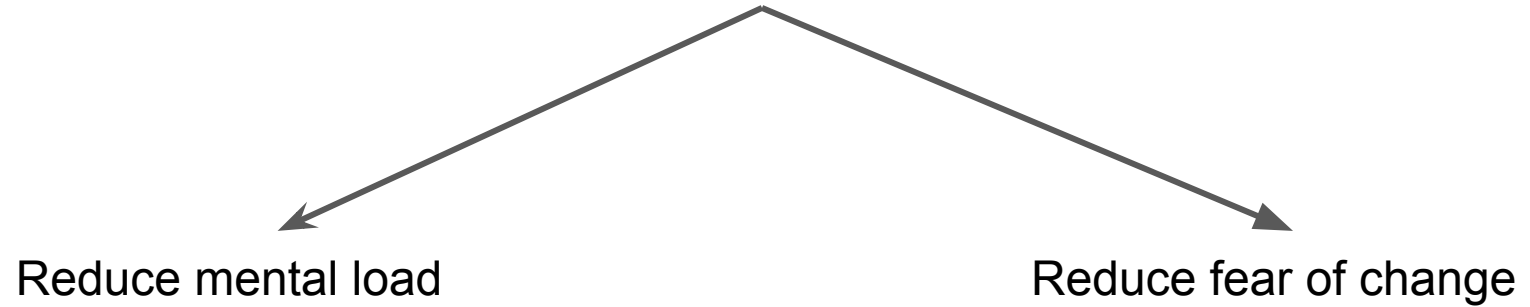


Dependency Injection (DI)

with `Dry::Container`

Build software that is easy to change!



SOLID DI TDD DDD XP

Single Responsibility (**SOLID**)



Create pressure in the fuel system

Store fuel

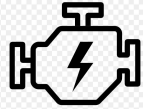
Interface Segregation (SOLID)

So you are an engine? There is a fuel pump for you!



Dependency Inversion (SOLID)

I need an



I need a Toyota 2JZGTE!

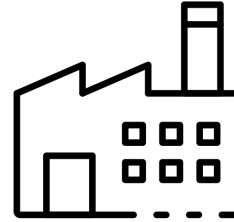


I am an



Inversion of Control (IoC)

I'll take Toyota 2JZGTE engine



Here is Toyota 2JZGTE engine



Inheritance vs Composition

class BaseCar



class BaseCarWithEngine < BaseCar



class Engine



class Car



Coupling

```
class Engine
  def accelerate
  end

  def start
  end

  def stop
  end

  def rpm
  end
end

class FuelPump
  end
```

Single Responsibility

```
class Engine2JZGTE < Engine
  def initialize
    @fuel_pump = FuelPump.new
  end

  def start
    super
  end
end

class Car
  def initialize
    @engine = Engine2JZGTE.new
  end
end

car = Car.new
```


Decouple

```
class Engine2JZGTE < Engine
  def initialize(fuel_pump) # fuel_pump: an instance of FuelPump
    @fuel_pump = fuel_pump
  end

  def start
    super
  end
end
```

```
class Car
  def initialize(engine) # engine - an instance of Engine
    @engine = engine
  end
end
```

```
fuel_pump = FuelPump.new
engine = Engine2JZGTE.new(fuel_pump: fuel_pump)
car = Car.new(engine: engine)
```

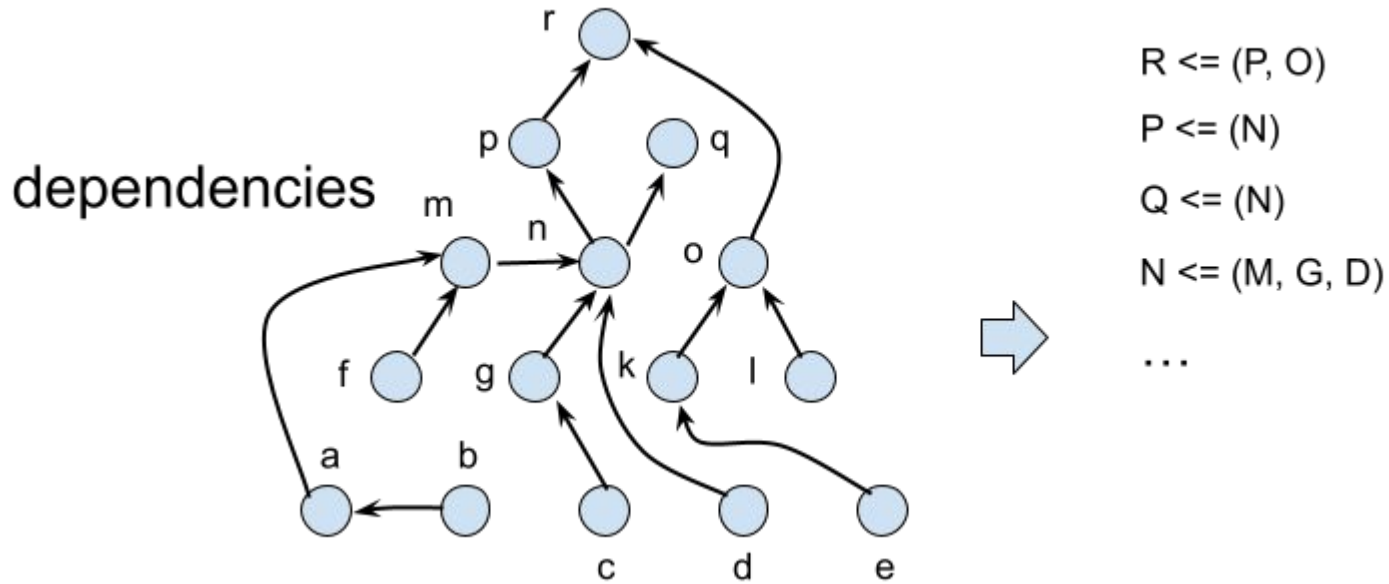
Dependency Inversion

Inversion of Control

Benefits:

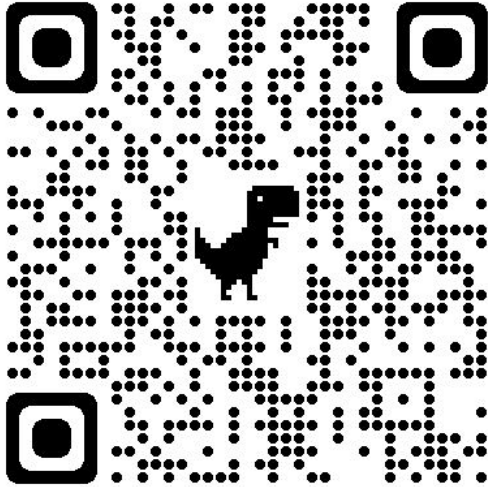
- Can clearly see all the dependencies
- Easier to change (each object and the entire app)
- Easier to test (can pass a fake object or mock as dependency, not need to patch)

Dependency Locator (Container)

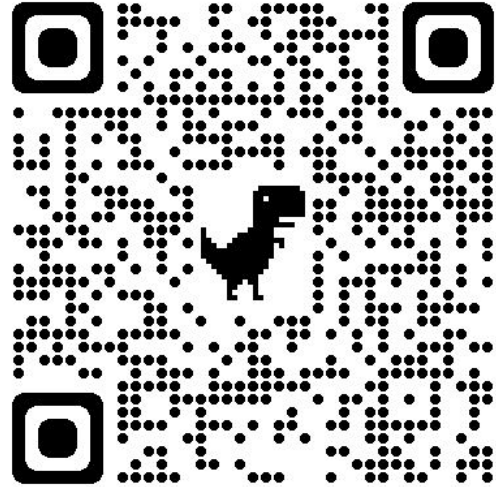


TODO List App example

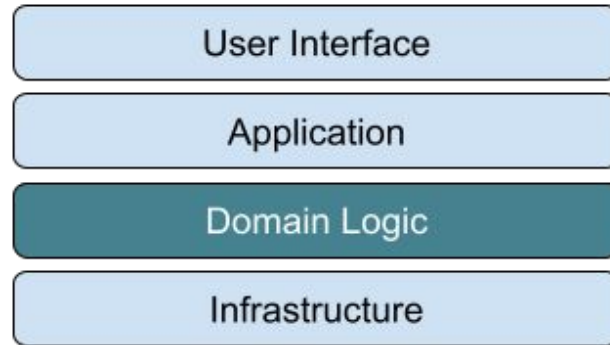
[The App on GitHub](#)



[Dry::Container](#)



Using Container inside a framework



[Using Container with Django example](#)

[Providers](#)