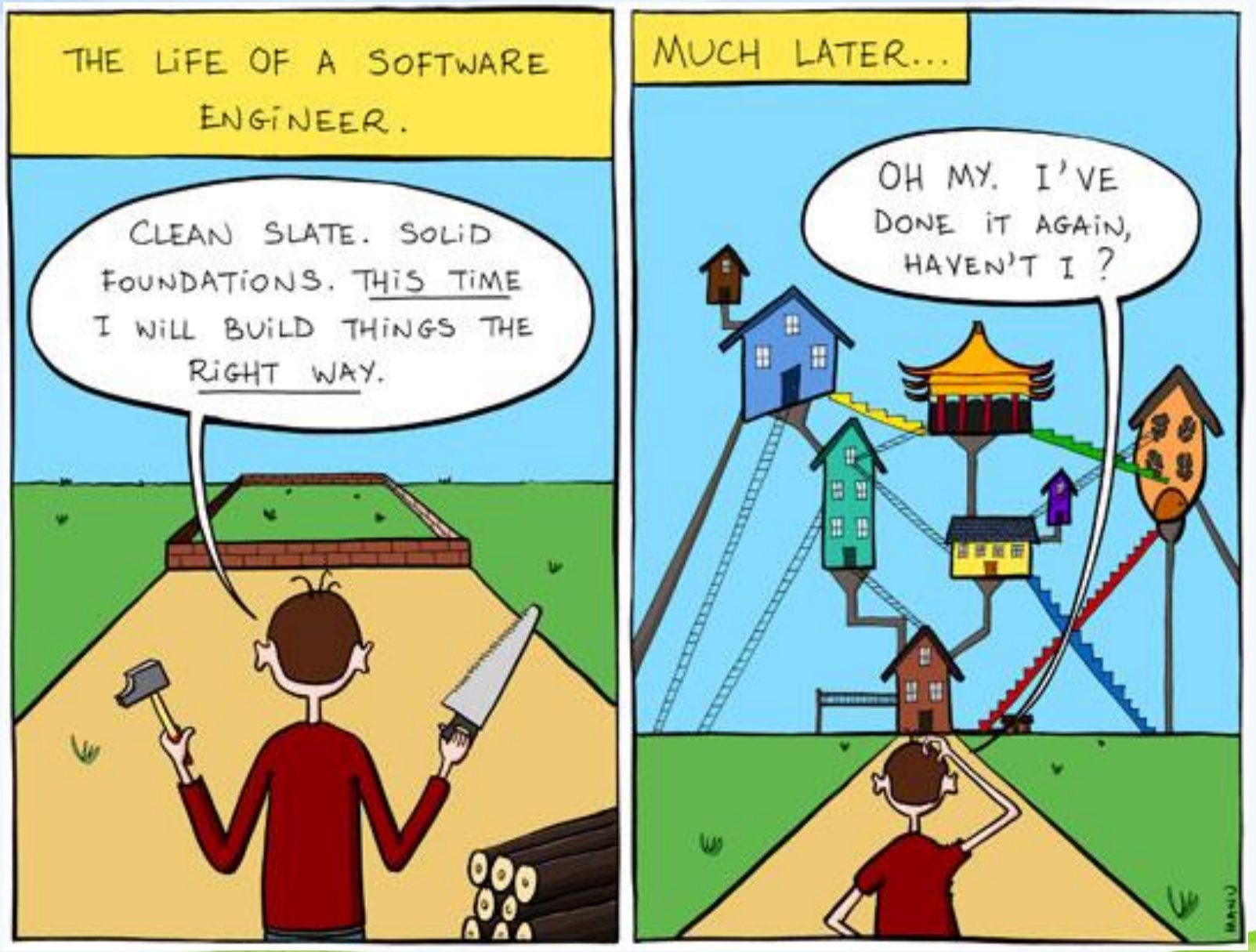
Wstrzykiwanie zależności

Tomasz Bartoszewski


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?

Co przypomina Twój kod?

Co nas boli

- Wszelkie ,,drobne'' zmiany zlecane przez klienta
- Utrudnione rozszerzanie aplikacji o dodatkowe funkcjonalności
- Klasy silnie ze sobą powiązane
- Tworzenie jednych obiektów w innych

Odwrócenie sterowania (Inversion of Control)

- ,,Nie dzwoń do nas, my zadzwonimy do Ciebie''
- Stolarz nie rąbie drzew, a mechanik nie wytwarza narzędzi

Dostarczaj obiektom to czego potrzebują!

Wstrzykiwanie zależności (Dependency Injection)

- Jeden z pomysłów realizacji IoC, obecnie najpopularniejszy
- Odwrócone sterowanie czyli tworzenie i wiązanie obiektów

Zalety

- Luźne powiązania obiektów
- Ułatwiona pielęgnacja kodu, wprowadzania zmian, dalszy rozwój
- Dodatkowy profit podczas pisania testów jednostkowych

"Software is always under constant pressure to change, so it's very important to keep your code flexible. Inversion of control helps you break your application into pieces of manageable size, and then glue them back together in a much more flexible way. Then, the next time your client asks you to make a 'little' change to the product, you can laugh triumphantly instead of quivering in pain.“

-Nate Kohari(Creator of Ninject)

Przykłady

```
public class Ballon
{
    public string Fly()
    {
        return "Lecę bo chcę";
    }
}
```

```
public class Bike
{
    public string Ride()
    {
        return "Jadę na rowerze bo to warto mieć styl";
    }
}
```

```
public class Car
{
    public string Move()
    {
        return "Jadę Polonezem na gaz";
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        var voyager = new Voyager();
        voyager.Travel();
        Console.ReadLine();
    }
}
```

```
public class Voyager
{
    private Ballon vehicle;

    public void Travel()
    {
        vehicle = new Ballon();
        Console.WriteLine(vehicle.Fly());
    }
}
```

Własne wstrzykiwanie zależności

```
public interface IVehicle
{
    string Move();
}
```

```
public class Voyager
{
    private IVehicle Vehicle;

    public Voyager(IVehicle vehicle)
    {
        Vehicle = vehicle;
    }

    public void Travel()
    {
        Console.WriteLine(Vehicle.Move());
    }
}
```

```
public class Ballon : IVehicle
{
    public string Move()
    {
        return "Lecę bo chcę";
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        var ballon = new Ballon();
        var voyager = new Voyager(ballon);
        voyager.Travel();
        Console.ReadLine();
    }
}
```

Wykorzystaj gotowe sprawdzone rozwiązania

autofac

SPRING.net
Application Framework

SpringPython

Wykorzystanie Springa

```
class Program
{
    static void Main(string[] args)
    {
        IApplicationContext context = ContextRegistry.GetContext();
        var voyager = (Voyager)context.GetObject("Voyager");
        voyager.Travel();
        Console.ReadLine();
    }
}
```

Druga strona medalu

```
<spring>
  <context>
    <resource uri="config://spring/objects" />
  </context>
  <objects xmlns="http://www.springframework.net">
    <object name="Ballon" type="DIExample.Vehicle.Ballon, DIExample" singleton="false"></object>
    <object name="Bike" type="DIExample.Vehicle.Bike, DIExample" singleton="false"></object>
    <object name="Car" type="DIExample.Vehicle.Car, DIExample" singleton="false"></object>
    <object name="Voyager" type="DIExample.Voyager, DIExample" singleton="false">
      <constructor-arg name="vehicle" ref="Ballon"/>
    </object>
  </objects>
</spring>
```

Ninject

```
internal class CustomModule : StandardModule
{
    public override void Load()
    {
        Bind<IVehicle>().To<Ballon>().Always();
    }
}
```

```
public class NinjectFactory
{
    public static IKernel GetNinjectKernel()
    {
        IModule module = new CustomModule();
        return new StandardKernel(module);
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        IKernel kernel = NinjectFactory.GetNinjectKernel();
        var programmer = kernel.Get<Voyager>();
        programmer.Travel();
        Console.ReadLine();
    }
}
```

```
public class Voyager
{
    [Inject]
    public IVehicle Vehicle { get; set; }

    public void Travel()
    {
        Console.WriteLine(Vehicle.Move());
    }
}
```


CAUTION
THIS MACHINE
HAS NO BRAIN
USE YOUR OWN
bangedupcom
M-7219