S.O.L.I.D. Python

aleasoluciones

Alea Soluciones
Bifer Team

@eferro
@pasku1
@apa42
@nestorsalceda

Usual OO Systems

Rigid

Fragile

Immobile

Viscous

Why S.O.L.I.D. principles?

To create easy to **maintain** OO systems

Improve **reusability**

Easy **testing**

For creating

Clean Code

It's all about money

S.O.L.I.D

SRP - Single responsibility principle

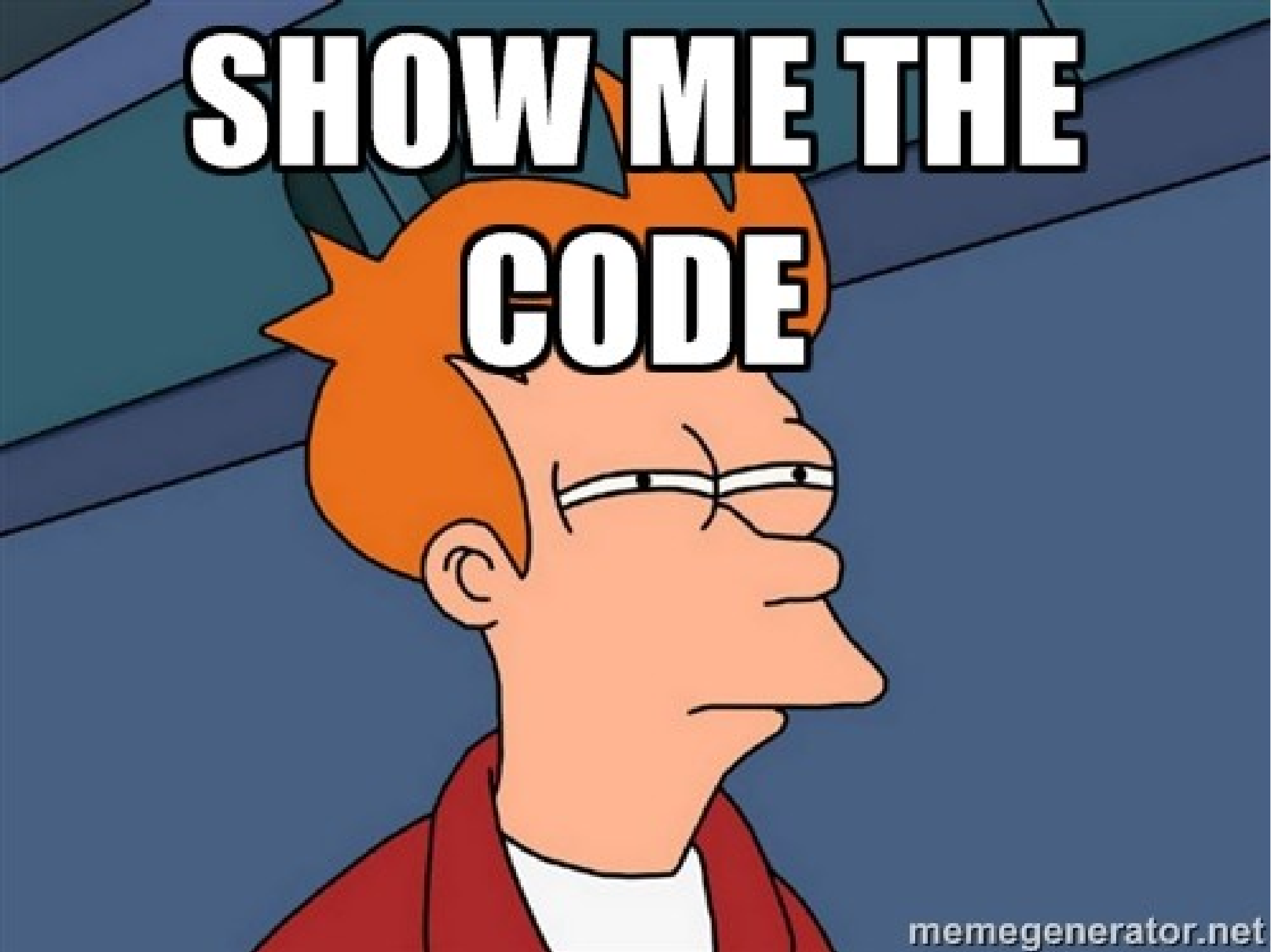
DIP - Dependency inversion principle

OCP - Open/closed principle

LSP - Liskov substitution principle

ISP - Interface segregation principle

Principles
Not Rules


SHOW ME THE
CODE
memegenerator.net

```
Car wash service
  car wash job
    when the car enters in the car wash
      ✓ it registers a job
  customer notification
    when service completed
      ✓ it notifies the customer
  reporting
    when client report requested
      ✓ it shows all wash services for that customer

3 examples ran in 0.0178 seconds
```

```
class CarWashService(object):

    def __init__(self, sms_sender):
        self.persistence = {}
        self.sms_sender = sms_sender

    def require_car_wash(self, car, customer):
        service_id = uuid.uuid4().hex
        self.persistence[service_id] = (car, customer)
        return service_id

    def wash_completed(self, service_id):
        car, customer = self.persistence[service_id]
        self.sms_sender.send(mobile_phone=customer.mobile_phone,
            text='Car %{car.plate} whased'.format(car=car))
```

Single Responsibility Principle

Just because you *can* doesn't mean you *should*.

```
class CarWashService(object):

    def __init__(self, sms_sender):
        self.persistence = {}
        self.sms_sender = sms_sender

    def require_car_wash(self, car, customer):
        service_id = uuid.uuid4().hex
        self.persistence[service_id] = (car, customer)
        return service_id

    def wash_completed(self, service_id):
        car, customer = self.persistence[service_id]
        self.sms_sender.send(mobile_phone=customer.mobile_phone,
            text='Car %{car.plate} whased'.format(car=car))
```

```python
class CarWashService(object):

    def __init__(self, sms_sender):
        self.persistence = {}
        self.sms_sender = sms_sender

    def require_car_wash(self, car, customer):
        service_id = uuid.uuid4().hex
        self.persistence[service_id] = (car, customer)
        return service_id

    def wash_completed(self, service_id):
        car, customer = self.persistence[service_id]
        self.sms_sender.send(mobile_phone=customer.mobile_phone,
            text='Car %{car.plate} whased'.format(car=car))
```

...

Some refactors / versions later

```python
class CarWashService(object):

    def __init__(self, notifier, repository):
        self.repository = repository
        self.notifier = notifier

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        self.notifier.job_completed(car_wash_job)

    def services_by_customer(self, customer):
        return self.repository.find_by_customer(customer)
```

Dependency Inversion Principle

Would you solder a lamp directly
to the electrical wiring in a wall?

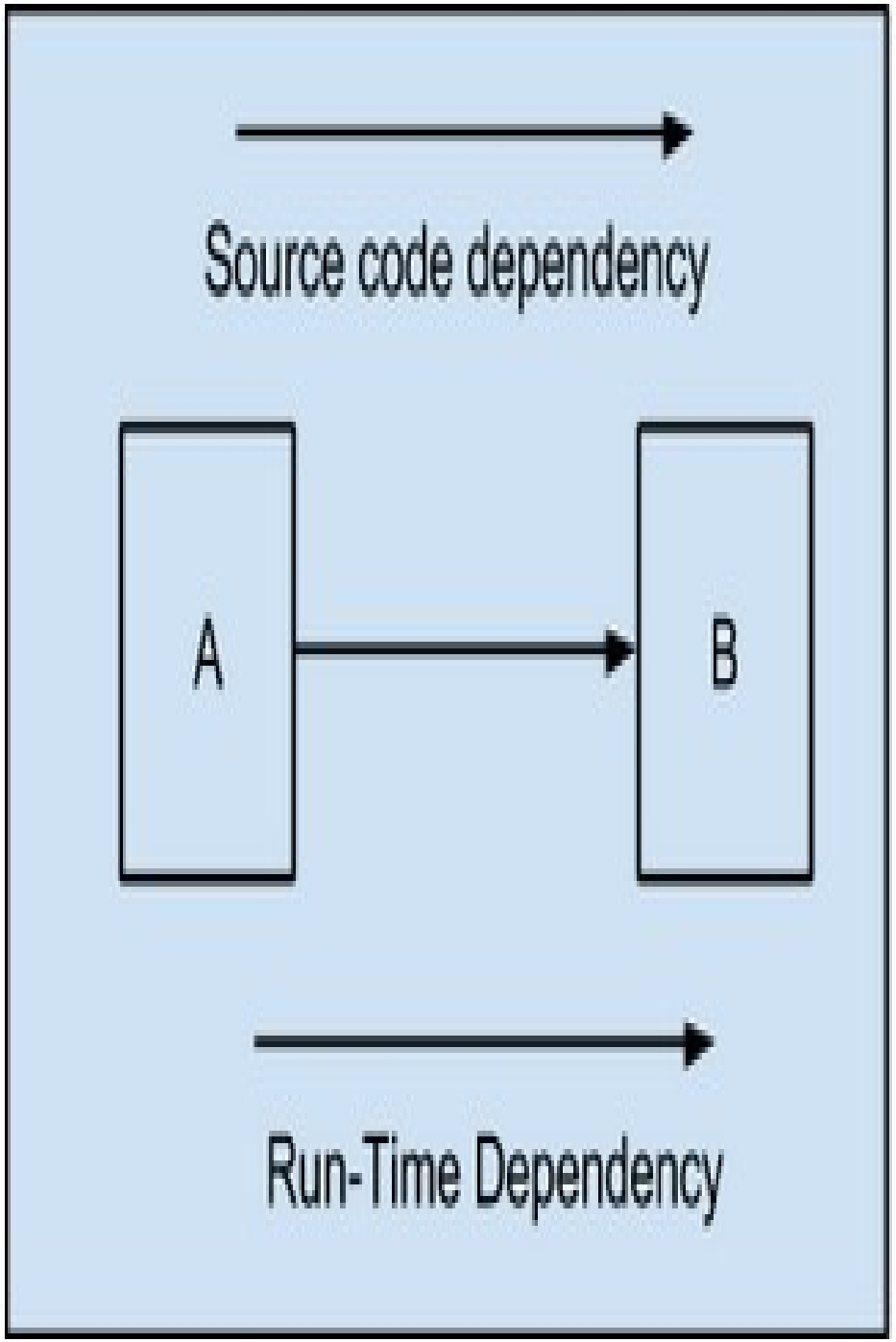
Depend upon Abstractions. Do not depend upon concretion

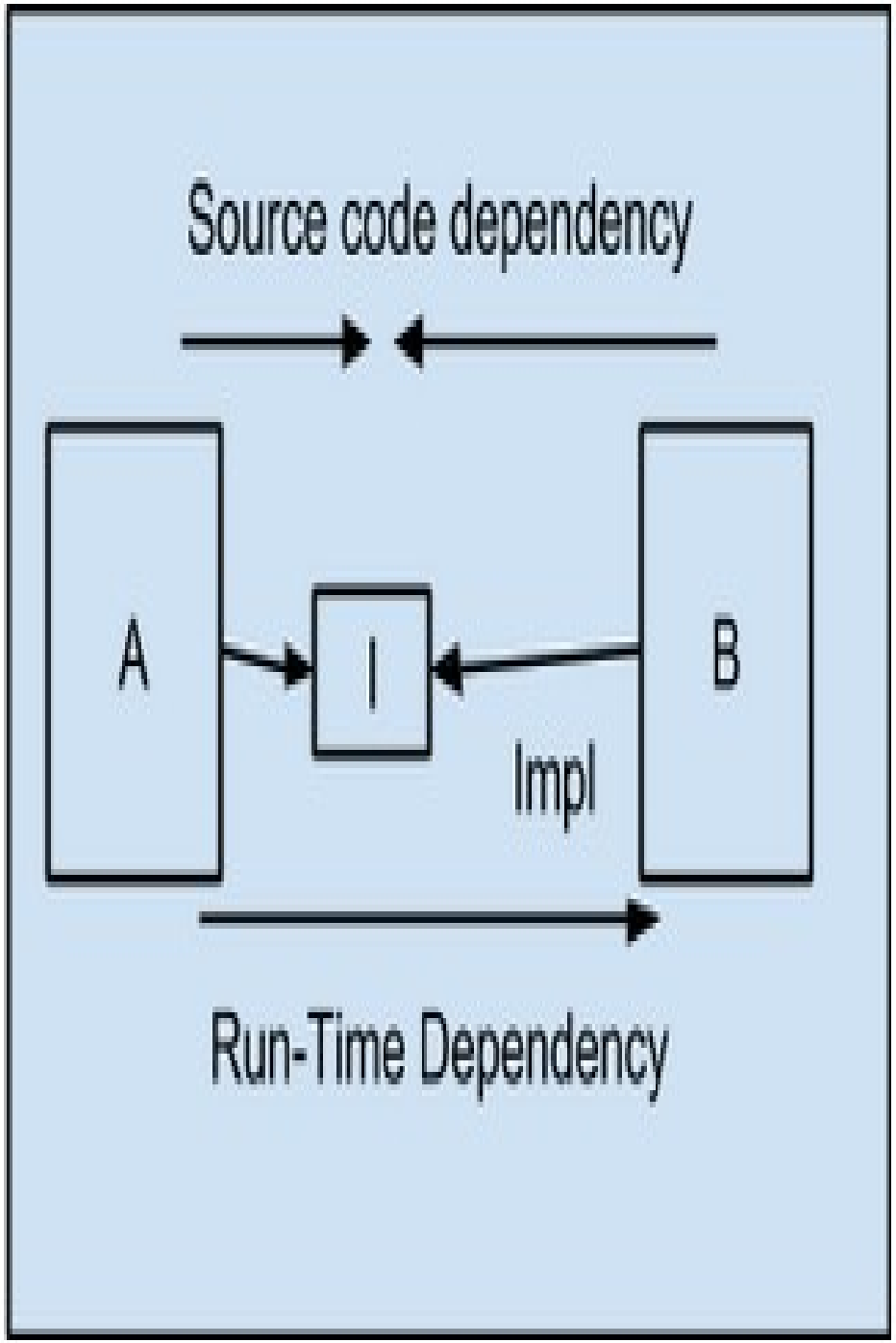
Compilation / Startup time dependency

from <package> import module

Runtime dependency

```
self.collaborator.message()
```


Source code dependency
A
B
Run-Time Dependency


Source code dependency
A
I
B
Impl
Run-Time Dependency

```python
class CarWashService(object):

    def __init__(self, repository):
        self.repository = repository

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        SmsNotifier.send_sms(car_wash_job)
```

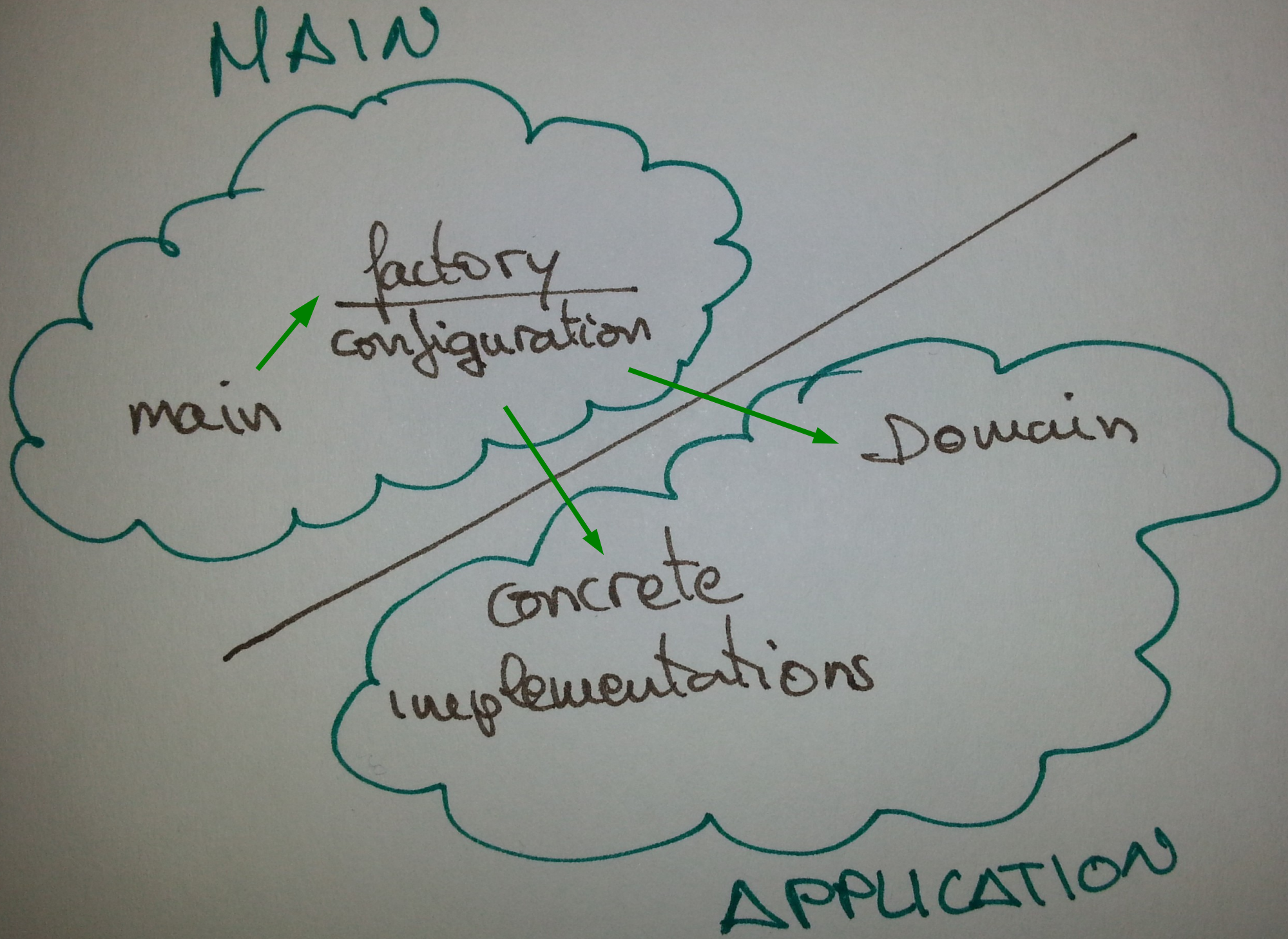
Global State Problem
Implicit dependency problem
Concrete API

```
class CarWashService(object):

    def __init__(self, repository):
        self.repository = repository
        self.notifier = SmsNotifier()

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        self.notifier.send_sms(car_wash_job)
```

No dependency injection
Implicit dependency problem

Concrete API

```python
class CarWashService(object):

    def __init__(self, notifier, repository):
        self.repository = repository
        self.notifier = notifier

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        self.notifier.job_completed(car_wash_job)
```


MAIN
main
factory
configuration
Domain
concrete
implementations
APPLICATION

```python
def in_memory_job_repository():
    return InMemoryJobRepository()

def file_job_repository():
    return FileJobRepository()

def console_log_notifier():
    return ConsoleJobNotifier()

def null_log_notifier():
    return NullJobNotifier()


def car_wash_service():
    return CarWashService(console_log_notifier(), file_job_repository())
```

```python
import factory
import car_wash


def main():
    service = factory.car_wash_service()
```

Open-Closed Principle

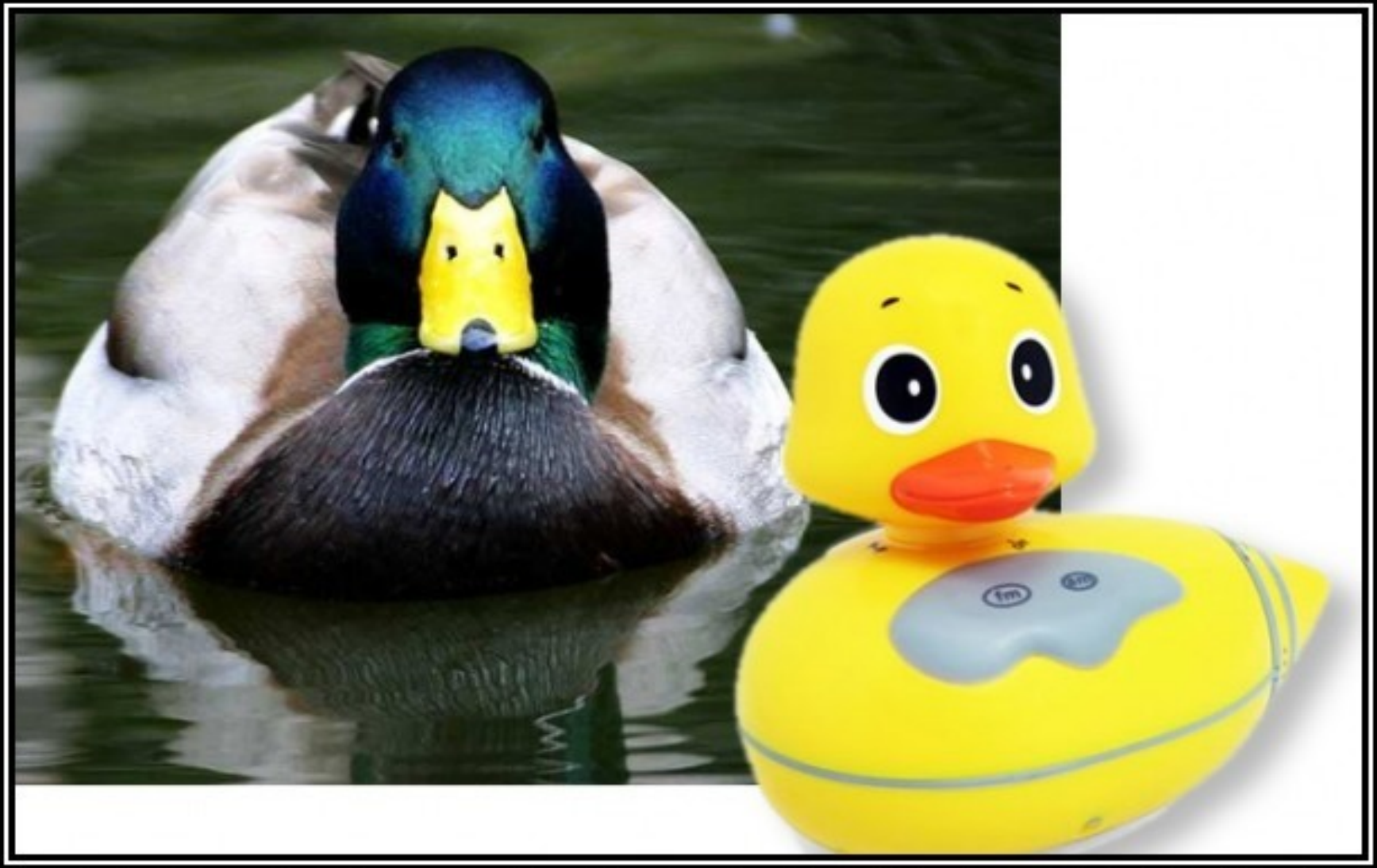
Open-chest surgery isn't needed when putting on a coat.

```python
class IJobRepository():

    def put(self, job):
        raise NotImplementedError()

    def find_by_id(self, job_id):
        raise NotImplementedError()

    def find_by_customer(self, customer):
        raise NotImplementedError()
```

```python
class InMemoryJobRepository(IJobRepository):

    def __init__(self):
        self._storage = {}

    def put(self, job):
        self._storage[job.service_id] = job

    def find_by_id(self, job_id):
        return self._storage.get(job_id)

    def find_by_customer(self, customer):
        return [job for job in self._storage.values()
         if job.has_customer(customer)]
```

```python
class InMemoryJobRepository(object):

    def __init__(self):
        self._storage = {}

    def put(self, job):
        self._storage[job.service_id] = job

    def find_by_id(self, job_id):
        return self._storage.get(job_id)

    def find_by_customer(self, customer):
        return [job for job in self._storage.values()
         if job.has_customer(customer)]
```

Duck Typing Approved!!!



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

```python
class InMemoryJobRepository(dict):

    def put(self, job):
        self[job.service_id] = job

    def find_by_id(self, job_id):
        return self.get(job_id)

    def find_by_customer(self, customer):
        return [job for job in self.values()
            if job.has_customer(customer)]
```

Liskov Substitution principle violation

Python don't force type inheritance
For API implementation
(So, for reuse code, prefer Composition)

Derived types must be completely substitutable for their base types

Interface Segregation Principle

You want me to plug this in *where?*

It isn't so important

A narrow interface is a better interface

SOLID Motivational Posters, by Derick Bailey

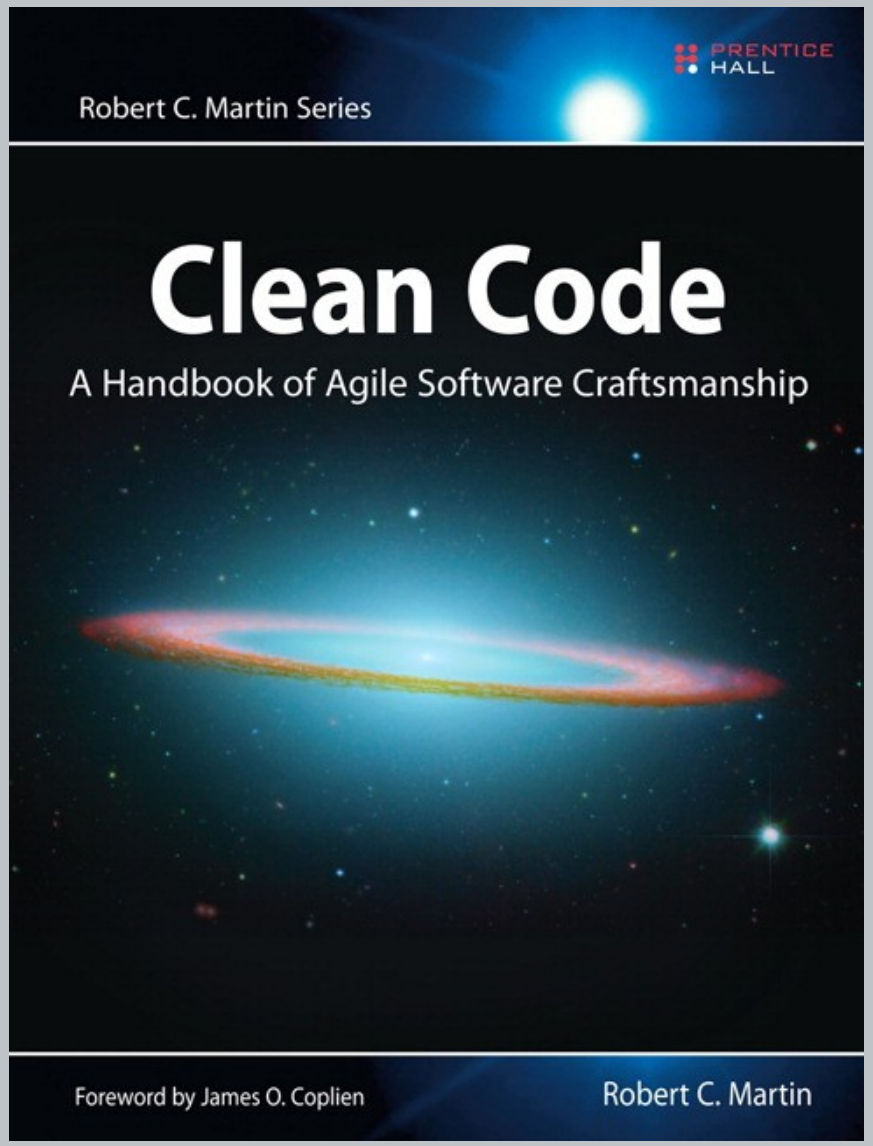
http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/

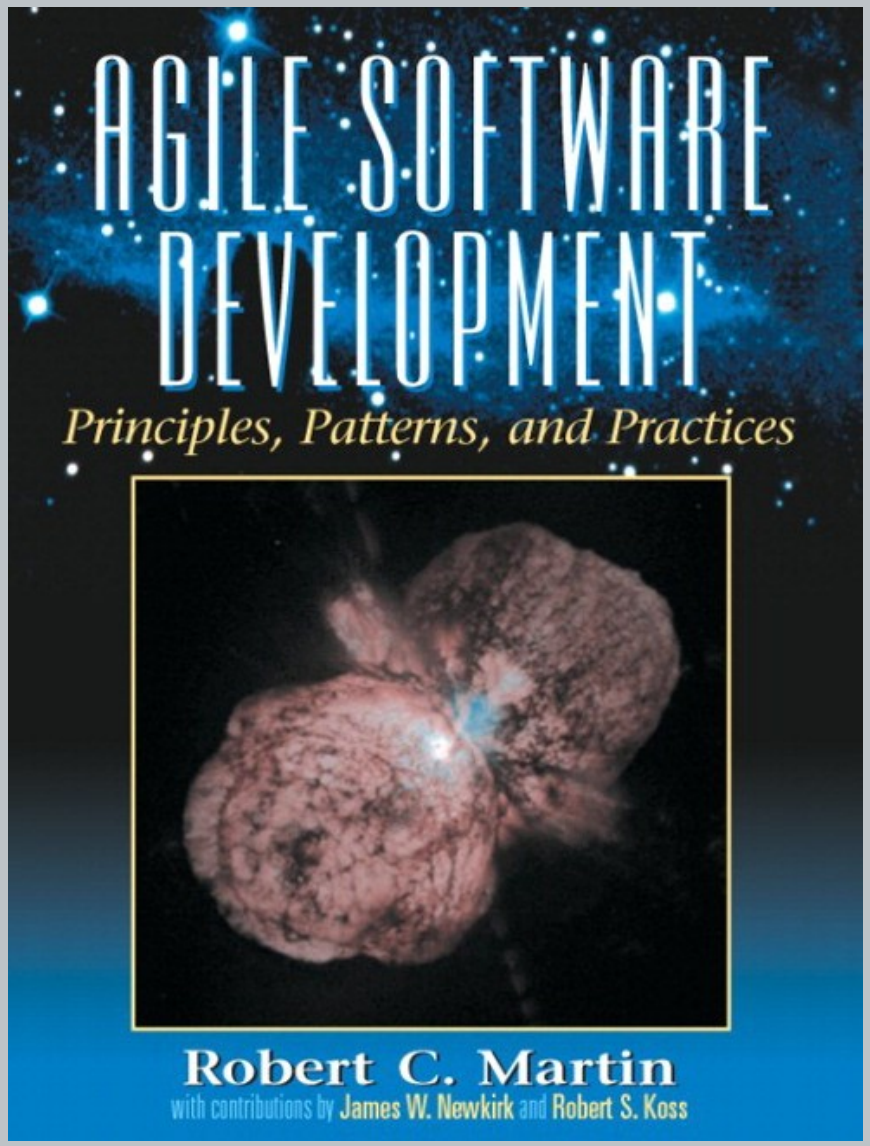
car_wash code example
https://github.com/aleasoluciones/car_wash

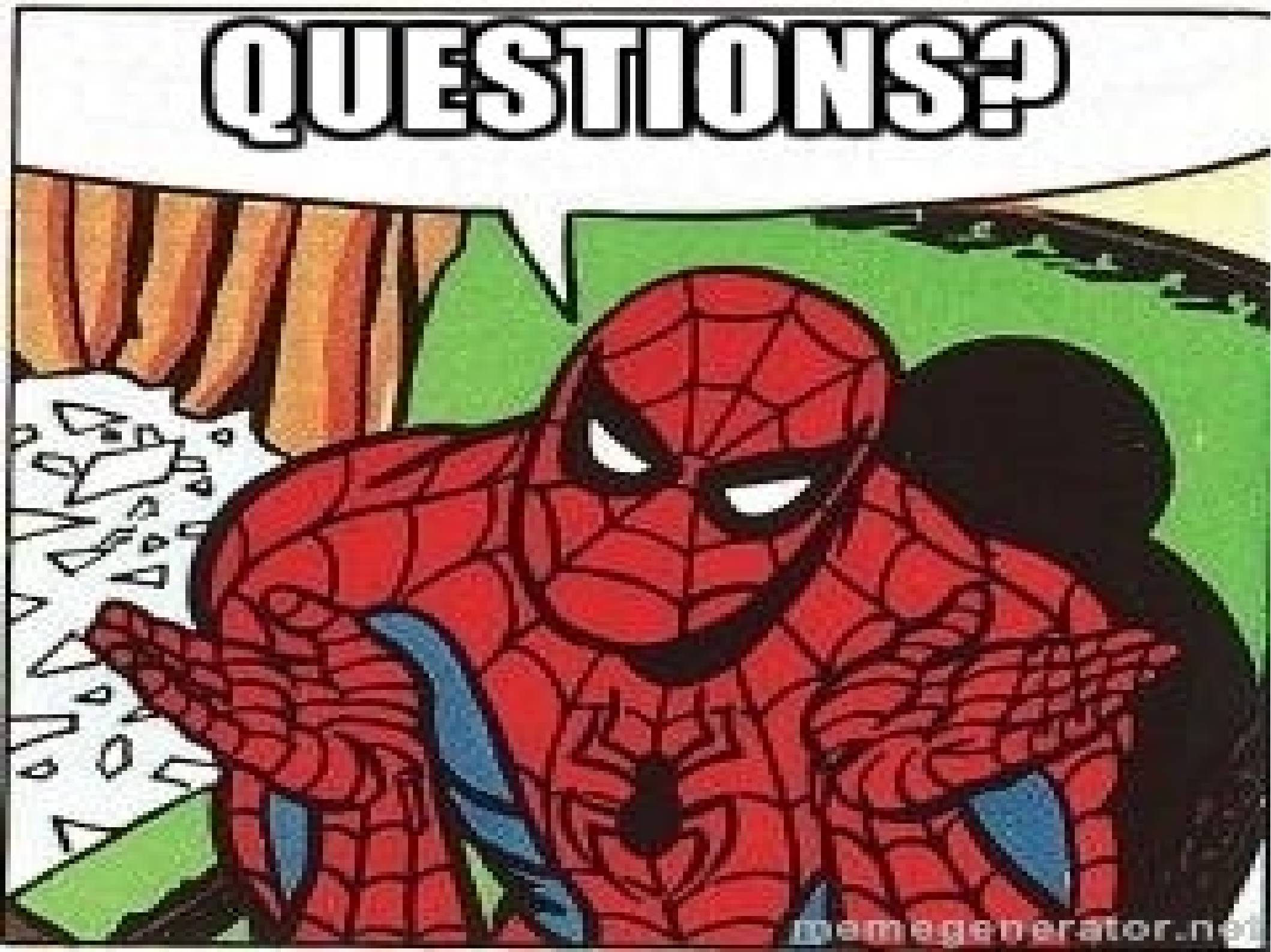
SOLID definition (at wikipedia)
http://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

Getting a SOLID start (Uncle Bob)
http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Video SOLID Object Oriented Design (Sandi Metz)
http://www.confreaks.com/videos/240-goruco2009-solid-object-oriented-design


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin


AGILE SOFTWARE DEVELOPMENT
Principles, Patterns, and Practices
Robert C. Martin
with contributions by James W. Newkirk and Robert S. Koss


QUESTIONS?
memegenerator.net

Thanks !!!

@eferro

@pasku1

@apa42

@nestorsalceda

aleasoluciones

S.O.L.I.D. Python

aleasoluciones

Alea Soluciones
Bifer Team

Hacemos producto
Telecomunicaciones
Sistemas + Software
Extreme Programming
Aportamos valor

@eferro
@pasku1
@apa42
@nestorsalceda

Usual OO Systems

Rigid

Fragile

Immobile

Viscous

Why S.O.L.I.D. principles?

To create easy to **maintain** OO systems

Improve **reusability**

Easy **testing**

For creating

Clean Code

It's all about money

S.O.L.I.D

SRP - Single responsibility principle

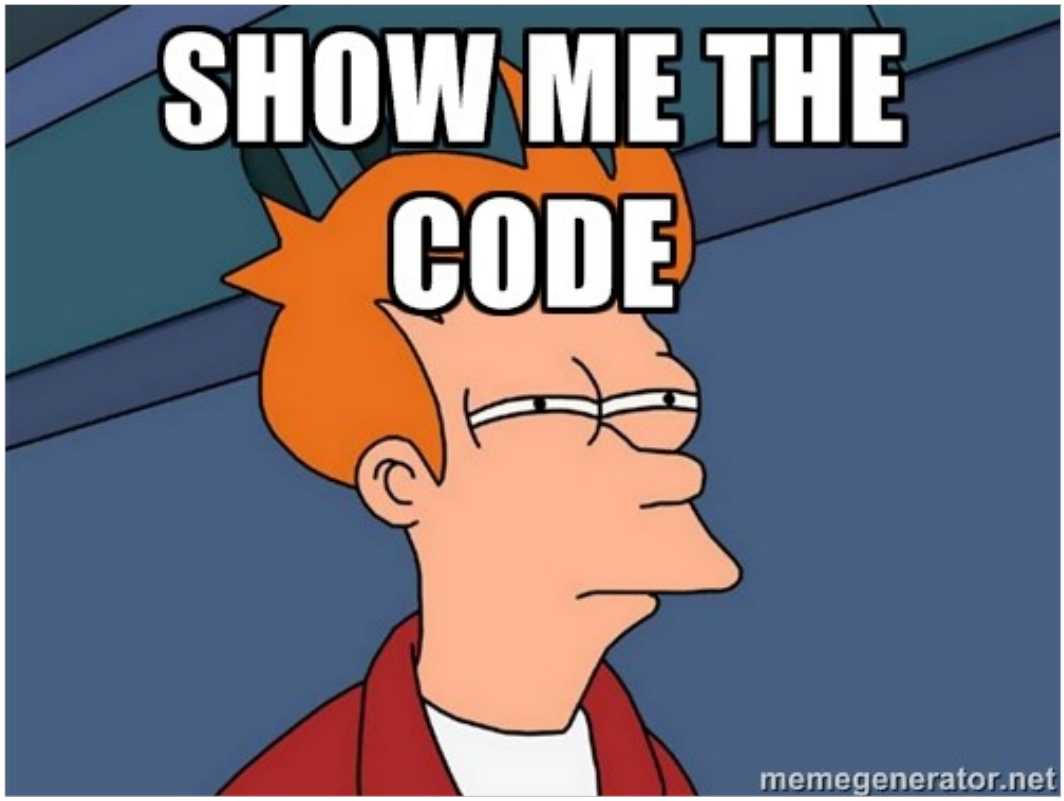
DIP - Dependency inversion principle

OCP - Open/closed principle

LSP - Liskov substitution principle

ISP - Interface segregation principle

Principles
Not Rules


SHOW ME THE
CODE
memegenerator.net

```
Car wash service
  car wash job
    when the car enters in the car wash
      ✓ it registers a job
  customer notification
    when service completed
      ✓ it notifies the customer
  reporting
    when client report requested
      ✓ it shows all wash services for that customer

3 examples ran in 0.0178 seconds
```

```python
class CarWashService(object):

    def __init__(self, sms_sender):
        self.persistence = {}
        self.sms_sender = sms_sender

    def require_car_wash(self, car, customer):
        service_id = uuid.uuid4().hex
        self.persistence[service_id] = (car, customer)
        return service_id

    def wash_completed(self, service_id):
        car, customer = self.persistence[service_id]
        self.sms_sender.send(mobile_phone=customer.mobile_phone,
            text='Car %{car.plate} whased'.format(car=car))
```

Un módulo o una función debe tener una y solo una responsabilidad, o lo que es lo mismo, debe tener una y solo una razón para cambiar.

Más de una responsabilidad hace que el código sea dificil de leer, de testear y mantener. Es decir, hace que el código sea menos flexible, más rígido, mucho más resistente al cambio.

¿Y qué es una responsabilidad?

Se trata de la audiencia de un determinado módulo o función, actores que reclaman cambios al software. Las responsabilidades son básicamente familias de funciones que cumplen las necesidades de dichos actores.

```python
class CarWashService(object):

    def __init__(self, sms_sender):
        self.persistence = {}
        self.sms_sender = sms_sender

    def require_car_wash(self, car, customer):
        service_id = uuid.uuid4().hex
        self.persistence[service_id] = (car, customer)
        return service_id

    def wash_completed(self, service_id):
        car, customer = self.persistence[service_id]
        self.sms_sender.send(mobile_phone=customer.mobile_phone,
            text='Car %{car.plate} whased'.format(car=car))
```

```python
class CarWashService(object):

    def __init__(self, sms_sender):
        self.persistence = {}
        self.sms_sender = sms_sender

    def require_car_wash(self, car, customer):
        service_id = uuid.uuid4().hex
        self.persistence[service_id] = (car, customer)
        return service_id

    def wash_completed(self, service_id):
        car, customer = self.persistence[service_id]
        self.sms_sender.send(mobile_phone=customer.mobile_phone,
            text='Car %{car.plate} whased'.format(car=car))
```

...

Some refactors / versions later

```python
class CarWashService(object):

    def __init__(self, notifier, repository):
        self.repository = repository
        self.notifier = notifier

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        self.notifier.job_completed(car_wash_job)

    def services_by_customer(self, customer):
        return self.repository.find_by_customer(customer)
```

Una clase no debe tener más que una razón para cambiar

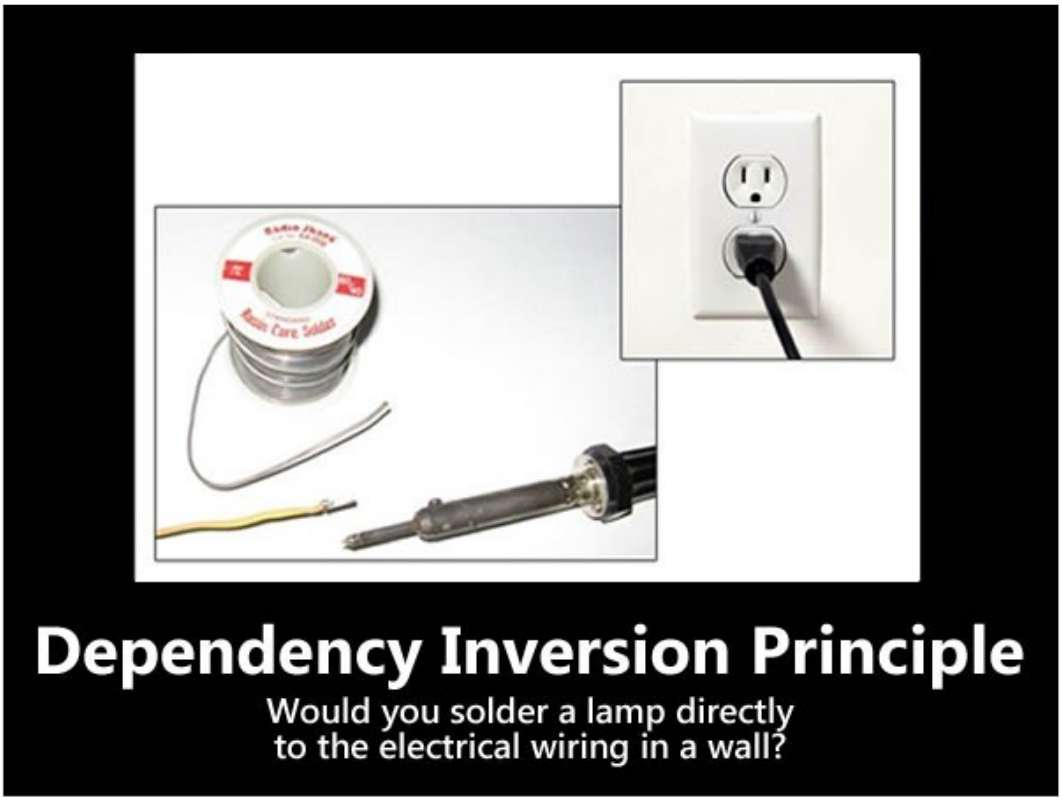
Responsabilidad / Razón para cambiar

Responsabilidad / Role en la aplicación

Reusar una clase y su contexto
Si tiene varias responsabilidades tiene un contexto muy complejo

Cohesión: Qué tan fuertemente relacionadas y enfocadas están las distintas responsabilidades de un módulo.

Acoplatmiento: El grado en el cual cada módulo de un programa depende de cada uno de los otros módulos



Este ejemplo me gustó mucho cuando lo ví!!

Ejemplo: Tiempos Viejunos: Cuando estaban CGA, VGA, SuperVGA, etc…
Ejemplo: Tiempos Viejunos: Al imprimir=> driver para una impresora específica

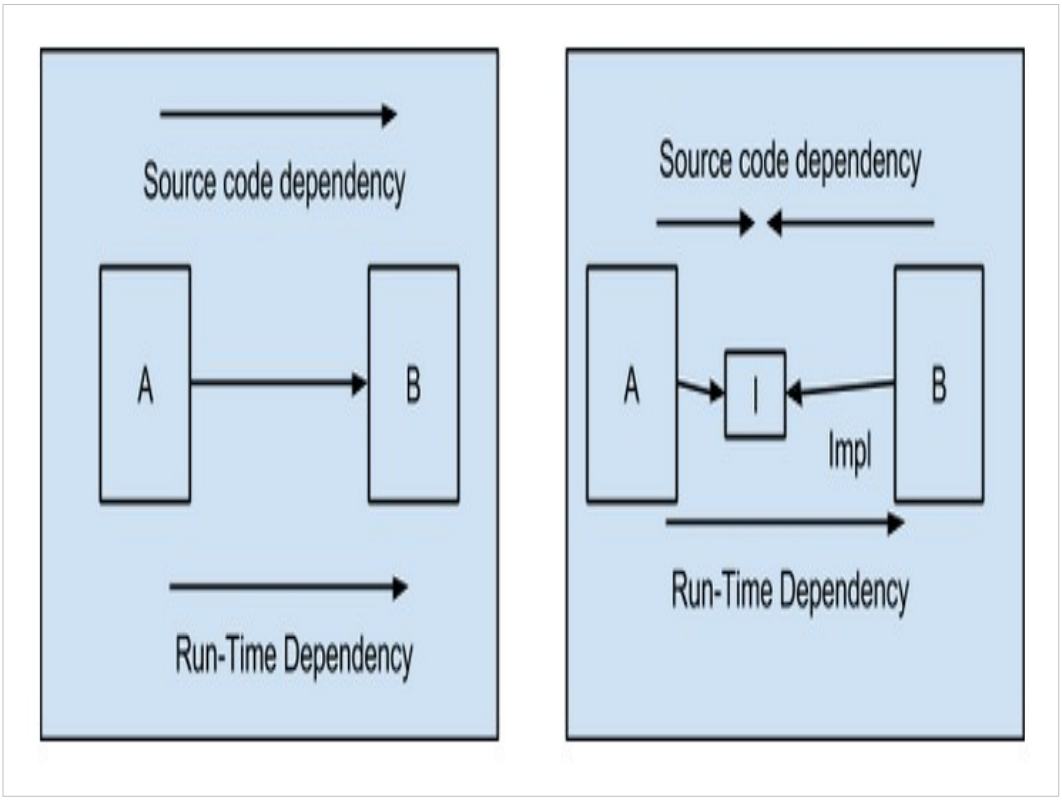
Depend upon Abstractions. Do not depend upon concretion

Compilation / Startup time dependency

from <package> import module

Runtime dependency

self.collaborator.message()


Source code dependency
A
B
Run-Time Dependency
Source code dependency
A
I
B
Impl
Run-Time Dependency

```python
class CarWashService(object):

    def __init__(self, repository):
        self.repository = repository

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        SmsNotifier.send_sms(car_wash_job)
```

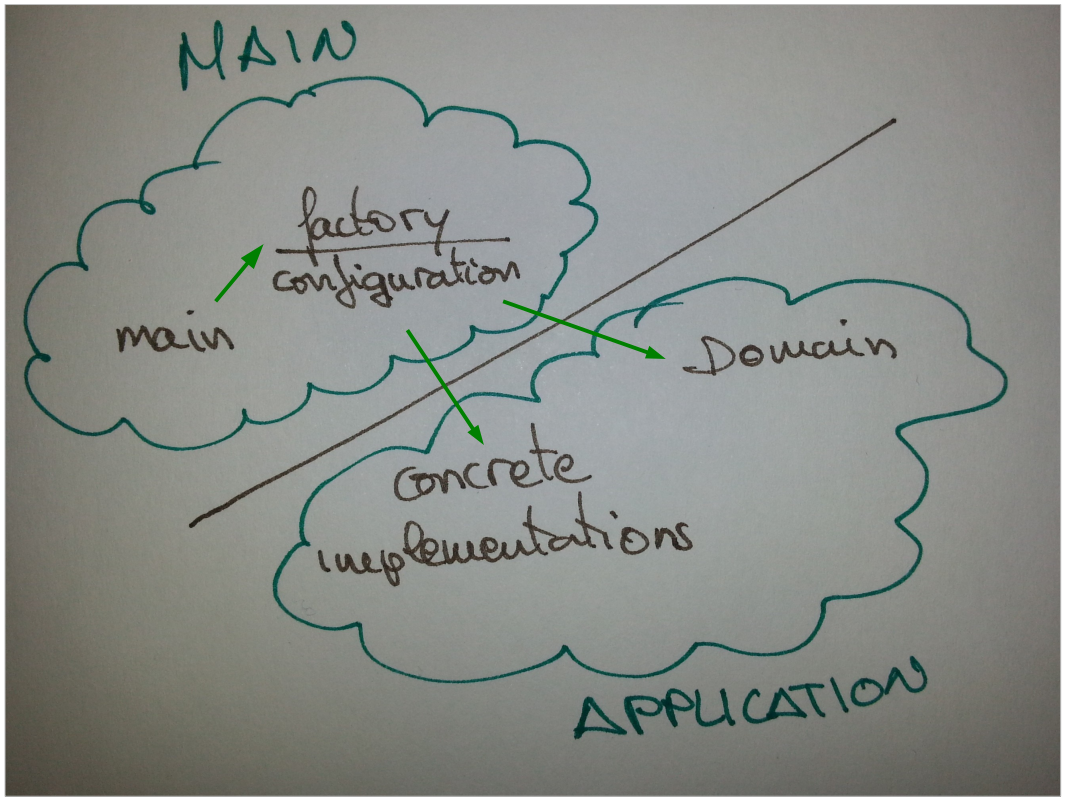
Global State Problem
Implicit dependency problem
Concrete API

```python
class CarWashService(object):

    def __init__(self, repository):
        self.repository = repository
        self.notifier = SmsNotifier()

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        self.notifier.send_sms(car_wash_job)
```

No dependency injection
Implicit dependency problem

Concrete API

```python
class CarWashService(object):

    def __init__(self, notifier, repository):
        self.repository = repository
        self.notifier = notifier

    def enter_in_the_car_wash(self, car, customer):
        job = CarWashJob(car, customer)
        self.repository.put(job)
        return job

    def wash_completed(self, service_id):
        car_wash_job = self.repository.find_by_id(service_id)
        self.notifier.job_completed(car_wash_job)
```


MAIN
factory
configuration
main
Domain
concrete
implementations
APPLICATION

```python
def in_memory_job_repository():
    return InMemoryJobRepository()

def file_job_repository():
    return FileJobRepository()

def console_log_notifier():
    return ConsoleJobNotifier()

def null_log_notifier():
    return NullJobNotifier()


def car_wash_service():
    return CarWashService(console_log_notifier(), file_job_repository())
```

```python
import factory
import car_wash


def main():
    service = factory.car_wash_service()
```

Open-Closed Principle

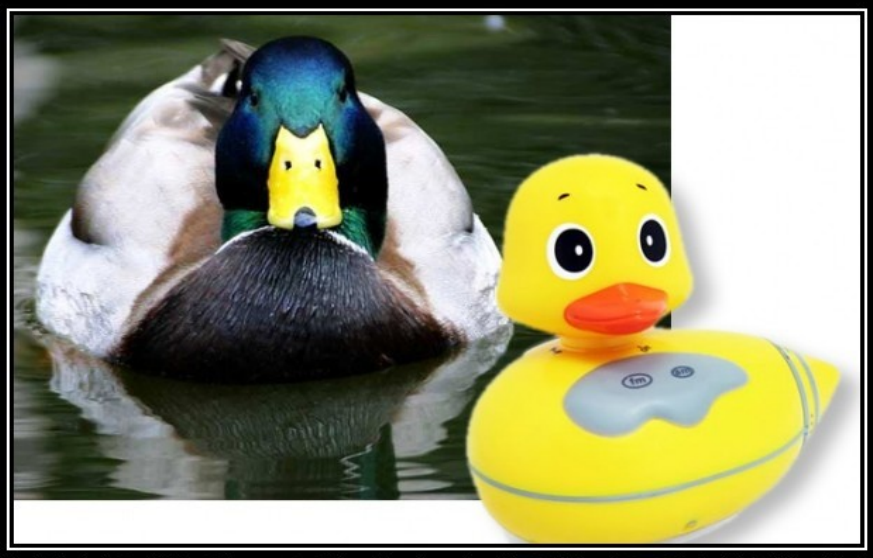
Open-chest surgery isn’t needed when putting on a coat.

```python
class IJobRepository():

    def put(self, job):
        raise NotImplementedError()

    def find_by_id(self, job_id):
        raise NotImplementedError()

    def find_by_customer(self, customer):
        raise NotImplementedError()
```

```python
class InMemoryJobRepository(IJobRepository):

    def __init__(self):
        self._storage = {}

    def put(self, job):
        self._storage[job.service_id] = job

    def find_by_id(self, job_id):
        return self._storage.get(job_id)

    def find_by_customer(self, customer):
        return [job for job in self._storage.values()
         if job.has_customer(customer)]
```

```python
class InMemoryJobRepository(object):

    def __init__(self):
        self._storage = {}

    def put(self, job):
        self._storage[job.service_id] = job

    def find_by_id(self, job_id):
        return self._storage.get(job_id)

    def find_by_customer(self, customer):
        return [job for job in self._storage.values()
         if job.has_customer(customer)]
```

Duck Typing Approved!!!

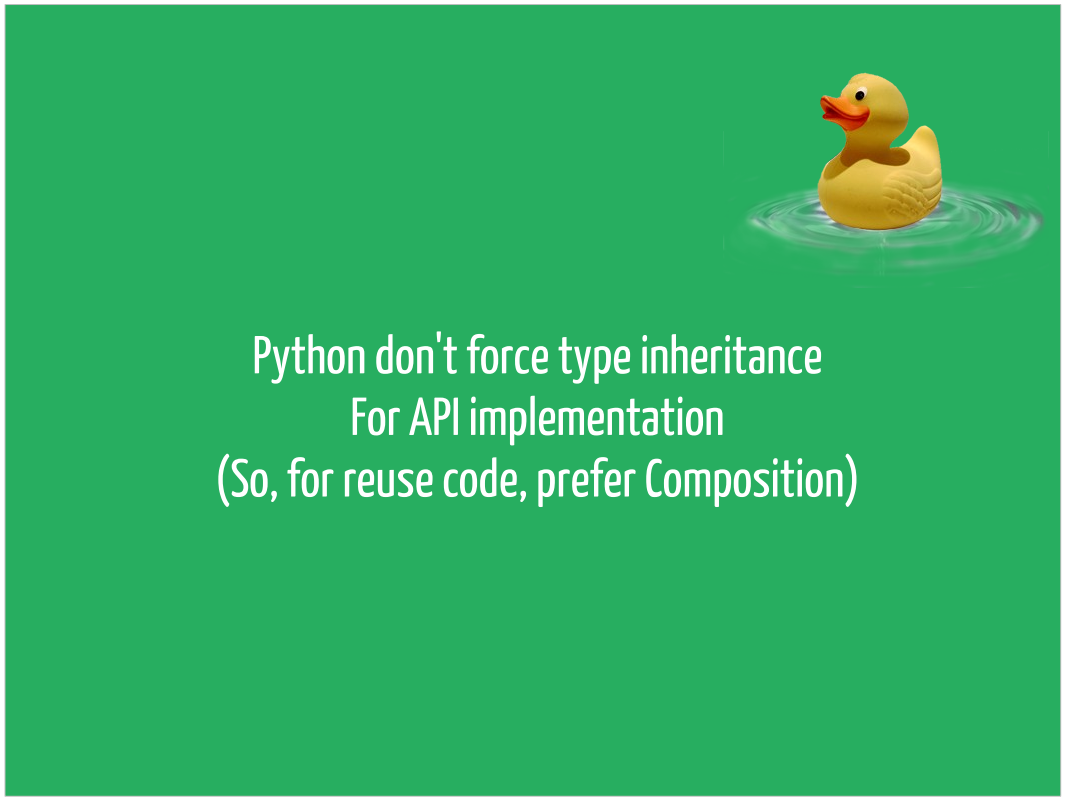


LISKOV SUBSTITUTION PRINCIPLE

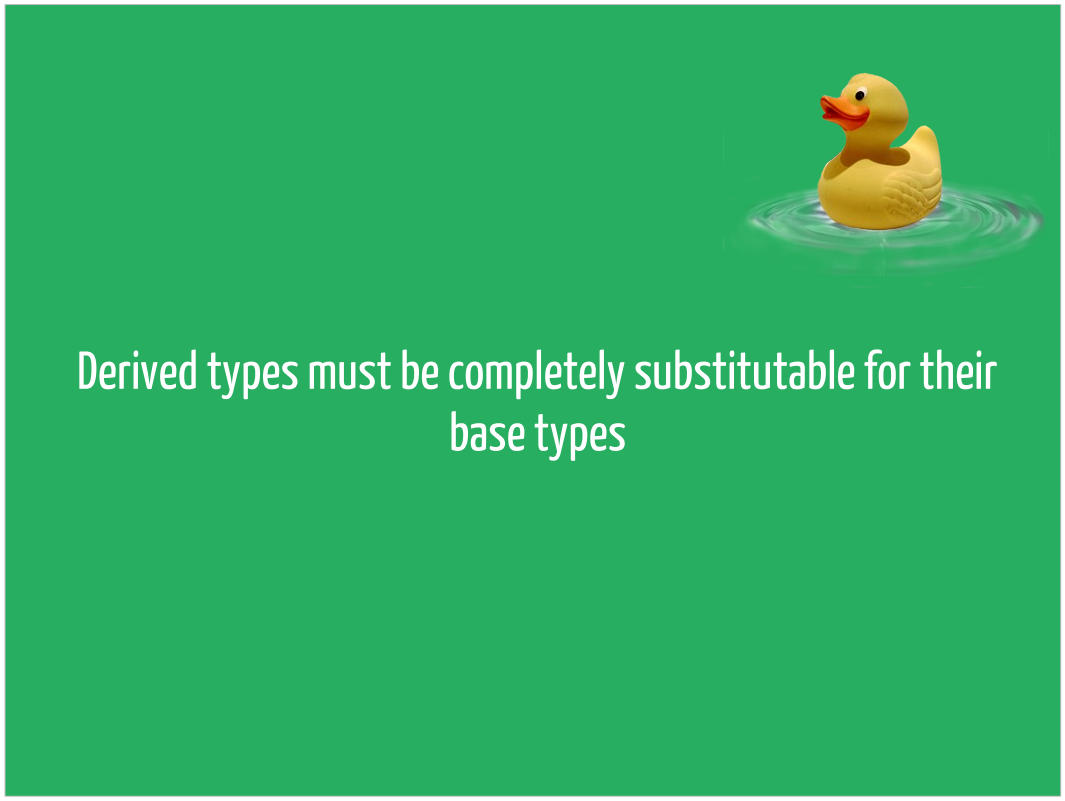
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

```python
class InMemoryJobRepository(dict):

    def put(self, job):
        self[job.service_id] = job

    def find_by_id(self, job_id):
        return self.get(job_id)

    def find_by_customer(self, customer):
        return [job for job in self.values()
            if job.has_customer(customer)]
```

Liskov Substitution principle violation



En Python herencia
- Herencia tipos (sólo se usaría para las excepciones y si usas instance_of)
- Herencia Implementación (en muchos casos es facilmente sustituible por composicion (ademas es recomendable))



En Python herencia
- Herencia tipos (sólo se usaría para las excepciones y si usas instance_of)
- Herencia Implementación (en muchos casos es facilmente sustituible por composicion (ademas es recomendable))

Interface Segregation Principle

You want me to plug this in *where?*

It isn't so important

A narrow interface is a better interface

SOLID Motivational Posters, by Derick Bailey

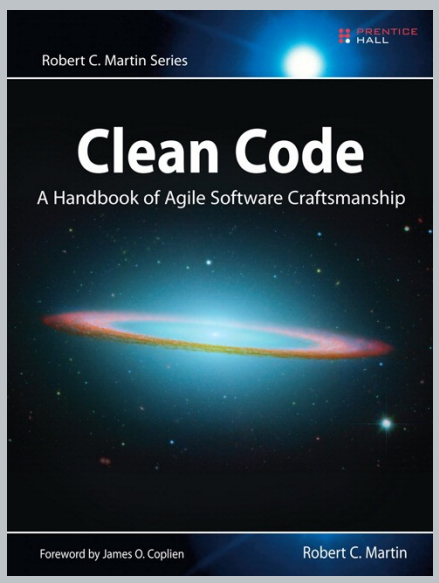
http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/

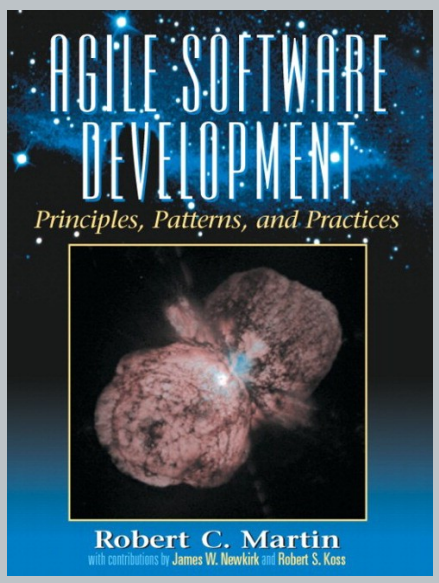
car_wash code example
https://github.com/aleasoluciones/car_wash

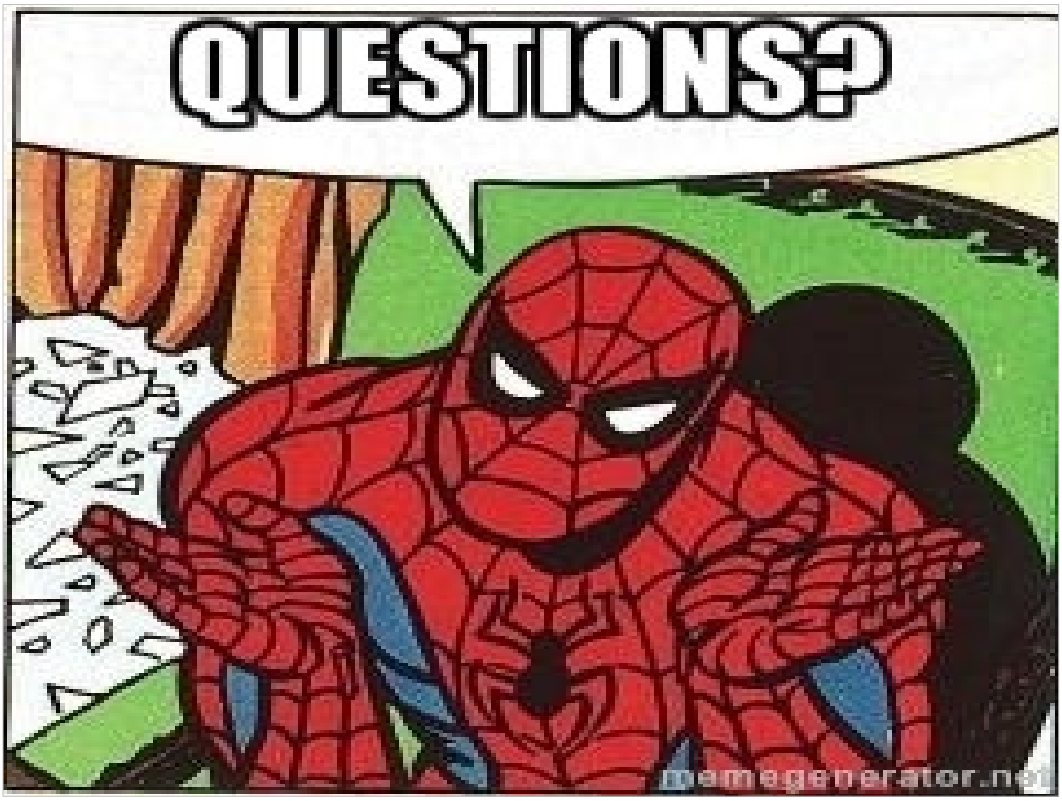
SOLID definition (at wikipedia)
http://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

Getting a SOLID start (Uncle Bob)
http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Video SOLID Object Oriented Design (Sandi Metz)
http://www.confreaks.com/videos/240-goruco2009-solid-object-oriented-design


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin


AGILE SOFTWARE DEVELOPMENT
Principles, Patterns, and Practices
Robert C. Martin
with contributions by James W. Newkirk and Robert S. Koss


QUESTIONS?
memegenerator.net

Desksurfings
Nos gusta hablar de estas cosas