Unit Testing

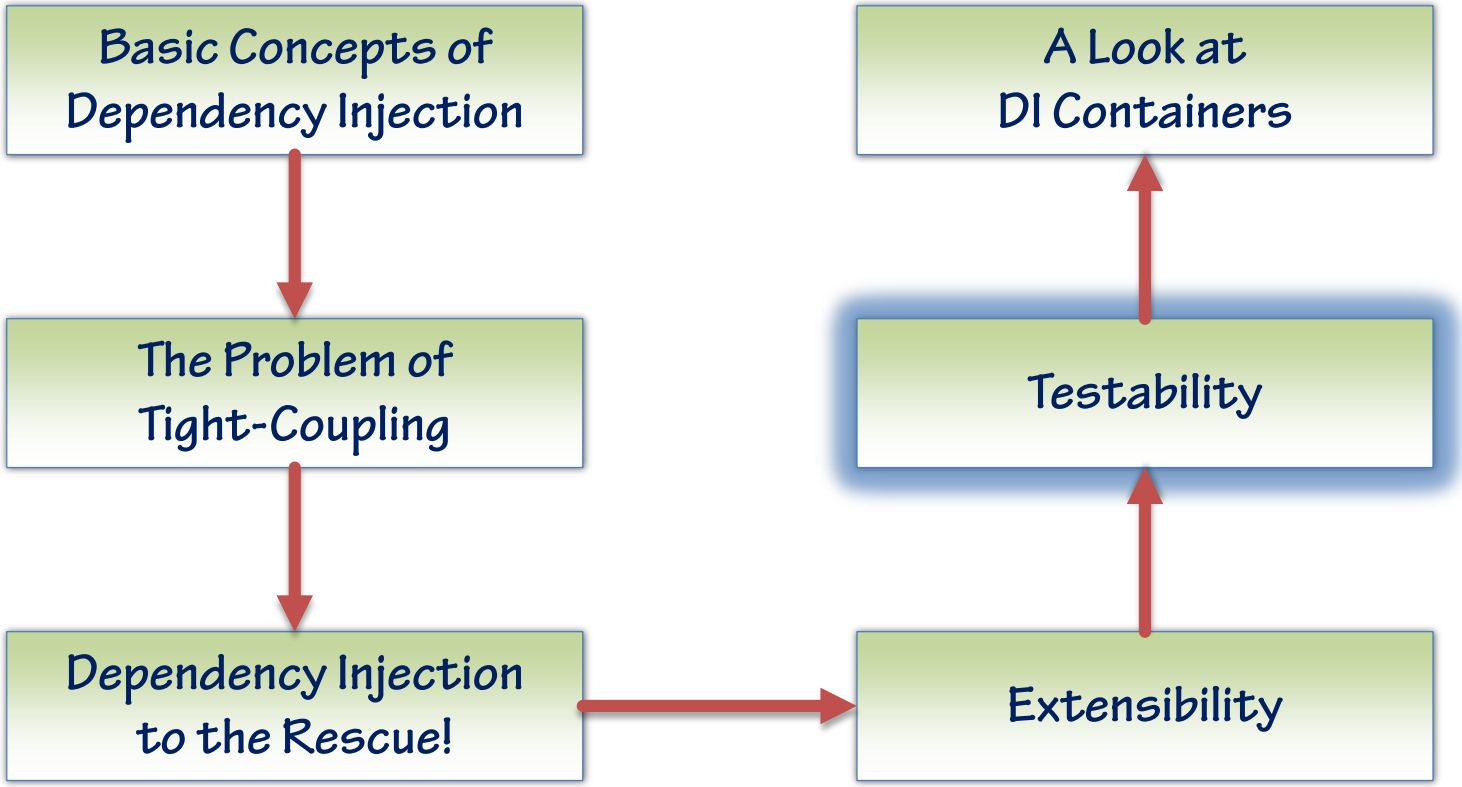
Easy Unit Testing with Dependency Injection

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

pluralsight
hardcore developer training

Goal

- Get Comfortable with Dependency Injection



Unit Testing

- **Testing small pieces of code**
  - Usually on the method level
- **Testing in isolation**
  - Eliminate outside interactions that might break the test
  - Reduce the number of objects needed to run the test
- **Note: We still need Integration Testing**
  - Testing that the pieces all work together

View Model Unit Tests

```
public class PeopleViewerViewModel : INotifyPropertyChanged
{
    public IEnumerable<Person> People { ... }

    public void Execute(object parameter) // RefreshPeopleCommand
    {
        ViewModel.People = ViewModel.Repository.GetPeople();
    }

    public void Execute(object parameter) // ClearPeopleCommand
    {
        ViewModel.People = new List<Person>();
    }
}
```

- RefreshPeopleCommand – Execute method
- ClearPeopleCommand – Execute method
- People property

Unit Testing with Tight-Coupling

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

    public PeopleViewerViewModel()
    {
        Repository = new ServiceRepository();
    }
    ...
}
```

```
public class ServiceRepository
{
    PersonServiceClient _serviceProxy = new PersonServiceClient();
    ...
}
```

- **To test the View Model**
  - Must create a ServiceRepository
  - Must create a PersonServiceClient
  - Service must be running

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

Mocking

- **Create "Placeholder" Objects**
  - In-Memory
  - Only Implement Behavior We Care About
- **Great for Unit Testing**
- **Mocking Frameworks**
  - RhinoMocks
  - Microsoft Fakes
  - Moq

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

Summary

- **Unit Testing**
  - Easy Isolation with Constructor Injection
- **Mocking**
  - Easy to Add Fake Dependencies for Testing
- **Property Injection**
  - Swap Out a Dependency for Testing

- **Next Up: Dependency Injection Containers**
  A Look at Unity and Ninject