Solw'IT | Let's Solve It

Kontenery IoC w praktyce

Zakręć się z technologią

Konferencja w ramach 65-lecia WETI

Kto ja? Łukasz Rybka ☺

- Team Leader / Senior Software Developer w Solwit S.A.
- Trener / szkoleniowiec w infoShare Academy
- Wykładowca na Politechnice Gdańskiej

Agenda

- Omówienie problemy
- Dependency Injection
- Inversion of Control
- Ninject
- Q&A

Codzienne problemy developera

- Zbyt „skomplikowany” kod

Codzienne problemy developera

- Zbyt „skomplikowany” kod
- Zmiana w API generuje błędy w znaczącej części aplikacji

Codzienne problemy developera

- Zbyt „skomplikowany” kod
- Zmiana w API generuje błędy w znaczącej części aplikacji
- Zmieniające się wymagania klienta są trudne do wprowadzenia

Codzienne problemy developera

- Zbyt „skomplikowany” kod
- Zmiana w API generuje błędy w znaczącej części aplikacji
- Zmieniające się wymagania klienta są trudne do wprowadzenia
- Utrudnione testowanie aplikacji/komponentu

Prosty program do zarządzania dostawami

```
public class Program
{
    0 references
    static void Main(string[] args)
    {
        ShippingService shippingService = new ShippingService();
        shippingService.ShipProduct(args[0]);
    }
}
```

Typowy scenariusz

W trakcie sesji planowania kolejnego sprintu Product Owner prosi Zespół o rozszerzenie aplikacji do zarządzania dostawami o wysyłanie powiadomienia SMS do klienta kiedy tylko pojawi się problem z dostawą i zakupiony przedmiot dotrze do niego z opóźnieniem.

ShippingService – wersja oryginalna

```
public class ShippingService
{
    private ProductLocator locator;
    private PricingService pricingService;
    private InventoryService inventoryService;
    private TrackingRepository trackingRepository;
    private Logger logger;

    1 reference
    public ShippingService()
    {
        this.locator = new ProductLocator();
        this.pricingService = new PricingService();
        this.inventoryService = new InventoryService();
        this.trackingRepository = new TrackingRepository();
        this.logger = new Logger();
    }

    1 reference
    public void ShipProduct(string productId)
    {
        // Some very complicated logic...
    }
}
```

ShippingService – implementacja SMS

```
public class ShippingService
{
    private ProductLocator locator;
    private PricingService pricingService;
    private InventoryService inventoryService;
    private TrackingRepository trackingRepository;
    private Logger logger;
    private SMSService messageService;

    1 reference
    public ShippingService()
    {
        this.locator = new ProductLocator();
        this.pricingService = new PricingService();
        this.inventoryService = new InventoryService();
        this.trackingRepository = new TrackingRepository();
        this.logger = new Logger();
        this.messageService = new SMSService();
    }

    // ...
```

Pytanie do publiczności

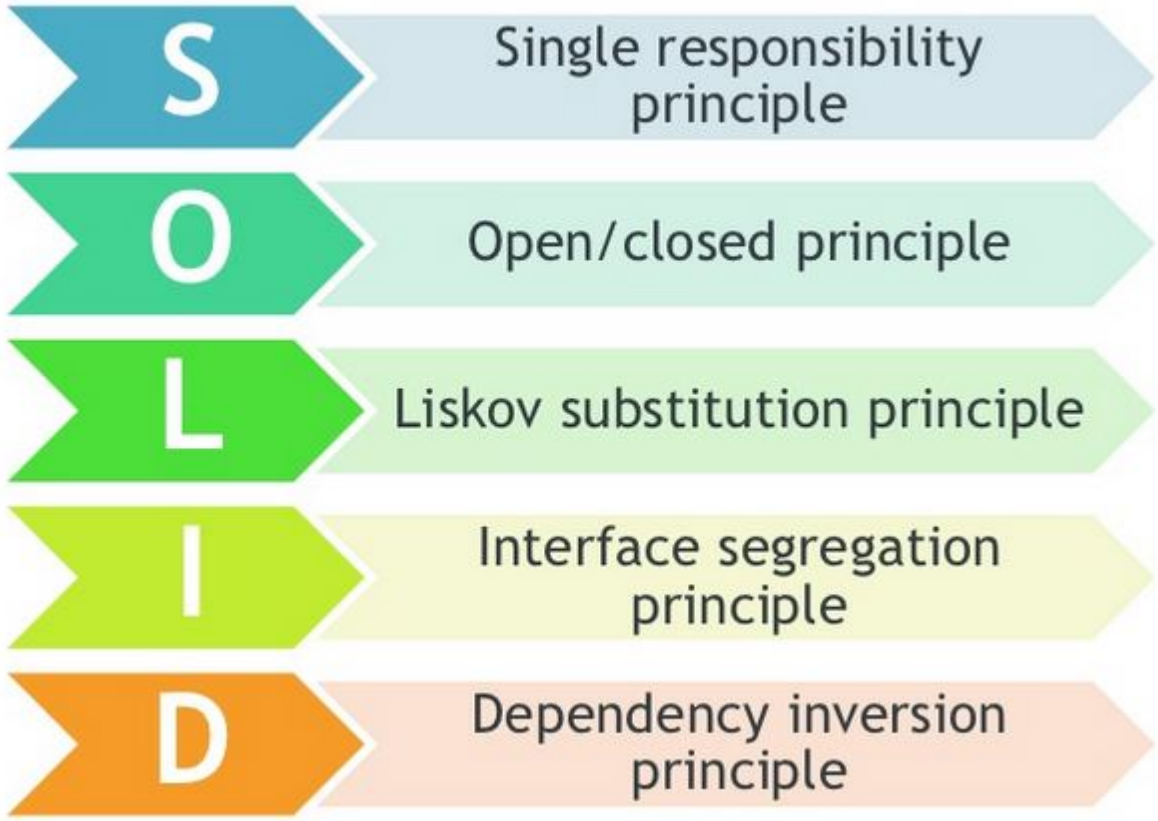
Czy taki kod da się łatwo przetestować jednostkowo?

Pytanie do publiczności

Jaka zasada programowania została tutaj złamana?

Pytanie do publiczności

Jaka zasada programowania została tutaj złamana?



Dependency Injection

- Wzorzec projektowy / technika, w której jeden obiekt dostarcza zależności drugiego obiektu
- Pozwala na elastyczne konfigurowanie logiki aplikacji
- „Rozrywa” ścisłe połączenie między obiektami
- Nie jest bez wad...

Dependency Injection – podejście #1

```
public ShippingService(ProductLocator locator, PricingService pricingService,
    InventoryService inventoryService, TrackingRepository trackingRepository,
    Logger logger, SMSService messageService)
{
    this.locator = locator;
    this.pricingService = pricingService;
    this.inventoryService = inventoryService;
    this.trackingRepository = trackingRepository;
    this.logger = logger;
    this.messageService = messageService;
}
```

Dependency Injection – podejście #1

```
public class Program
{
    0 references
    static void Main(string[] args)
    {
        var productLocator = new ProductLocator();
        var pricingService = new PricingService();
        var inventoryService = new InventoryService();
        var trackingRepository = new TrackingRepository();
        var logger = new Logger();
        var messageService = new SMSService();

        ShippingService shippingService = new ShippingService(productLocator, pricingService, inventoryService,
            trackingRepository, logger, messageService);

            shippingService.ShipProduct(args[0]);
        }
}
```

Typowy scenariusz – zmiana serwisu

W trakcie kolejnej sesji planowania sprintu Product Owner prosi Zespół o zmianę kanału powiadamiania klienta o opóźnieniach w dostarczeniu produktu z SMS'ów na wiadomości email.

Dependency Injection – podejście #2

```
public class ShippingService
{
    private IProductLocator locator;
    private IPricingService pricingService;
    private IInventoryService inventoryService;
    private ITrackingRepository trackingRepository;
    private ILogger logger;
    private IMessageService messageService;

    1 reference
    public ShippingService(IProductLocator locator, IPricingService pricingService,
        IInventoryService inventoryService, ITrackingRepository trackingRepository,
        ILogger logger, IMessageService messageService)
    {
        this.locator = locator;
        this.pricingService = pricingService;
        this.inventoryService = inventoryService;
        this.trackingRepository = trackingRepository;
        this.logger = logger;
        this.messageService = messageService;
    }

    // ...
```

Dependency Injection – podejście #2

```
public class Program
{
    0 references
    static void Main(string[] args)
    {
        var productLocator = new ProductLocator();
        var pricingService = new PricingService();
        var inventoryService = new InventoryService();
        var trackingRepository = new TrackingRepository();
        var logger = new Logger();
        var messageService = new EmailService();

        ShippingService shippingService = new ShippingService(productLocator, pricingService, inventoryService,
            trackingRepository, logger, messageService);

            shippingService.ShipProduct(args[0]);
        }
}
```

Kolejne zmiany w aplikacji

Product Owner prosi Zespół o zmianę kanału powiadamiania klienta o opóźnieniach w dostarczeniu produktu na SMS'y w całej aplikacji (nie tylko fragmencie odpowiedzialnym za dostawy).

Konfiguracja projektu ma być od teraz zaszyfrowana.

Sposób logowania zmienia się z wiadomości email na wywołania REST API.

...

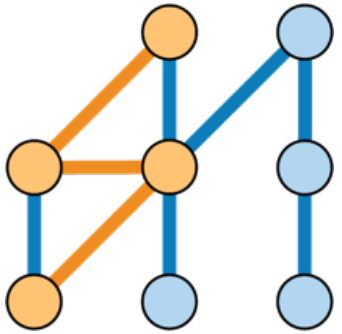
Inversion of Control (IoC)

- Wzorzec projektowy, w którym fragment aplikacji otrzymuje „flow of control” z generycznego narzędzia
- Zwiększa modularność aplikacji oraz jej rozszerzalność (zasada SOLID!)
- Dany moduł koncentruje się jedynie na logice z myślą o której został zaprojektowany

> *I think that people who use IoC containers are (A) very smart and (B) lacking in empathy for people who aren't as smart as they are. Everything makes perfect sense to them, so they have trouble understanding that many ordinary programmers will find the concepts confusing.*

Joel Spolsky

Który framework IoC wybrać?

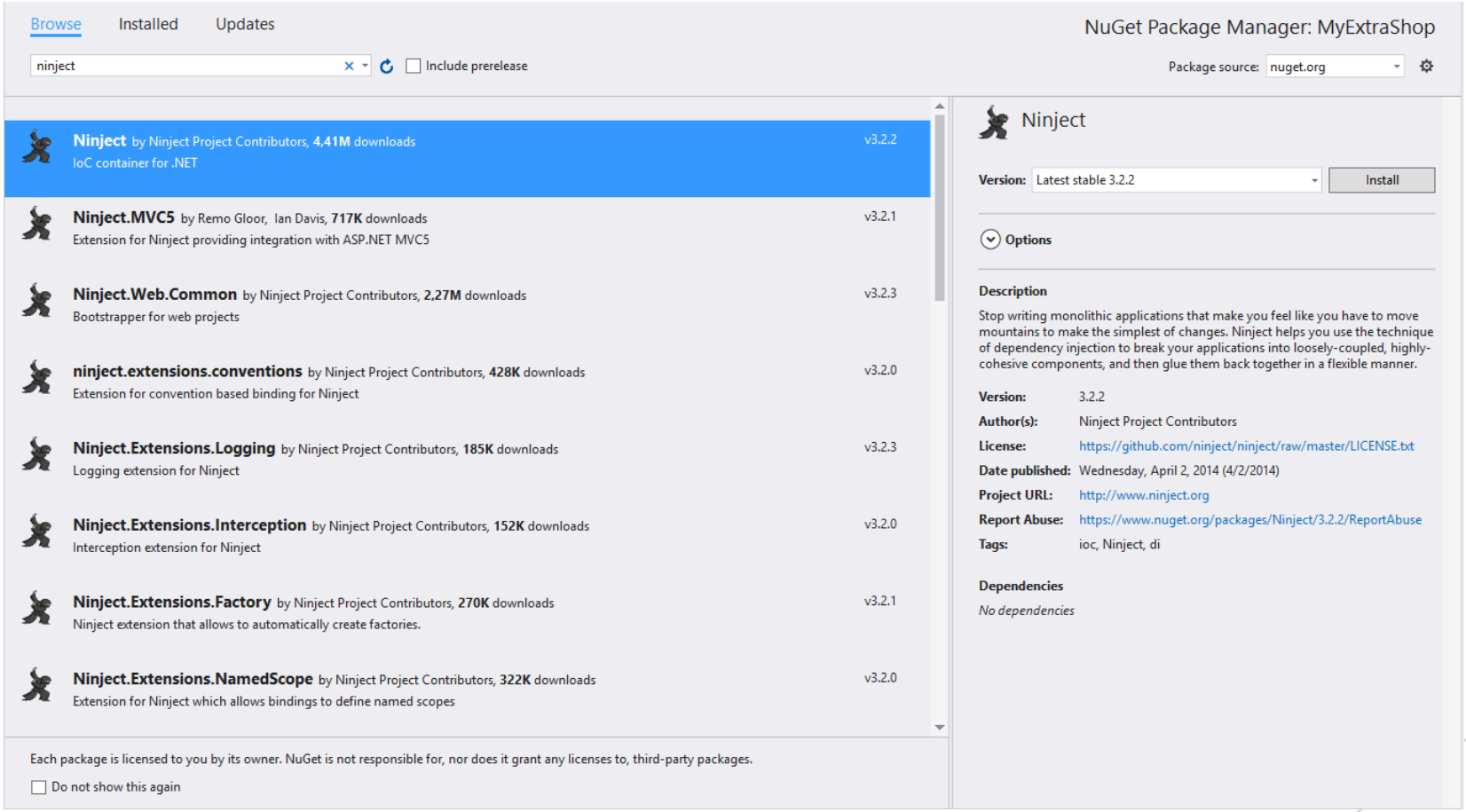


Microsoft®
patterns & practices
proven practices for predictable results

Ninject

- Relatywnie łatwy i prosty w użyciu
- Szybki – oparty na generowaniu kodu w CLR zamiast refleksi (8-50x szybszy)
- Zbudowany z myślą o modularności i rozszerzalności

Ninject – instalacja (NuGet)



Ninject – wykorzystanie (Program.cs)

```
public class Program
{
    0 references
    static void Main(string[] args)
    {
        IKernel kernel = new StandardKernel();

        kernel.Bind<IProductLocator>().To<ProductLocator>();
        kernel.Bind<IPricingService>().To<PricingService>();
        kernel.Bind<IInventoryService>().To<InventoryService>();
        kernel.Bind<ITrackingRepository>().To<TrackingRepository>();
        kernel.Bind<ILogger>().To<Logger>();
        kernel.Bind<IMessageService>().To<EmailService>();

        var shippingService = kernel.Get<ShippingService>();
        shippingService.ShipProduct(args[0]);
    }
}
```

Container Manager

- Jedna klasa odpowiedzialna za konfigurację kontenera IoC
- Centralne miejsce do zmiany logiki

Container Manager

```
public class ContainerManager
{
    private readonly IKernel kernel;

    1 reference
    public ContainerManager()
    {
        this.kernel = new StandardKernel();

        this.LoadModules();
    }

    1 reference
    public void LoadModules()
    {
        this.kernel.Bind<IProductLocator>().To<ProductLocator>();
        this.kernel.Bind<IPricingService>().To<PricingService>();
        this.kernel.Bind<IInventoryService>().To<InventoryService>();
        this.kernel.Bind<ITrackingRepository>().To<TrackingRepository>();
        this.kernel.Bind<ILogger>().To<Logger>();
        this.kernel.Bind<IMessageService>().To<EmailService>();
    }

    1 reference
    public T Get<T>()
    {
        return this.kernel.Get<T>();
    }
}
```

Container Manager – Program.cs

```
public class Program
{
    0 references
    static void Main(string[] args)
    {
        ContainerManager manager = new ContainerManager();

        var shippingService = manager.Get<ShippingService>();

        shippingService.ShipProduct(args.Length > 0 ? args[0] : string.Empty);
    }
}
```

Container Manager - singleton

```
public class ContainerManager
{
    private static volatile ContainerManager instance;
    private static readonly object SingletonLock = new object();
    private readonly IKernel kernel;

    2 references
    private ContainerManager()
    {
        this.kernel = new StandardKernel();

        this.LoadModules();
    }

    0 references
    public static ContainerManager Instance
    {
        get
        {
            if (instance == null)
            {
                lock (SingletonLock)
                {
                    if (instance == null)
                    {
                        instance = new ContainerManager();
                    }
                }
            }

            return instance;
        }
    }
```

Container Manager - singleton

```
public class Program
{
    0 references
    static void Main(string[] args)
    {
        var shippingService = ContainerManager.Instance.Get<ShippingService>();

        shippingService.ShipProduct(args.Length > 0 ? args[0] : string.Empty);
    }
}
```

Ninject Modules

- Jeden moduł zawiera definicję wiązania wielu typów
- Pozwala na organizację wiązań w grupy według odpowiedzialności / przynależności
- Prosta klasa (zamiast pliku XML) rozszerzająca NinjectModule (implementująca INinjectModule)

Ninject Modules

```
public class ContainerManagerModule : NinjectModule
{
    0 references
    public override void Load()
    {
        Kernel.Bind<IProductLocator>().To<ProductLocator>();
        Kernel.Bind<IPricingService>().To<PricingService>();
        Kernel.Bind<IInventoryService>().To<InventoryService>();
        Kernel.Bind<ITrackingRepository>().To<TrackingRepository>();
        Kernel.Bind<ILogger>().To<Logger>();
        Kernel.Bind<IMessageService>().To<EmailService>();
    }
}
```

```
private ContainerManager()
{
    this.kernel = new StandardKernel(new ContainerManagerModule());
}
```

Injection Patterns

- Różne możliwości wstrzykiwania zależności do komponentu (klasy)
- Zarówno wzorzec IoC jak i sam Ninject umożliwiają wstrzykiwanie:
  - W konstruktorze
  - Parametru
  - Setter’a
  - Interfejsu

Injection Patterns – Property Setter Injection

```
public class EmailLogger
{
    [Inject]
    1 reference
    public IConfigProvider configProvider { private get; set; }

    0 references
    public void Log()
    {
        this.configProvider.Load();
    }
}
```

Problem – argumenty konstruktora

```
public class ConfigProvider : IConfigProvider
{
    private readonly string secret;

    1 reference
    public ConfigProvider(string secret)
    {
        this.secret = secret;
    }

    2 references
    public void Load()
    {
    }
}
```

Argumenty konstruktora w Ninject

```
this.kernel.Get<IConfigProvider>(new ConstructorArgument("secret", "some-top-secret-key"));
```

```
Kernel.Bind<IConfigProvider>().To<ConfigProvider>().WithConstructorArgument("secret", SECRET);
```

IoC (+ Ninject) – what's next

- Providers, Factory Methods oraz Activation Context
- Activation Process
- Contextual Binding
- Conventions-Based Binding

Q&A

Solw'IT | Let's Solve It

SOFTWARE ENGINEERING

Thanks!