Dependency Injection Containers

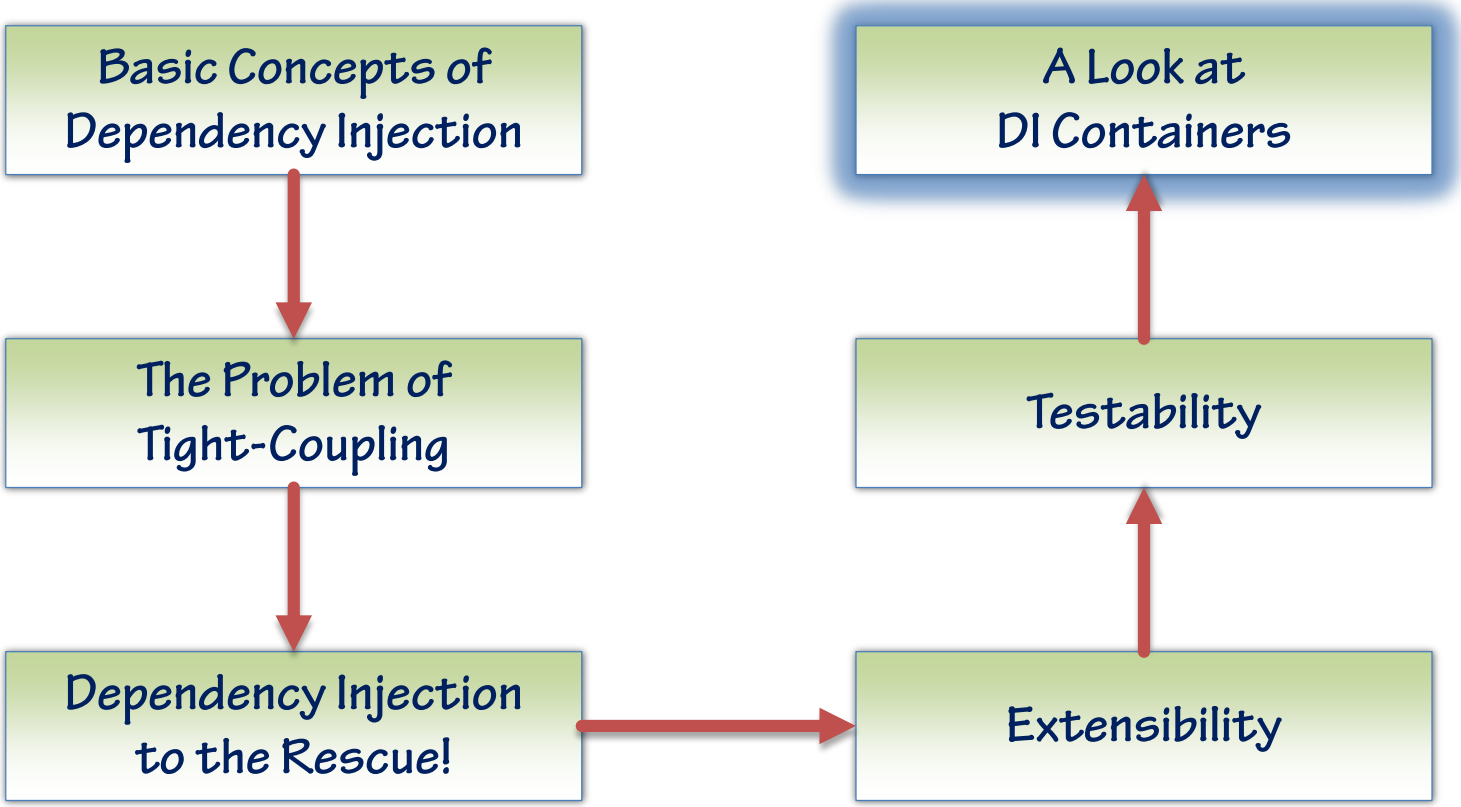
A Look at the Advantages of Using a DI Container

Jeremy Clark
www.jeremybytes.com
jeremy@jeremybytes.com

pluralsight
hardcore developer training

Goal

- Get Comfortable with Dependency Injection

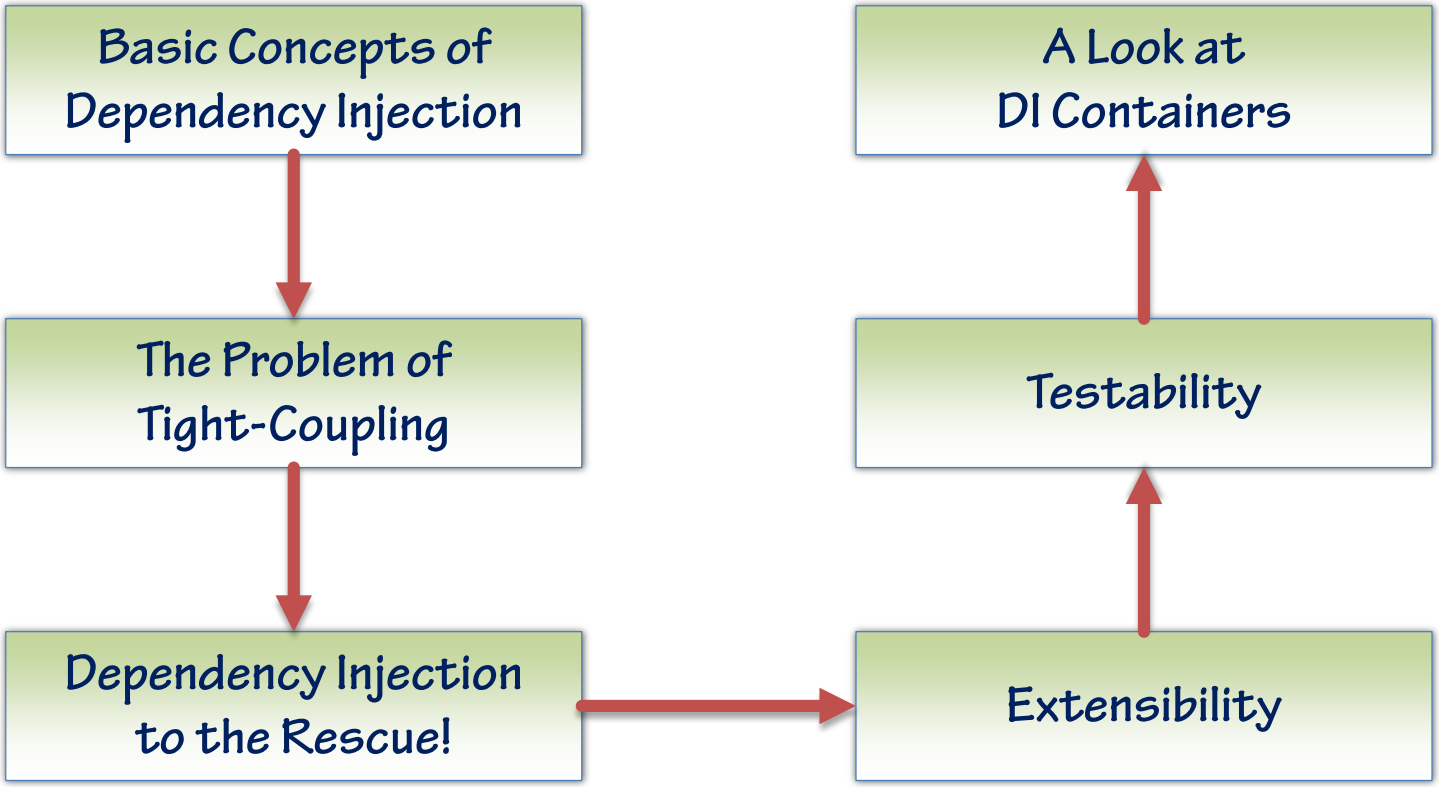


Dependency Injection Containers

- **Unity**
  - Microsoft Patterns & Practices
- **Spring.NET**
  - .NET Port of the Java Spring Framework
- **Castle Windsor**
  - Part of the Castle Project
- **Ninject**
- **Autofac**
- **StructureMap**
- **Plus, many, many others**

Goal

- Get Comfortable with Dependency Injection



What is Dependency Injection?

Dependency Injection is a set of software design principles and patterns that enable us to develop loosely coupled code.

-Mark Seemann

Seemann. *Dependency Injection in .NET*. Manning, 2012.

Why Loosely-Coupled Code?

- **Extensibility**
- **Testability**
- **Late Binding**
- **Parallel Development**
- **Maintainability**

Dependency Injection Concepts

- **Patterns**
  - Constructor Injection
  - Property Injection
  - Method Injection
  - Ambient Context
  - Service Locator
- **Object Composition**
  - Composition Root
- **DI Containers**
  - Unity, Ninject, Castle Windsor, Autofac, StructureMap, Spring.NET, and many others

Constructor Injection & Composition Root

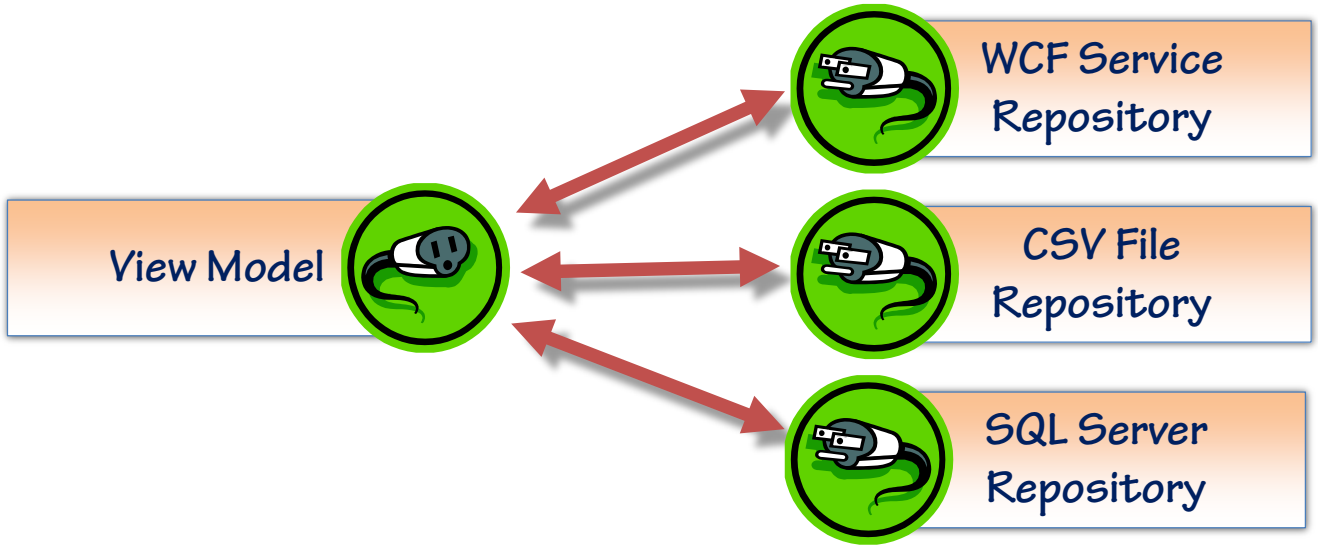
```
public class PeopleViewerViewModel : INotifyPropertyChanged
{
    protected IPersonRepository Repository;

    public PeopleViewerViewModel(IPersonRepository repository)
    {
        Repository = repository;
    }
    ...
}
```

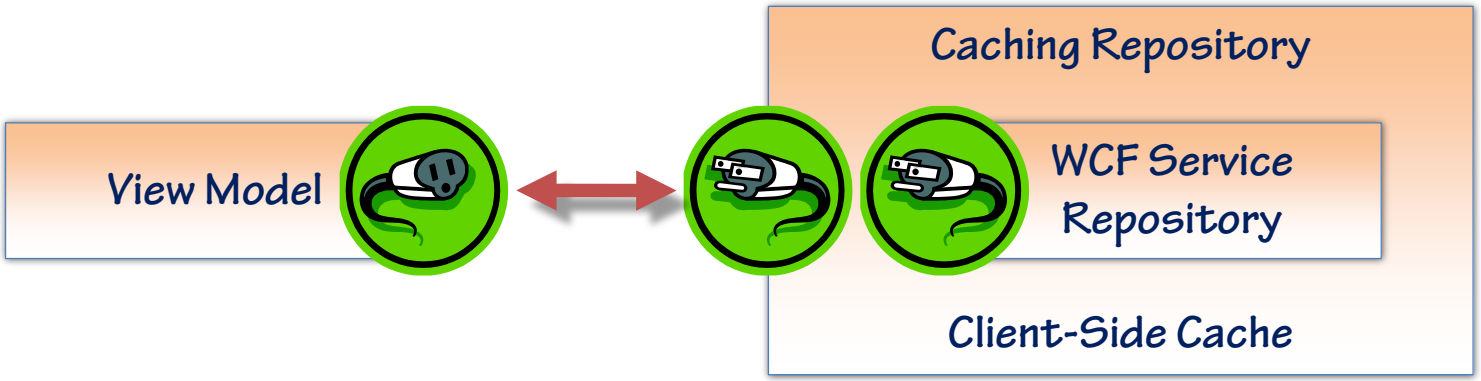
```
private static void ComposeObjects()
{
  var repository = new ServiceRepository();
  var viewModel = new PeopleViewerViewModel(repository);
  Application.Current.MainWindow = new PeopleViewerWindow(viewModel);
}
```

- **The Repository is "Someone Else's Problem"**
- **The Composition Root snaps our blocks of code together**

Different Repositories



Client-Side Caching



Unit Testing with Loose Coupling

```
public class PeopleViewerViewModel : INotifyPropertyChanged
{
    protected IPersonRepository Repository;

    public PeopleViewerViewModel(IPersonRepository repository)
    {
        Repository = repository;
    }
    ...
}
```

- The View Model is longer tied to the ServiceRepository
- We can use a fake or mock Repository for Unit Testing

Property Injection

```
public class ServiceRepository : IPersonRepository
{
    private IPersonService _serviceProxy;
    public IPersonService ServiceProxy
    {
        get
        {
            if (_serviceProxy == null)
                _serviceProxy = new PersonServiceClient();
            return _serviceProxy;
        }
        set { _serviceProxy = value; }
    }
}
```

- PersonServiceClient will be used by default
- We can change the default by assigning to the ServiceProxy property
- Useful for swapping out a fake or mock in testing

Dependency Injection Containers

- **Lifetime Management**
  - Singleton
  - Transient
  - Per Thread
- **Object Resolution**
  - Automatically Creates Dependencies Required By Constructor Injection
  - Configurable for Property Injection Objects
- **Late Binding**
  - Dynamic Loading through Configuration
- **Unit Testing with a Container**

Dependency Injection On-Ramp

An Introduction to the Principles of Dependency Injection

Jeremy Clark
www.jeremybytes.com
jeremy@jeremybytes.com

pluralsight
hardcore developer training