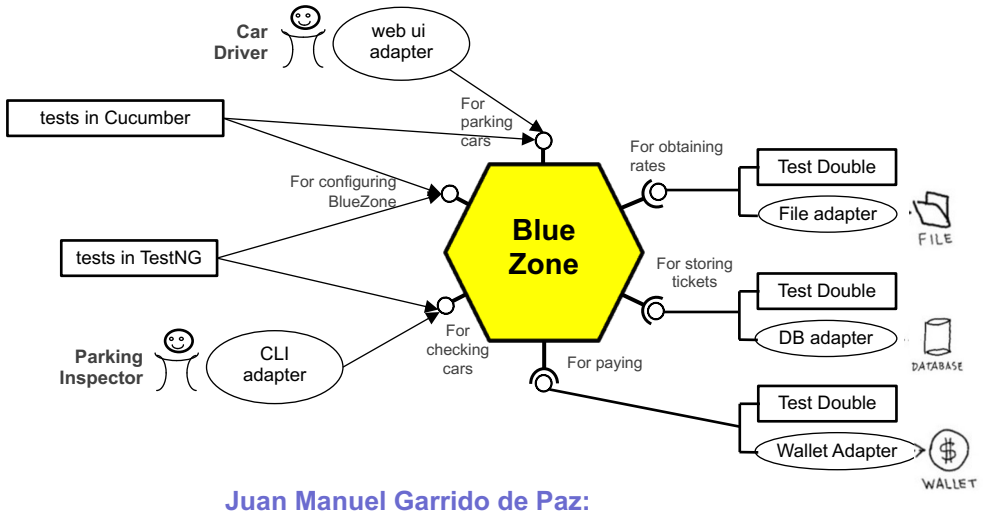
BlueZone allows car drivers to pay remotely for parking cars at zones in a city, instead of paying with coins using parking meters



Juan Manuel Garrido de Paz:
https://github.com/jmgarridopaz/bluezone

Juan's "Blue Zone" example

BlueZone allows car drivers to pay remotely for parking cars at zones in a city, instead of paying with coins using parking meters



Juan Manuel Garrido de Paz:
https://github.com/jmgarridopaz/bluezone

Juan's "Blue Zone" example

https://github.com/jmgarridopaz/bluezone

Ports

- Driven Ports
  - ForPaying.java
  - ForObtainingRates.java
  - ForStoringTickets.java
- Driving Ports
  - ForCheckingCars.java
  - ForConfiguringApp.java
  - ForParkingCars.java

```
/**
 * DRIVEN PORT
 */
public interface ForObtainingRates {
    public Set<Rate> findAll();
    public Rate findByName (String rateName );
    public void addRate ( Rate rate );
    public boolean exists ( String rateName );
    public void empty();
}
```

Adapters

- Driven Adapters
  - bluezone-adapter-forpaying-spy
  - bluezone-adapter-forobtainingrates-stub
  - bluezone-adapter-forstoringtickets-fake
- Driving Adapters
  - bluezone-driver-forcheckingcars-test
  - bluezone-adapter-forparkingcars-webui
  - bluezone-driver-forparkingcars-test

```
/**
 * Driven adapter that implements "forobtainingrates" port
   with a stub test double.
 */
@Adapter(name="test-double")
public class StubRateProviderAdapter implements ForObtainingRates {
        private Set<Rate> rates;
```


The folders: Port declarations and Adapters

- Hexagonal Project Structure
  - Driven Adapters
  - Driving Adapters
  - Tax Calculator App
    - Driven Ports
    - Driving Ports
    - TaxCalculator

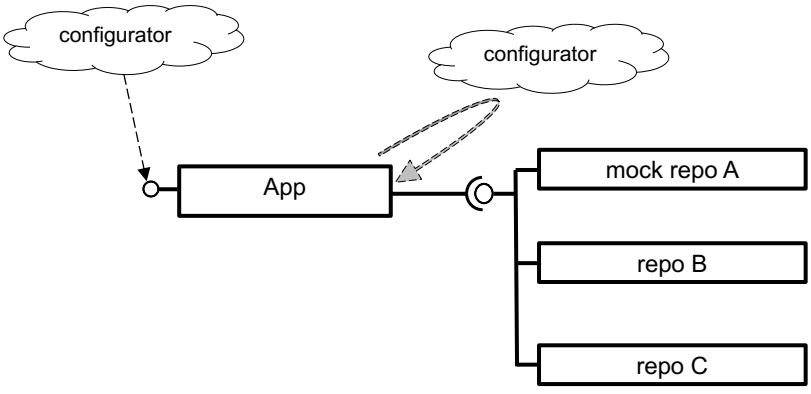
Port declaration folders in the app

- Driven Ports
  - ForPaying.java
  - ForObtainingRates.java
  - ForStoringTickets.java
- Driving Ports
  - ForCheckingCars.java
  - ForConfiguringApp.java
  - ForParkingCars.java

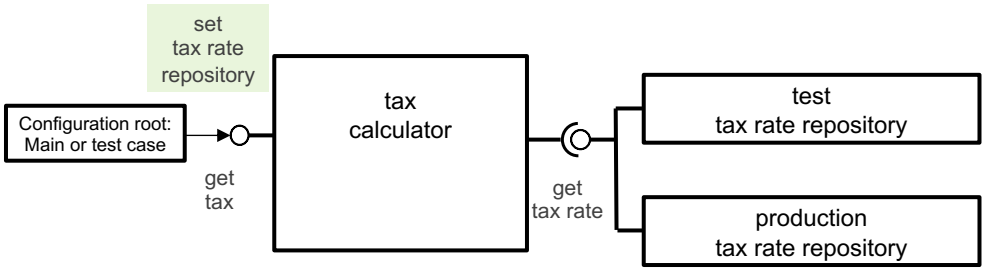
Adapter folders outside the app

- bluezone-adapter-forpaying-spy
- bluezone-adapter-forobtainingrates-stub
- bluezone-adapter-forstoringtickets-fake
- bluezone-driver-forcheckingcars-test
- bluezone-adapter-forparkingcars-webui
- bluezone-driver-forparkingcars-test

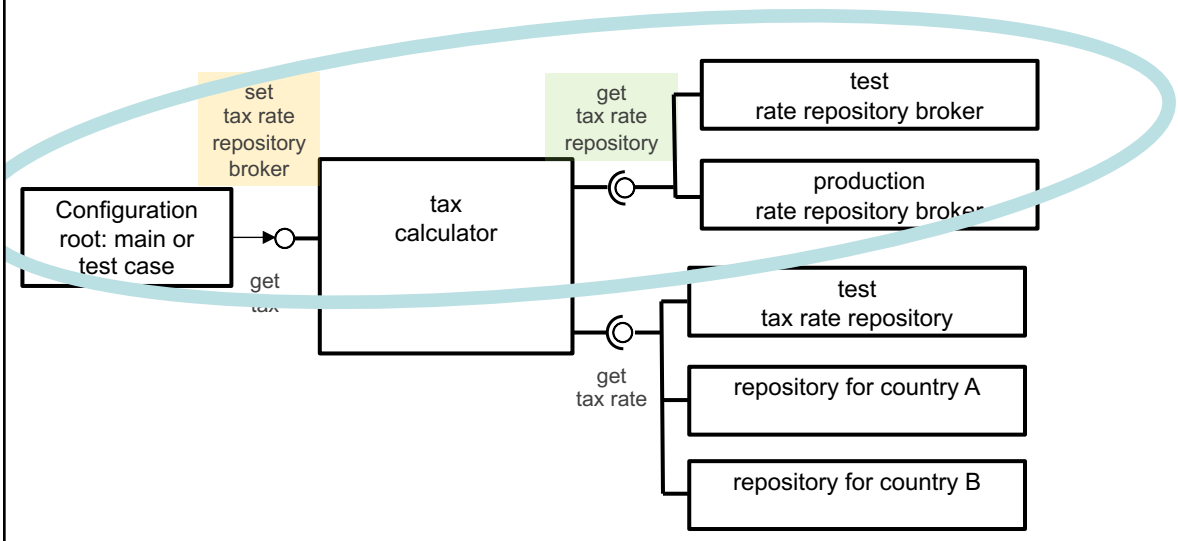
How do we design the configurator?



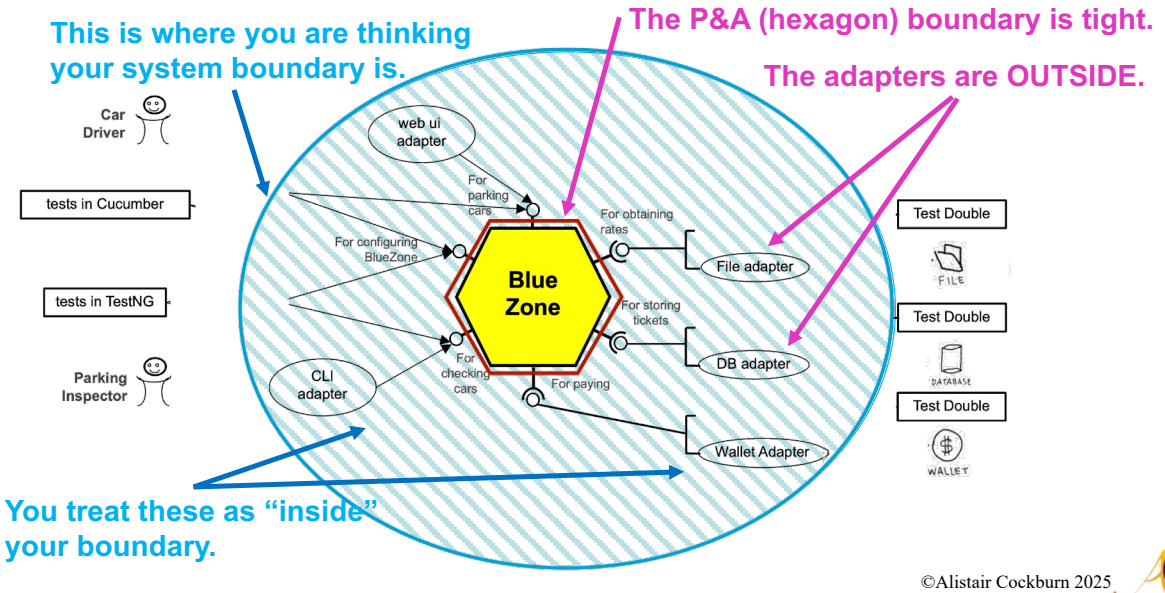
Configurator design #1: setter method
(Dependency Injection)



Configurator design #2: repository broker
(Dependency Lookup)

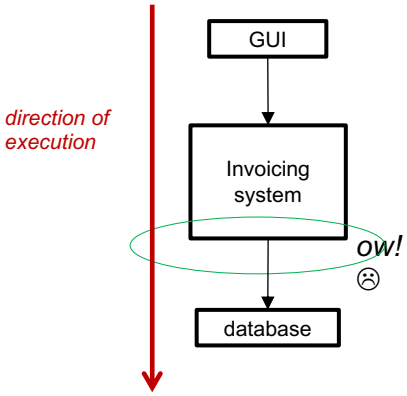


Why is it called "Ports & Adapters"?



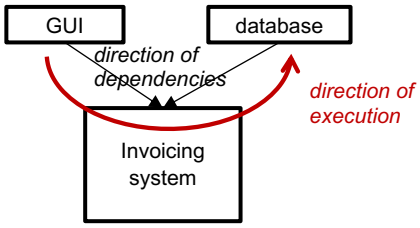
FAQ: Ports & Adapters versus "layers" [1/3]

Standard 3-layer dependencies give difficulties



Keep the app independent from everything else

Flow goes against external dependency

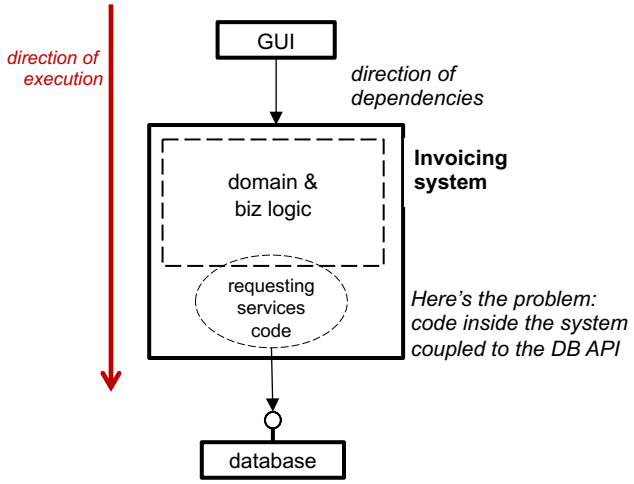


(Warning: this drawing is pretty & simple but still not quite correct: the database is in the wrong place... Stay tuned...)

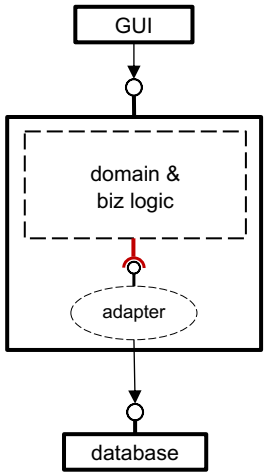
FAQ: Ports & Adapters versus "layers" [2/3]

Identify & separate the adapter code

Standard 3-layer dependencies with difficulties



Create an adapter



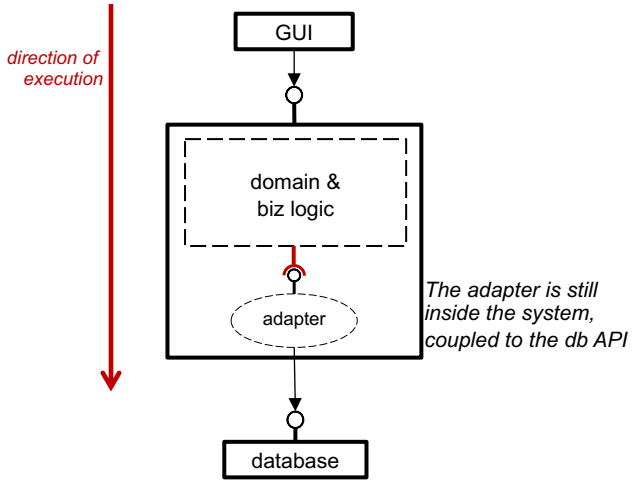
Step 1: Have the domain code request services using domain concepts only.

Create a separate "adapter". (... but the adapter is still dependent on the DB API)

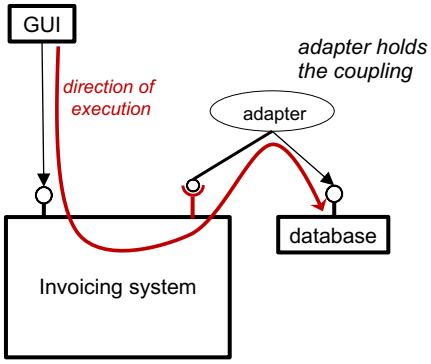
FAQ: Ports & Adapters versus "layers" [3/3]

Adapter goes outside, to protect the dependencies

Standard 3-layer dependencies with difficulties



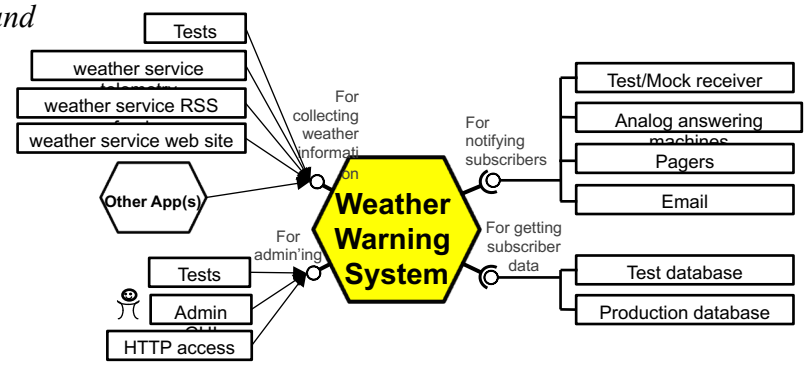
Ports & Adapters with all dependencies downward



This is the correct drawing

Ports & Adapters Pattern
(aka Hexagonal Architecture)

Create your application to work without either a UI or a database so you can run automated regression-tests against the application, protect your code from leakage between business and I/O, work when the database becomes unavailable, upgrade to new technology, and link applications together..

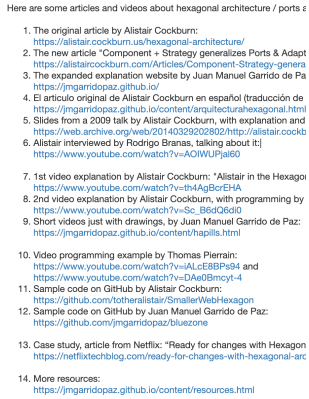


With many thanks to Juan Manuel Garrido de Paz for continual close reading, exactness, and code.

Additional Resources (1)

Juan Manuel Garrido de Paz:
https://jmgarridopaz.github.io/

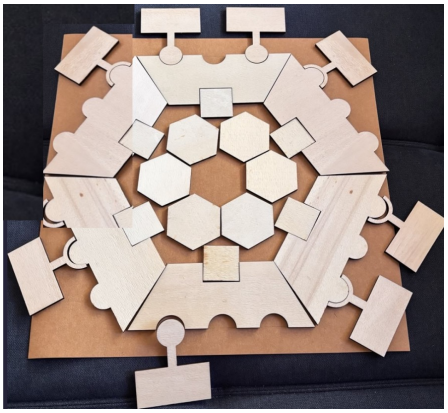
https://hexagonalarchitecture.org/



Here are some articles and videos about hexagonal architecture / ports a

1. The original article by Alistair Cockburn:
   https://alistair.cockburn.us/hexagonal-architecture/
2. The new article "Component + Strategy generalizes Ports & Adapt
   https://alistaircockburn.com/Articles/Component-Strategy-genera
3. The expanded explanation website by Juan Manuel Garrido de Pa
   https://jmgarridopaz.github.io/
4. El articulo original de Alistair Cockburn en español (traducción de
   https://jmgarridopaz.github.io/content/arquitecturahexagonal.html
5. Slides from a 2009 talk by Alistair Cockburn, with explanation and
   https://web.archive.org/web/20140329202802/http://alistair.cockb
6. Alistair interviewed by Rodrigo Branas, talking about it:
   https://www.youtube.com/watch?v=AOIWUPjal60
7. 1st video explanation by Alistair Cockburn: "Alistair in the Hexago
   https://www.youtube.com/watch?v=th4AgBcrEHA
8. 2nd video explanation by Alistair Cockburn, with programming by
   https://www.youtube.com/watch?v=Sc_B6dQ6di0
9. Short videos just with drawings, by Juan Manuel Garrido de Paz:
   https://jmgarridopaz.github.io/content/hapills.html
10. Video programming example by Thomas Pierrain:
    https://www.youtube.com/watch?v=iALcE8BPs94 and
    https://www.youtube.com/watch?v=DAe0Bmcyt-4
11. Sample code on GitHub by Alistair Cockburn:
    https://github.com/totheralistair/SmallerWebHexagon
12. Sample code on GitHub by Juan Manuel Garrido de Paz:
    https://github.com/jmgarridopaz/bluezone
13. Case study, article from Netflix: "Ready for changes with Hexagon
    https://netflixtechblog.com/ready-for-changes-with-hexagonal-arc
14. More resources:
    https://jmgarridopaz.github.io/content/resources.html

Tangible hexagonal: Tom & Kathi Asel
https://tangible-concepts.de



Additional Resources (2)

Paper Book:
Amazon or elsewhere

Hexagonal Architecture Explained

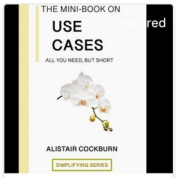
How the Ports & Adapters architecture simplifies your life, and how to implement it

Updated 1st edition

Alistair Cockburn
Juan Manuel Garrido de Paz

ePub version & even a Mug:
https://alistaircockburn.company.site

Epub: Hexagonal Architecture Explained Updated 1st ed.



Epub: Mini-Book on Use Cases
The 20% of the technique that gets you 80% of the

Epub: Unifying User Stories, Use Cases, Story Maps, Preview ed.

Mug: Hexagonal Architecture
Store / Mugs&Ts
$12.00
Hexagonal Architecture for your daily coffee! Drink from it while you read the book!
Quantity: 1
Add to Bag

Hexagonal Architecture
(Ports & Adapters)

Alistair Cockburn

Slides:
https://alistaircockburn.com/Articles/Talk-Hexagonal-Nov-25

Your challenge:
Program a simple Ports&Adapters app during this talk!

Choose your own app, or
Bios of German poets

Please

- *ATDD or at least have & show the tests*
- *OK just 1 user service (small is ok, the structure is key)*
- *At least 1 driven actor (secondary actor in use case terms)*
- *Easy to read – the structure is what we are after*
- *Folder structure is important (for readability)*

and show us the code at the end of this talk