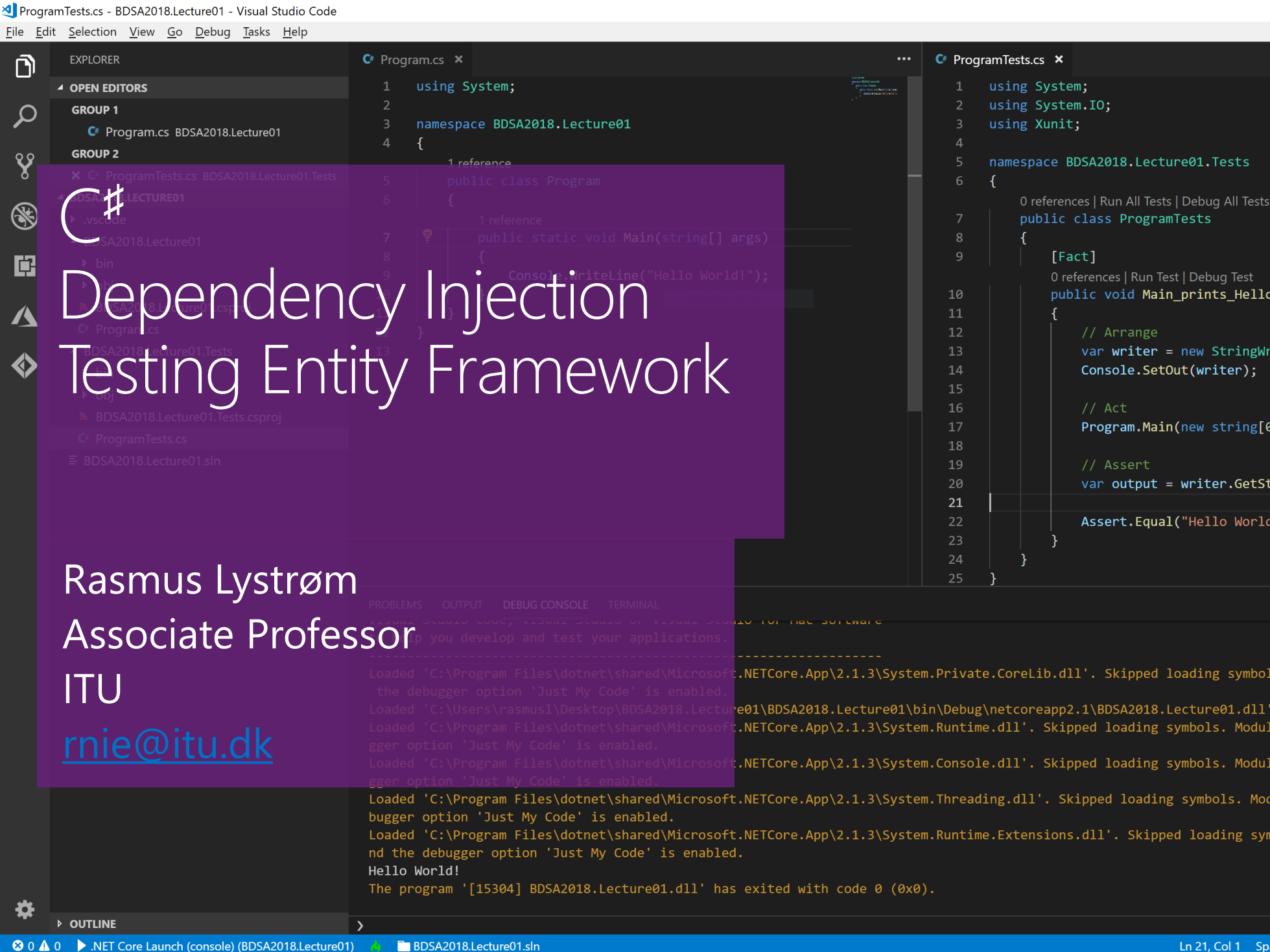


C#
Dependency Injection
Testing Entity Framework

Rasmus Lystrøm
Associate Professor
ITU
rnie@itu.dk

Agenda

Testing …

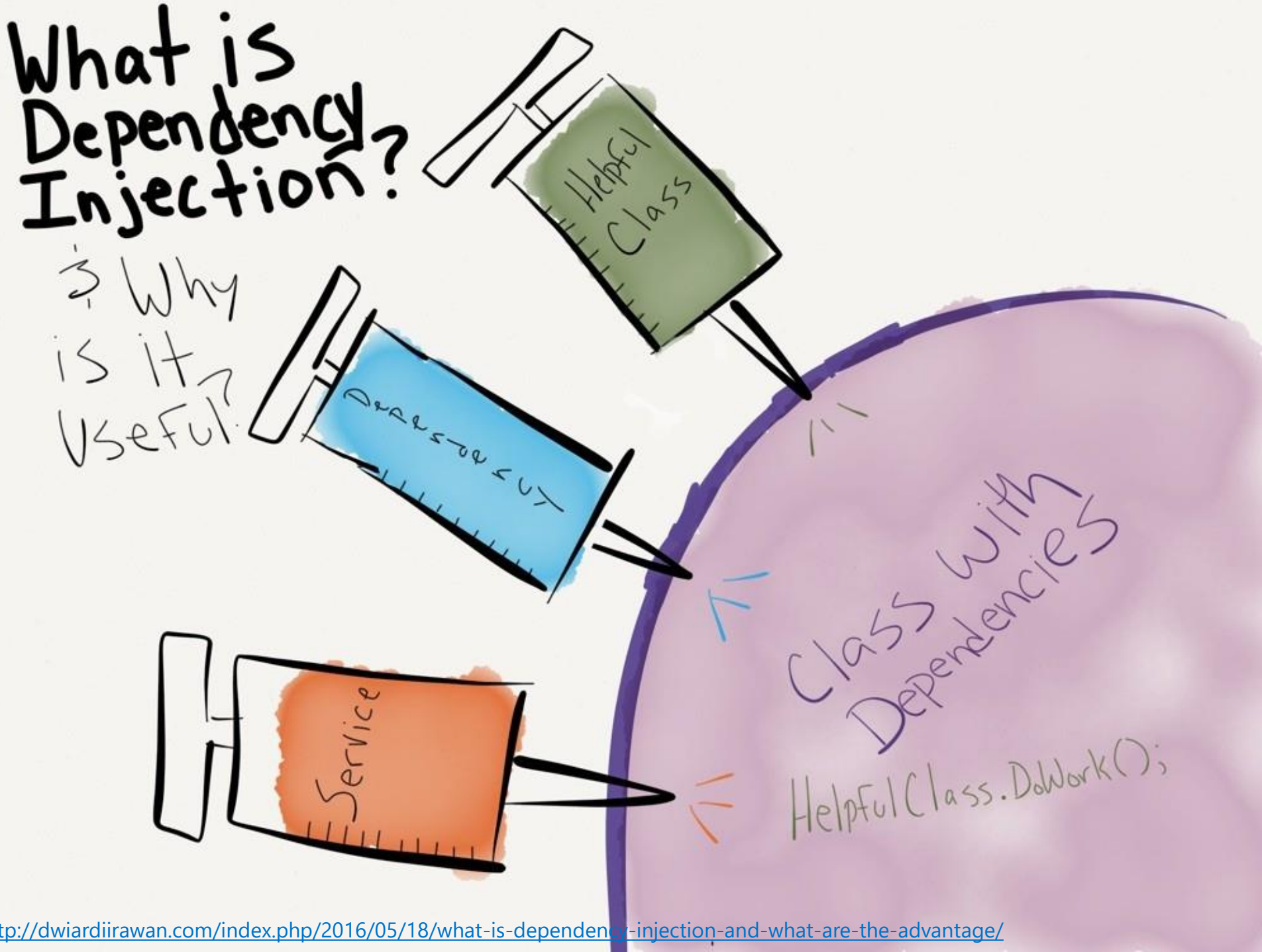
Dependency Injection

Testing Entity Framework

Testing …

Timing

a. Testing live databases is hard
b. Testing live full systems is hard
c. By transitivity: Testing … is hard…



http://dwiardiirawan.com/index.php/2016/05/18/what-is-dependency-injection-and-what-are-the-advantage/

Dependency Injection (DI)

Software design pattern which implements Inversion of Control (IoC)

Constructor Injection

Property (setter) Injection

Interface Injection

Structered readable code
Testable code
Dependency Inversion Principle
Separation of Concerns

Rock SOLID!!!

AWESOME!!

Programming to interface, not implementation…

```
public interface IFooService
{
    bool Update(Foo foo);
}

public class FooService : IFooService
{
    bool Update(Foo foo)
    {
        // Implementation
    }
}
```

Using IF

```
public clas
{
    public
    {
        IFc
        var

        IFc

        ret
    }
}
```

Constructor Injection (preferred)

```
public class Worker
{
    private readonly IFooService _service;

    public Worker(IFooService service)
    {
        _service = service;
    }

    public bool DoWork(FooDto fooDto)
    {
        // Implementation
    }
}
```

Private readonly field

Initialize from constructor

Property Injection

Public setter

```
public class Worker
{
    public IFooService Service { private get; set; }

    public void DoWork(Foo foo)
    {
        Service?.Update(foo);
    }
}
```

Is this King?

Interface Injection

```
public interface IServiceSetter<T>
{
    void SetService(T service);
}

public interface IServiceSetter<T>
{
    T Service { set; }
}
```

Interface Injection II

Interface

```
public class Worker : IServiceSetter<IFooService>
{
    private IFooService _service;

    public void SetService(IFooService service)
    {
        _service = service;
    }

    public void DoWork(FooDto fooDto)
    {
        // Implementation
    }
}
```

Implement interface

Interface Injection III

```
public class Worker : IServiceSetter<IFooService>
{
    public IFooService Service { private get; set; }

    public bool DoWork(FooDto fooDto)
    {
        // Implementation
    }
}
```

Interface

Implement interface

Best practices

Use Adapter to enable interface if needed

Use constructor injection

Program to interface

Use an IoC container

In a couple of weeks...

IoC Container

```
PM> Install-Package Microsoft.Extensions.DependencyInjection
```

```
IServiceCollection services = new ServiceCollection();

services.AddScoped<IService, Service>();

var provider = services.BuildServiceProvider();

var service = provider.GetRequiredService<IService>();
```

Unit Testing

Unit Testing Best Practices

Never test against a live database, file, or web service

Single Responsibility Principle

Atomic tests

Only test the "System Under Test"

Use either mocks or stubs

Stub testing


```
public class FooServiceFalseStub : IFooService
{
    public bool Update(Foo foo)
    {
        return false;
    }
}
```

Stub testing II

```
public class WorkerTests
{
    [Fact]
    public void DoWork_when_IFooService_Update_false_returns_false()
    {
        IFooService service = new FooServiceFalseStub();

        using (var worker = new Worker(service))
        {
            var result = worker.DoWork(new FooDto());

            Assert.False(result);
        }
    }
}
```

Mock testing

Mock using Moq

```
public class WorkerTests
{
    [Fact]
    public void DoWork_when_IFooService_Update_false_returns_false()
    {
        var mock = new Mock<IFooService>();
        IFooService service = mock.Object;

        using (var worker = new Worker(service))
        {
            var result = worker.DoWork(new FooDto());

            Assert.False(result);
        }
    }
}
```

Mock testing II

Configure the mock

```
public class WorkerTests
{
    [Fact]
    public void DoWork_when_IFooService_Updat_true_returns_true()
    {
        var mock = new Mock<IFooService>();
        mock.Setup(m => m.Update(It.IsAny<Foo>())).Returns(true);

        using (var worker = new Worker(mock.Object))
        {
            var result = worker.DoWork(new FooDto());

            Assert.True(result);
        }
    }
}
```

Demo

Testing Entity Framework

In Memory Database

```
dotnet add package Microsoft.EntityFrameworkCore.InMemory
```

```
// In Memory Database:
var builder = new DbContextOptionsBuilder<MyContext>()
              .UseInMemoryDatabase(databaseName: nameof(<name>));
```

```
dotnet add package Microsoft.EntityFrameworkCore.SqlLite
```

```
// SQLite:
var connection = new SqliteConnection("DataSource=:memory:");
connection.Open();

var builder = new DbContextOptionsBuilder<MyContext>()
              .UseSqlite(connection);

var context = new FuturamaContext(builder.Options);
context.Database.EnsureCreated();
```

Best practices

Wrap in logical units/service classes/repositories

Don’t test built-in code…

Program to interface

Demo