# ITKPU: Læringsmål

* Redegøre for principperne for komponentbaseret programudvikling

designe, implementere og anvende DDL-filer og plugins i C++.

* Redegøre for COM's arkitektur og terminologi
* Redegøre for .NETs komponentmodel samt kunne designe, implementere og anvende .Net komponenter ved brug af C#.
* Redegøre for anvendelsen af framework til dependency injection.
* Redegøre for begrebet "Interoperability" og kunne designe for og anvende interoperability mellem unmanaged og managed kode.
* Redegøre for Cross Platform Development på .Net platformen, og kunne designe for og anvende Cross Platform Development.
* Designe og implementere Windows RT Compoments
* Designe og implementere Windowsservices ved brug af .Net og C#.

Indhold

[ITKPU: Læringsmål 1](#_Toc42715992)

[Spørgsmål 1: Komponentbaseret programudvikling, designprincipper 3](#_Toc42715993)

[Redegør for ideerne bag komponentbaseret programudvikling: 3](#_Toc42715994)

[Redegør for tilhørende designprincipper: 5](#_Toc42715995)

[Komponent specifikke design principper 5](#_Toc42715996)

[Spørgsmål 2: Komponentbaseret programudvikling, DLL i C++, vis eksempel plugins C++ 7](#_Toc42715997)

[Redegør for ideerne bag komponentbaseret programudvikling: 7](#_Toc42715998)

[Redegør for udvikling og brug af DLL-filer i C++: 8](#_Toc42715999)

[Vis og forklar hvorledes man kan udvikle og anvende plugins i C++: 8](#_Toc42716000)

[Spørgsmål 3: .Net komponentmodel, Lifecycle Management, Components C#, Eksempel 9](#_Toc42716001)

[Redegør for .Nets komponentmodel og Lifecycle Management: 9](#_Toc42716002)

[Vis og forklar hvorledes man kan udvikle og anvende komponenter i C#: 10](#_Toc42716003)

[Spørgsmål 4: Dependency Injection, IoC-containere, typer DI +/ Extensibility Frameworks, MEF. Eksempel MEF/Unity 11](#_Toc42716004)

[Redegør for begrebet dependency injection: 11](#_Toc42716005)

[Redegør for brugen af IoC containere: 11](#_Toc42716006)

[Giv et overblik over Microsofts forskellige dependency injectons og/eller Extensibility Frameworks: 11](#_Toc42716007)

[Redegør for de grundlæggende begreber i MEF: Managed extensibility framework 12](#_Toc42716008)

[Vis et eksempel på brug af MEF eller Unity: 12](#_Toc42716009)

[Spørgsmål 5: Prism, Pros & Cons, Prism Eksempel 13](#_Toc42716010)

[Redegør for den grundlæggende arkitektur og begreber i Prism: 13](#_Toc42716011)

[Hvilke fordele og ulemper er der ved at bruge Prism: 13](#_Toc42716012)

[Vis et eksempel på brug af Prism: 13](#_Toc42716013)

[Spørgsmål 6: Interoperability, PInvoke, Interoperability COM og .Net, interoperability eksempel 14](#_Toc42716014)

[Redegør for begrebet ”Interoperability” generelt: 14](#_Toc42716015)

[Redegør for brugen af PInvoke samt interoperability mellem COM og .Net 14](#_Toc42716016)

[Vis eksempler på interoperability: 14](#_Toc42716017)

[Spørgsmål 7: Cross Platform .Net, Cross platform component eksempel, Windows RT components. 15](#_Toc42716018)

[Redegør for problemer og muligheder for cross-platform developement på .Net platformen: 15](#_Toc42716019)

[Vis et eksempel på en cross-platform component: 15](#_Toc42716020)

[Redegør for hvorledes man designer og implementerer Windows RT components: 15](#_Toc42716021)

[Spørgsmål 8: Cross platform Android + iOS, Xamarin + platformer, Pros & Cons Xamarin, Xamarin eksempel. 16](#_Toc42716022)

[Giv et overblik over de forskellige muligheder for cross-platform udvikling for Android og iOS enheder: 16](#_Toc42716023)

[Redegør for Xamarins arkitektur og virkemåde for de forskellige understøttede platforme: 16](#_Toc42716024)

[Redegør for både fordele og udfordringer ved brug af Xamarin til App udvikling: 16](#_Toc42716025)

[Vis et eksempel på brug af Xamarin: 17](#_Toc42716026)

[Spørgsmål 9: Services, Windowsservice .Net C#, Windowsservice eksempel 18](#_Toc42716027)

[Redegør for begrebet ”Services”: 18](#_Toc42716028)

[Redegør for hvorledes man designer og implementerer en Windows Service ved brug af .Net og C#: 18](#_Toc42716029)

[Vis et eksempel på en Windows Service: 18](#_Toc42716030)

[Spørgsmål 10: Components in Clientside web, UI-components JS, Web Component eksempel (u. framework, evt. LitElement) 19](#_Toc42716031)

[Redegør for brugen af komponenter ved client side webudvikling: 19](#_Toc42716032)

[Hvad skal browseren understøtte for at man kan lave UI-komponenter, som er uafhængige af et JS framework: 19](#_Toc42716033)

[Vis et eksempel på en web komponent, som er udviklet uden brug af framework eller evt. ved brug af LitElement: 19](#_Toc42716034)

ITKPU Eksamensdispositioner:

## Spørgsmål 1: Komponentbaseret programudvikling, designprincipper

### Redegør for ideerne bag komponentbaseret programudvikling:

Formål:

Oprindeligt C++, mere moderne anvendt i dag .Net

Hvad er det:

Lille stykke software der udfører et stykke arbejde. Kan skiftes ud med et andet komponent givet at interfacet til det er det samme.

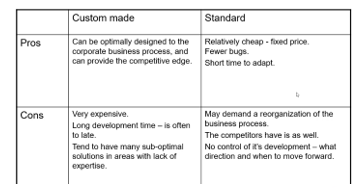
Fordel: du samler en masse små komponenter til et samlet produkt. Men du laver det ikke fra bunden selv.

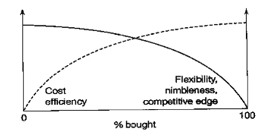
Binær fil der udfører et arbejde som kan inddrages i ens produkt. Det er afhængigt af teknologien omkring komponentet. Man kan dog ”translate” et komponent til en anden platform – koster + ikke ideelt.

Genbrug = -(Udviklingstid + omkostninger).

KPU har større mængde genbrug af kode. Godt.

Komponent skal være nemt at anvende.

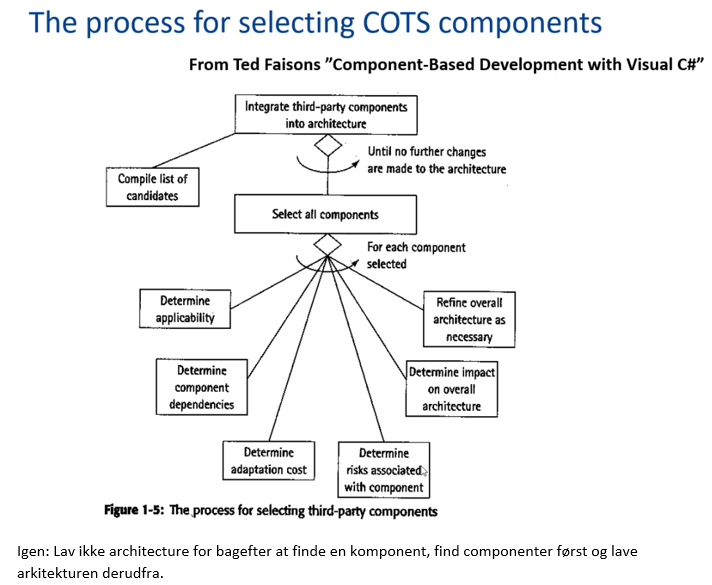




Component based developement (CBD) shares methods with OOP

CBD also addresses: Packaging, Deployment, Licensing and potentially Security.

OOP runs on devs machine. CBD runs on end users machine.



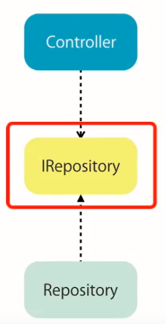
### Redegør for tilhørende designprincipper:

Agile design:

Konstant udvikling af design og arkitektur.

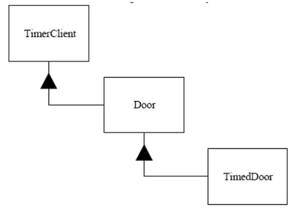
Dårlige designs er bestemt af;

* **Rigidity** Difficult to change
* **Fragility** Easy to break
* **Immobility** Difficult to reuse
* **Viscosity** Difficult to do the right thing
* **Needless complexity** Overdesign
* **Needless repetition** Too much cp + paste
* **Opacity** Difficult to read and navigate the code

Dependency Inversion Principle ((DIP) - D form SOLID)

High level objects should not depend on low level implementations and both should depend on abstractions.

Interface Segregation Principle ((ISP) – I from SOLID)



Nedarvning

Undgår interface pollution

Thumb rule: interface 1-12 medlemmer og kan strækkes til 20: men ALDRIG over 20.

### Komponent specifikke design principper

**Granularity:** Et vigtigt spørgsmål er om jeg skal lave mange små komponenter eller skal jeg lave en enkelt stor komponent?

|  |  |
| --- | --- |
| REP: **R**euse/**R**elease **E**quivalency **P**rinciple | *Man må ikke lave sine pakker mindre end der er et system til at holde styr på det.* |
| CRM: **C**ommon **R**euse **P**rinciple | Alle klasser i en komponent skal bruges sammen. Hvis en klasse ikke bruges, skal den klasse måske ikke være en del af den komponent. |
| CCP: **C**ommon **C**losure **P**rinciple | (Single Responsibility for components). Denne Princip gør Release og Reevaluate processen nemmere. *Ting der skal ændres sammen skal nok også pakkes sammen* |

**Stability**: Hvordan gør vi en komponent stabilt?

|  |  |
| --- | --- |
| Asyclic Dependency Principle | Der må ikke være nogen cyklisk afhængighed. Hvis 2 pakker kræver hinanden, så skal de pakker laves som en enkelt komponent *(det kan ikke lade sig gøre på assembler niveau eller i c# men kan godt i cpp).* |
| Stable Dependencies Principle | En komponent burde kun være afhængig af ting der ændres i mindre en komponent der bruger det.  En stabil komponent er uafhængig. En ustabil komponent har op til flere afhængigheder. |
| Stable-Abstraction Principle | En pakke skal være ligeså abstrakt som den er ikke abstrakt. Måles ved at se på hvor mange afhængigheder du har udenfor i forhold til indenfor pakken.    Ved 1,1 haves der en masse abstraktioner uden at der er nogen der bruger dem -> useless. |

## Spørgsmål 2: Komponentbaseret programudvikling, DLL i C++, vis eksempel plugins C++

### Redegør for ideerne bag komponentbaseret programudvikling:

Formål:

Oprindeligt C++, mere moderne anvendt i dag .Net

Hvad er en komponent:

Lille stykke software der udfører et stykke arbejde. Kan skiftes ud med en anden komponent giver at interfacet til det er det samme.

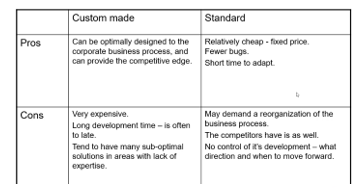
Fordel: du samler en masse små komponenter til et samlet produkt. Men du laver det ikke fra bunden selv.

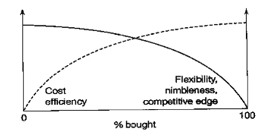
Binær fil der udfører et arbejde som kan inddrages i ens produkt. Det er afhængigt af teknologien omkring den enkelte komponent. Man kan dog ”translate” en komponent til en anden platform – koster + ikke ideelt.

Genbrug = -(Udviklingstid + omkostninger).

KPU har større mængde genbrug af kode. Godt.

Komponent skal være nemt at anvende.



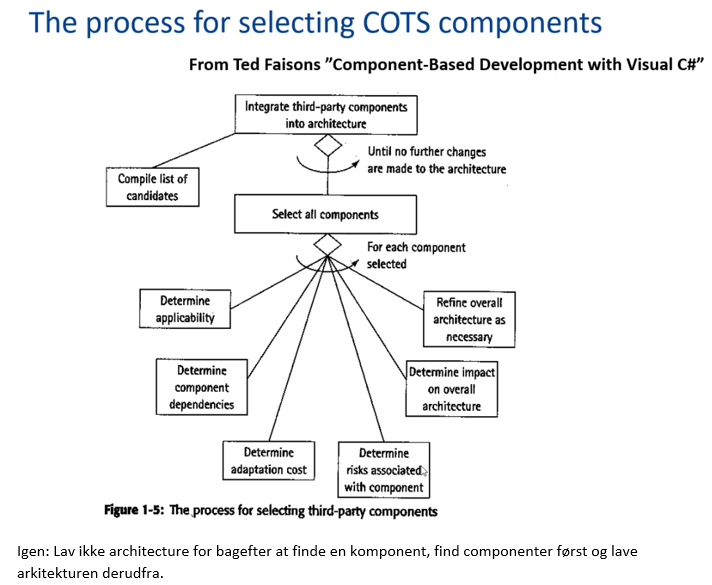


Component based development (CBD) shares methods with OOP

CBD also addresses: Packaging, Deployment, Licensing and potentially Security.

OOP runs on devs machine. CBD runs on end users machine.

COMMERCIAL FS



### Redegør for udvikling og brug af DLL-filer i C++:

Filtype der kan skabes i sammenhæng med en EXE fil.

Meningen med dem er at kunne genbruge kode uden at skulle kopi paste, ændre i koden, men blot ved at vedlægge dll filen.

DLL located once i fysisk memory doesn’t need recompile – as long as the interface for the DLL remains the same.

Types:

**Traditional DLL:** By windows OS and Custom DLLs by developers.

**COM based DLL:** Which always return the same 5 functions.

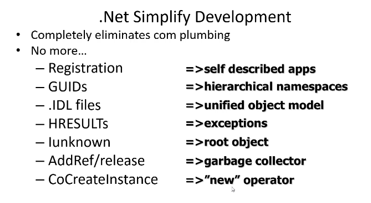
**.Net DLL:** Different structure and code is in (common intermediate language) IL format rather than code.

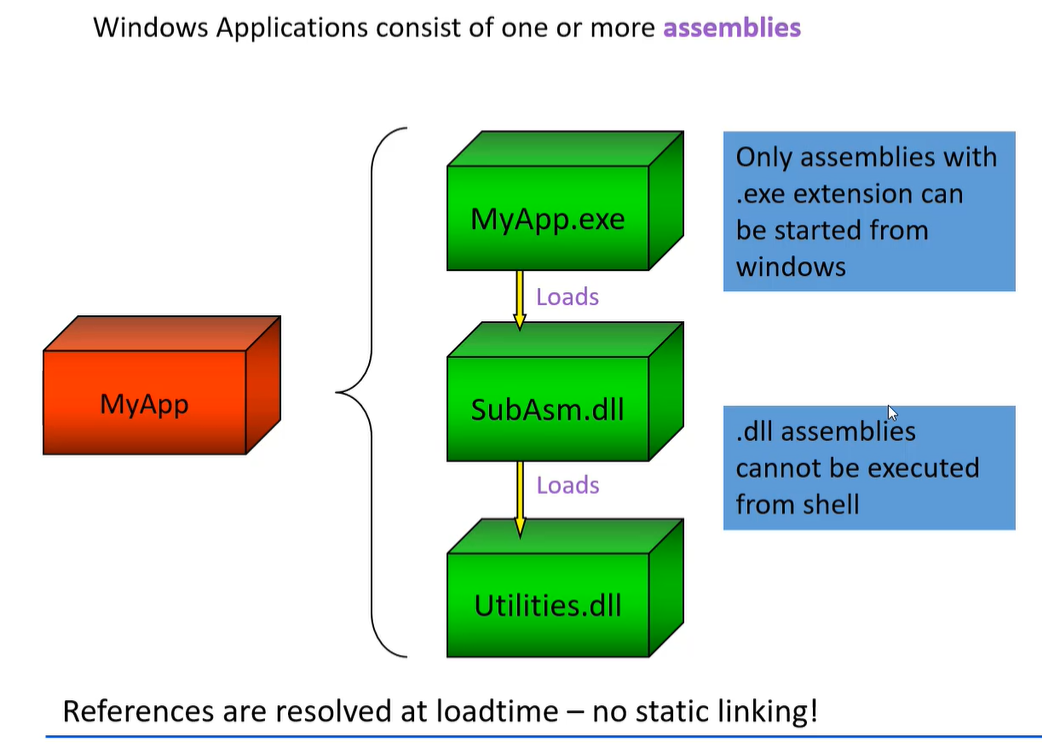
### Vis og forklar hvorledes man kan udvikle og anvende plugins i C++:

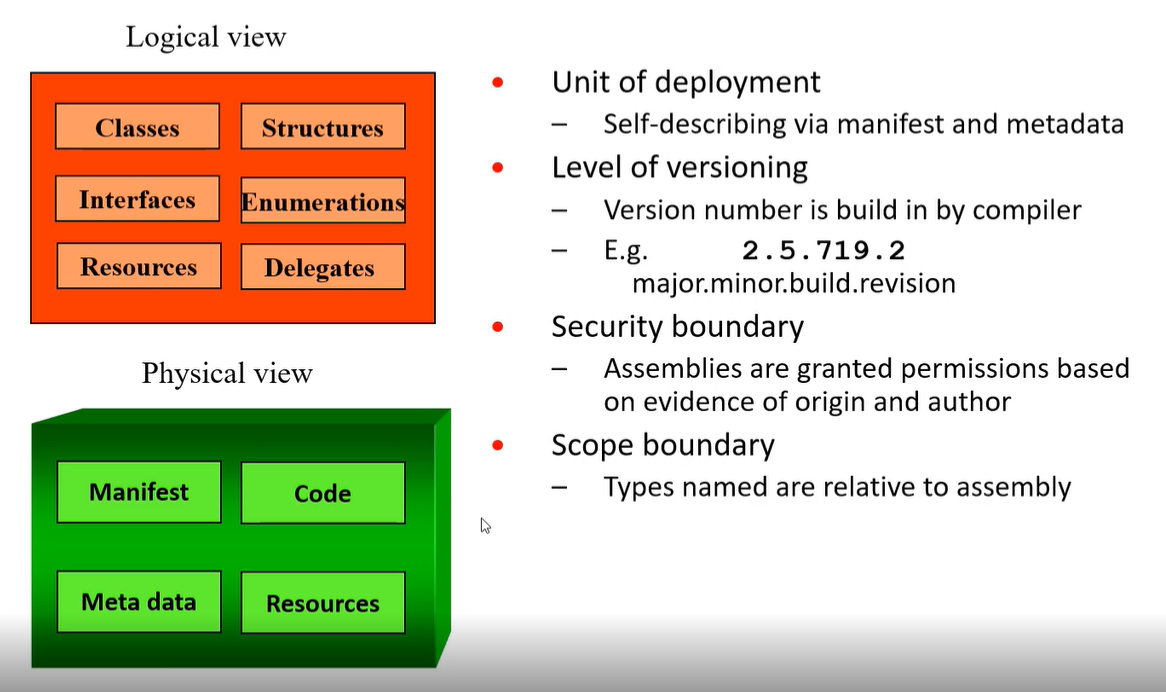
## Spørgsmål 3: .Net komponentmodel, Lifecycle Management, Components C#, Eksempel

### Redegør for .Nets komponentmodel og Lifecycle Management:

.Net introducerede CLR’en hvilken fjernede behovet for ”plumbing” ved at vælge at tvinge andre til at tillægge sig deres model.

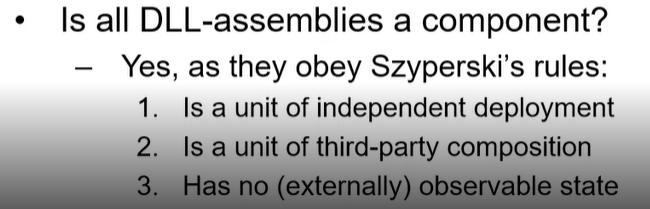






Assemblies bruges til at gruppere typer fysisk – namespaces er logisk.

### Vis og forklar hvorledes man kan udvikle og anvende komponenter i C#:

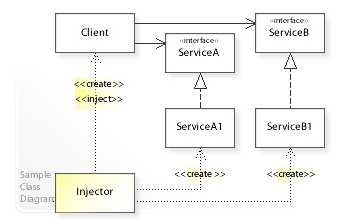


## Spørgsmål 4: Dependency Injection, IoC-containere, typer DI +/ Extensibility Frameworks, MEF. Eksempel MEF/Unity

### Redegør for begrebet dependency injection:

Når et objekt modtager andre Objekter som clienten depender på. De passede objekter hedder services. Services er også omtalt som dependencies og har typisk et ”using” forhold til fra clienten til en service.

Meningen med dette er at holde en lav kobling i sit system. Nemt at ændre /udvide senere.



### Redegør for brugen af IoC containere:

IoC = dependency injection framework – automatisere processen for dig.

En container er ansvarlig for objekt håndtering

* Creation
* Destruction
* Lifetime
* Conf
* Dependencies

En container tillader at objekter kan konfigureres af containeren frem for af applikationen.

### Giv et overblik over Microsofts forskellige dependency injectons og/eller Extensibility Frameworks:

Der findes mange depencendy injection frameworks.

* .Net
  + Unity
  + MEF
  + Spring
  + Ninject
* Java
  + Spring DI
  + Google Guide

### Redegør for de grundlæggende begreber i MEF: Managed extensibility framework

MEF er et library i .NET 4.0 -> bedre reuse applications + components

Dynamisk composering -> composering sker run time – opførsel forskellig baseret på konfiguration.

Extensibility = systematisk måling af muligheden for at udvide et system + hvor stor indsats dette tager.

Open/closed principle er gældende. (open for extension closed for modification)

Fordele ved MEF: dyb object hierarki af komponenter

Abstracte komponenter

Tillader en lidt mere doven instanstiering af komponenter

Tilføjer en katalog mekanisme med rig metadata for komponenter for at tillade dynamisk opdagelse af disse.

1. Contract 2. Import 3. Export 4. Catalog 5. Composition Container

### Vis et eksempel på brug af MEF eller Unity:

## Spørgsmål 5: Prism, Pros & Cons, Prism Eksempel

### Redegør for den grundlæggende arkitektur og begreber i Prism:

Prism er et framework, der anvendes til at bygge løst koblet, vedligeholdsmulige, testbare XAML applikationer i WPF og Xamarin Forms.

Selv tidligere anvendt til at bygge MVVM struktur i WPF i I4GUI kurset.

Spiller godt sammen med Unity of MEF med dependency injection

|  |  |
| --- | --- |
| **Shell** | Overall layout for systemet |
| **Module** | When a view and view model are together (can have multiple views)  The goal is to create modules as stand-alone solutions, which can be reused across the application |
| **Module Catalog** | Anvendt til at vide hvilke modules der skal loades og rækkefølge |
| **Regions** | Can be made in the shell or view. This Is used to define where to load views.  The regions are controlled by the region manager. |
| **Navigation** | State based navigation og view-switching navigation. I state based opdateres et eksisterende view og i view-switching bliver nye views skabt og erstatter de gamle i app UI. |
| **Event Aggregator** | Publish subscribe mekanisme anvendt for at undgå tæt kobling. Ofte anvendt for at tillade forskellige moduler at kommunikere. |
| **Dependency Injection** | Use Either MEF or Unity to handle the dependency injection. |
| **Services** | Same as web service: non-UI functionality such as db access. Logging etc. Should be used with the DI containers. |
| **Controllers** | Control what to show in the application (view switching nav) |
| **Bootstrapper** | Deprecated: done by the PrismApp class Responsible for Initializing the app |

### Hvilke fordele og ulemper er der ved at bruge Prism:

Fordele: Jo større system jo bedre med Prism. Stejl indlæringskurve. Løs kobling gennem hele systemet. Moduler kan skabes og testes individuelt.

Ulemper: Prism addresserer ikke følgende emner;

Connectivity, Data synchronization, Service og message infrastruktur design, Auth + Auth, Performance, versioning, efterfulgt af error handling og fault tolerance.

Key decisions? – MEF / Unity? Choose application-specific services for system ? Logging? Module discovering?

### Vis et eksempel på brug af Prism:

## Spørgsmål 6: Interoperability, PInvoke, Interoperability COM og .Net, interoperability eksempel

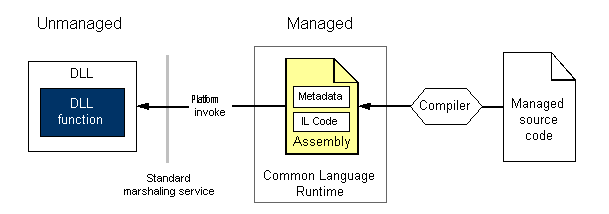
### Redegør for begrebet ”Interoperability” generelt:

Interoperability er når man kan operere på tværs af to platformer

COM og .NET følger to helt forskellige arkitekturer. Derfor behøves en metode til at kunne tillade anvendelsen af komponenter fra det andet elements applikationer.

### Redegør for brugen af PInvoke samt interoperability mellem COM og .Net

Platform Invoke er en service der tillader håndteret kode at kalde ikke håndterede funktioner, som er implementeret i DLL’er. PInvoke støtter sig til metadata for at lokalisere eksporterede funktioner og marshaller deres arguments på runtime. Når PInvoke kaldes udfolder sig følgende scenarie (4trin):



1. lokaliserer DLL’et der indeholder funktionen.
2. Loader DLL’et ind i memoryen.
3. lokaliserer addressen på funktionen i det allokerede memory fra trin 2 og skubber funktionens arguments ud på stacken.
4. Overfører kontrollen til den ikke managede funktion.

PInvoke kaster exceptions genereret af den ikke managede funktion til den managede ”caller”.

### Vis eksempler på interoperability:

## Spørgsmål 7: Cross Platform .Net, Cross platform component eksempel, Windows RT components.

### Redegør for problemer og muligheder for cross-platform developement på .Net platformen:

Det mest relevante problem for Cross-platform dev, er at vi har så mange forskellige Operativ systemer i dag. Windows, Linux, MacOS, iOS, Android osv. For at udvikle et system til den ene platform virker ikke nødvendigvis på den anden. Dette giver anledning til udbud for et godt cross platform Development tool.

|  |  |
| --- | --- |
| Electron | Use JavaScript to create a powerful web application using Vue, Angular or react. Encapsulate the app using electron to port out to Mac, Windows, web and with some complexity Android and iOS. |
| Qt, Qtk+ | Cpp: Probably the C++ based Qt widget toolkit is the most well-established way of writing desktop cross platform applications |
| Xamarin(mono) | C# to target Android, iOS, Linux, Windows etc. The mono framework is made to target mobile and desktop apps and support the standard MVC design pattern. |
| React-Native and Ionic | Use JavaScript to build native components (react native) and ionic does a similar thing just mainly for angular. |

### Vis et eksempel på en cross-platform component:

### Redegør for hvorledes man designer og implementerer Windows RT components:

Et RT komponent er et COM komponent med .NET type system

RT komponenter er kendt for at kunne anvendes af mange forskellige kodesprog: JS, C/Cpp, C# og VB.

C/Cpp producerer unmanaged komponenter hvor C# og VB producerer managede komponenter.

RT VS. Class Library –

RT komponenter kan anvendes af mange platformer

Tilgengæld er der restriktioner på public interface

Class library har så ingen restriktioner på public interface, men kan kun anvendes af platformene C# og VB.

4 Regler når man arbejder med disse;

Et interface skal have defineret hvilke metoder der er. Her er det vigtigt at anvende en type der er supported af widows RT systemet. Dette gælder for methods properties og events.

Der må ikke indgå polymorfi og disse skal forsejles.

Root namespace skal være ens med assembly navnet.

Og sidste – når der anvendes et managed RT komponent inde i managed kode, skal der ikke tages nogle specielle handlinger.

## Spørgsmål 8: Cross platform Android + iOS, Xamarin + platformer, Pros & Cons Xamarin, Xamarin eksempel.

### Giv et overblik over de forskellige muligheder for cross-platform udvikling for Android og iOS enheder:

Tid = penge -> derfor cross platform udvikling i stedet for at udvikle noget flere gange.

Mange forskellige værktøjer til dette i dag. Native apps (Jave / Swift) and hybrid native apps for iOS/android. Og yderligere hybrid apps for web, desktops og mobile devices, som medfølger med sine egne problematikker samt grader af sværhed.

|  |  |
| --- | --- |
| React Native | Primært anvendt til iOS/Android CP, kan udvides til web og desktop. Baseret på React.js meget sammenligneligt med struktur opbygget af komponenter. HTML udskiftet med native komponenter. |
| Apache Cordova | Apache Cordova er et brugbart lille tool til at bygge Web, iOS og android mm. Applikationer. Her anvendes HTML, JS og CSS. |
| Ionic | Ligesom React Native er dette et framwork til web miljøer der kan kompilere resultatet over på iOS/android applikationer. Baseret på React. |
| Flutter | Flutter er endnu en kandidat til multi cross platform. |
| Standard Progressive web apps | Applikation der kan installers – kræver en service worker (offline capabilites). Formålet med progressive web apps er at give brugeren en bedre UE (userEXP) som var det på en mobile app. Krav:   * Kører på Https * Har et web app Manifest inkluderet * Skal implementere en service worker |

### Redegør for Xamarins arkitektur og virkemåde for de forskellige understøttede platforme:

Xama iOS --> AoT (Ahead of time) kompilering. Android bruger JiT (just in time) kompilering. For at kompilere til iOS kræver dette en Mac…

Xamarin på iOS lediggører bindings som oversættes direjte til native iOS bindings gennem Marshalling.

Xamarin på android lediggører bindings til native android API’s med MCW’en (mono content wrapper).

Når ligeså snart Android har lavet lidt arbejde, binder ACW (android content wrapper) resultatet til MCW’en i runtime. Alt dette bliver håndteret af Xamarin og fra vores perspektiv ser vi kun C# kode.

### Redegør for både fordele og udfordringer ved brug af Xamarin til App udvikling:

|  |  |
| --- | --- |
| Pros | Cons |
| Med Xamarin Forms, nemmere end før at bygge apps med shared UI kode. Med meget begrænset mængde native kode påkrævet. | Kræver en licens til deres Enterprise version. |
| Kan kombineres med andre frameworks (Prism) | Ikke så god performance og ingen support til apps med tung grafisk byrde. |
|  |  |
| Forms er hurtigt til prototype arbejde. | Stor app filstørrelse. |
|  | Delay på 2uger+ for android / iOS opdateringer. |
|  | Begrænset mængde open source libs – begrænset til .Nets pakker. |

### Vis et eksempel på brug af Xamarin:

## Spørgsmål 9: Services, Windowsservice .Net C#, Windowsservice eksempel

### Redegør for begrebet ”Services”:

En service er en app, der ikke har et mål af sig selv, i stedet bliver denne kaldt og anvendt af andre komponenter.

En service er somregel mest Backend orienteret. Den har et interface som er kendt til af resten af systemet og en klasse der implementerer selve servicen.

Typer af services:

|  |  |
| --- | --- |
| Remote Service | Et godt eksempel på en microservice som er tilgængelig af et endpoint. |
| Lokal Service (windows service) | * **Kører udenfor nogen user context** * **Kører I sit eget vindue:** Even if they are started by the user. If that session breaks, only that goes down and not the user session and vice versa. * **Har egne sikkerheds settings.** * **Kører uden menneskelig indblanding** * **Kan starts af**: Windows på bootup, Bruger der anvender kontrolpanel eller af en applikation. |
| Component Service | Low level services used by components |

### Redegør for hvorledes man designer og implementerer en Windows Service ved brug af .Net og C#:

En windows service kan oprettes med en Konsol applikation eller ved brug af default service builder i visual studio.

Ligesom andre typer applikationer findes der mange forskellige lifecycle kald for services – OnStart stop, pause osvosv. Når disse kaldes skal de returnere indenfor 30 sekunder.

### Vis et eksempel på en Windows Service:

## Spørgsmål 10: Components in Clientside web, UI-components JS, Web Component eksempel (u. framework, evt. LitElement)

### Redegør for brugen af komponenter ved client side webudvikling:

Frameworks der anvender komponentbaseret model.

React -> Vue, Angular, React -> Formålet for en komponent er indholde og rendere et stykke af UI’en samt noget logik.

Tidligere på kurset: Native komponent -> any framework.

* Properties: Defined by the DOM
* Attributes: HTML specific. (attributes can be custom whereas props are pretty much set.

4 lifecycle hooks:

The constructor method: init code – minus DOM

The connectedCallback method: called when comp. added to DOM, code here init UI, like setting component innerHTML

The disconnected callback method: cleanup method

The adoptedCallback method: called when comp. moved to diff frame. Seldom needed.

### Hvad skal browseren understøtte for at man kan lave UI-komponenter, som er uafhængige af et JS framework:

### Vis et eksempel på en web komponent, som er udviklet uden brug af framework eller evt. ved brug af LitElement:

Vs code – 02-lit-html-demo polymer serve