

1 Redegør for ideerne bag komponentbaseret programudvikling, og tilhørende designprincipper

2 Redegør for ideerne bag komponentbaseret programudvikling og redegør for udvikling og brug af DLL-filer i C++ samt forklar hvorledes men kan udvikle og anvende plugins i C++.

3 Redegør for COM's arkitektur og terminologi samt forklar hvorledes men kan udvikle COM komponenter med brug af ATL.

4 Redegør for .Nets komponentmodel og Lifecycle Management. Samt forklar hvorledes man kan udvikle og anvende komponenter i C#.

5 Redegør for begrebet dependency injection og brugen af IoC-containere, samt interface baseret programmering.

5.1 Injection teknikker

Dependency Injection dækker over teknikker der anvendes til, at indsætte afhængigheder til andre klasser og funktioner.

Mest enkelte teknikker:

Listing 5.1: Constructor Injection

```
1 public class Foo
2 {
3     private Bar _bar;
4     Foo(Bar bar) //Constructor Injection
5     {
6         _bar = bar;
7     }
8
9     public Bar BarClass //Property Injection
10    {
11        set { _bar = value; }
12        get { return _bar; }
13    }
14 }
```

Fordelen ved *Construction Injection* er, at dette sættes så snart klassen oprettes og der fungerer i klassen ikke kan kaldes og bruge en reference = *null*.

Property Injection er også en god idé, hvis man på runtime har brug for at udskifte sin afhængighed, men skal sættes som det første, da referencen ellers er *null*.

5.2 Interfaces

Når man laver afhængigheder til andre klasser, binder man som regel til selve klassen. Problemet med dette er, at det bliver svært at teste, da man skal bruge den konkrete afhængighed i testen, hvilket ikke er ønsket i *unit tests*.

For at komme dette til livs laver man et interface for klassen - også selvom der kun skal være én der arver derfra.

Listing 5.2: Interfaces til injection

```
1 public interface IBar
2 {}
3
4 public class Bar : IBar
5 {}
6
7 public class Foo
8 {
9     private IBar _bar; //IBar
10    Foo(IBar bar) //IBar
11    {
12        _bar = bar;
13    }
14
15    public IBar BarClass //IBar
16    {
17        set { _bar = value; }
18        get { return _bar; }
19    }
20 }
```

På denne måde kan der testes med et framework, hvor *IBar* kan stubbes/mockes af som det behager. Ligeledes kan man bruge *public IBar BarClass* til alle typer klasser, der arver fra *IBar*.

5.3 IoC-Container

Inversion of Control, IoC-Containere, er det, der holder styr på klassernes afhængigheder. Det vil sige, at den ved klassen *Foo* skal have en klasse af type *IBar*.

Der findes flere IoC-Container som alle i bund og grund kan det samme, men hvis implementering varierer en smule.

Listing 5.3: IoC-Container registrering

```
1 public class RegistrationModule
2 {
3     private IUnityContainer _container;
4     public RegistrationModule(IUnityContainer container)
5     {
6         _container = container;
7     }
8
9     public void Initialize() //Implementering fra IUnityContainer
10    {
11        _container.RegisterType<IBar, Bar>();
12        _container.RegisterType<Foo>();
13    }
14 }
15
16 public class Test
17 {
18     private IUnityContainer _container;
19     private Foo _foo;
20     // ...
21     public void TestFunction()
22     {
23         var foo = _container.Resolve<Foo>();
24         foo.Write();
25     }
26 }
```

Det foregår ved, at man registrerer de enkelte implementering

6 Giv et overblik over Microsofts forskellige Extensibility Frameworks, og redegør for den grundlæggende arkitektur og begreber i MEF og PRISM.

7 Redegør for begrebet "Interoperability" generelt, og redegør for brugen af PInvoke samt interoperability mellem COM og .Net.

8 Redegør for problemer og muligheder for Cross Platform Development på .Net platformen.

9 Redegør for hvorledes man designer og implementerer Windows RT komponenter.

10 Redegør for problemstillingen omkring komponenter og flertrådede programmer, samt redegør for hvilke faciliteter .Net og C# giver programudvikleren.

11 Redegør for begrebet "Services", og redegør for hvorledes man designer og implementerer Windowsservices ved brug af .Net og C#.