Loose Coupling & Isolation

IoC and DIP

Isolation Is Key

Isolation Allows Us To …

- Test concerns separately
- Avoid fragile tests
- Avoid context sensitivity
- Keep tests repeatable
- **Have a better design**

Decoupling Allows Us To

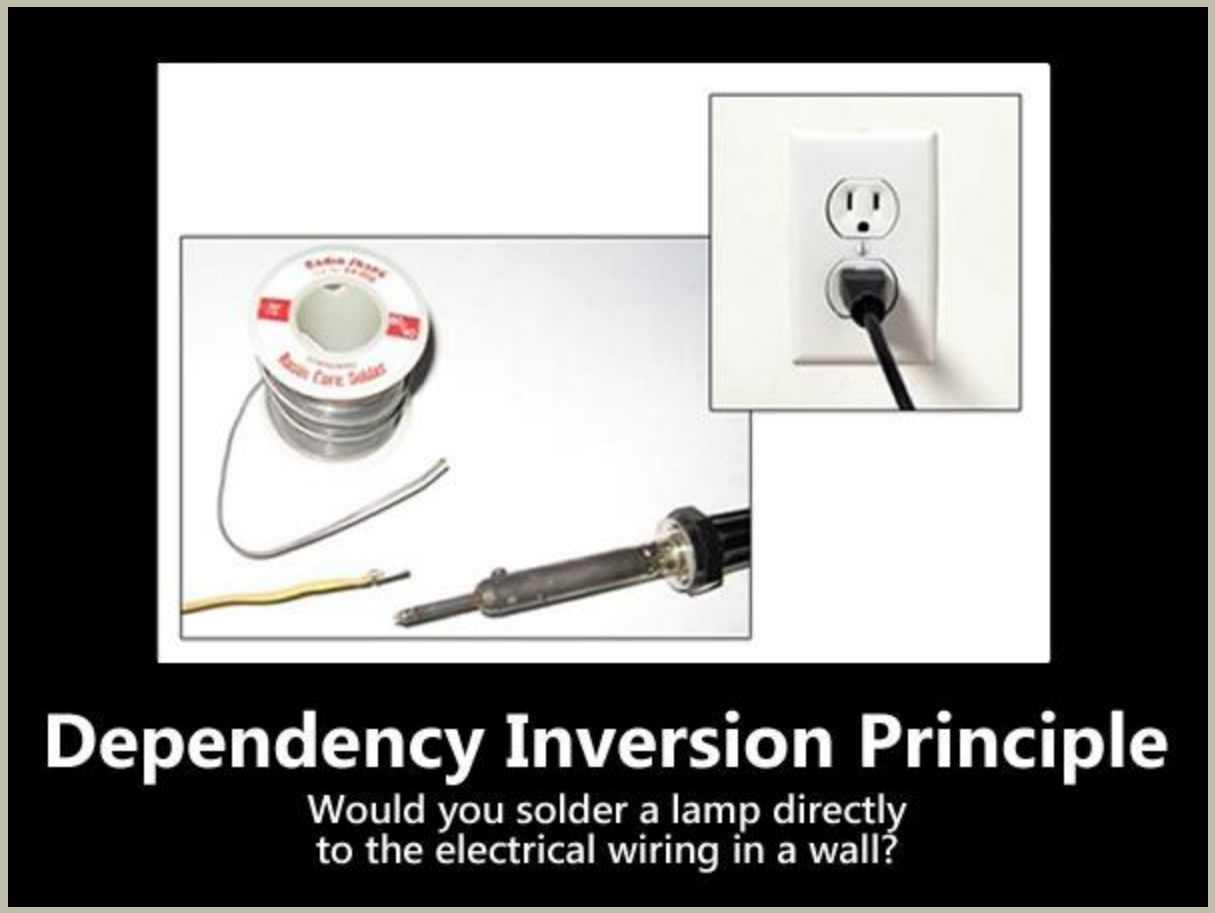
- Be more flexible
- Maintainable
- Testable
- Agile

Set at compile time

```
public class Payroll
{
    public void IssuePaycheck()
    {
        var emailServer = new EmailSender();
        emailServer.SendEmail("...");
        // ....
    }
}
```

Dependency Inversion Principle

- High level modules shouldn't depend on low level modules



Inversion of Control

- Control dependency creation
- Control the executable flow

```
public abstract class Payroll
{
    public void Process()
    {
        GetSalariedEmployees();
        SendPaycheck();
        SendEmail();
    }

    protected abstract void SendPaycheck();
    protected abstract void SendEmail();
    protected abstract void GetSalariedEmployees();
}
```

```
public class Payroll
{
    private IEmailSender _emailSender;

      blic void IssuePaycheck()

        _emailSender.SendEmail("");
        // ....
```

Dependency Injection

- Achieves dependency inversion
- Can also use service locator, factories, virtual methods

```
public class Payroll
{
    public Payroll(IEmailSender emailSender)
    {
        _emailSender = emailSender;
    }

    private IEmailSender _emailSender;
}
```

Inversion of Control Container

- **Enables inversion of control through dependency injection**
  - StructureMap
  - Ninject
  - Unity
  - MEF (to an extent)

```
[Export(typeof(IEmailSender))]
public class EmailSender : IEmailSender
{
    // ...
}
```

Dependency Injection

- **Pros**
  - Loosely coupled components
  - Testability
  - Unobtrusive
- **Cons**
  - Harder to see at compile time what components are in play

Test Doubles

- Like stunt doubles, they substitute for “the real thing”

Stubs and Fakes

- Stub will do the least amount of work possible
- A fake will provide a complete implementation of something you can't use in production

```
public class StubEmailSender : IEmailSender
{
    public void SendMessage(string message) {}
}
```

Spies

- Make it easy to inspect side-effects and component interactions

```
public class StubEmailSender : IEmailSender
{
    public void SendMessage(string message)
    {
        Messages.Add(message);
    }

    public IList<string> Messages = new List<string>();
}
```

Mocks

- Sophisticated and dynamic
- Can be a fake, stub, or spy
- Generally created using a framework

```
var mockSender = new Mock<IEmailSender>();
mockSender.Setup(m => m.SendMessage(It.IsAny<string>()));

mockSender.VerifyAll();
```

Summary

- A loosely coupled design is a testable design
- A testable design is a good design