**Relevant at vise kode i alle spørgsmål udover 1. spørgsmål**

**En eller flere af øvelserne kan inddrages**

**Viser man på egen pc? Vise kodestykker som er store?**

**Stuff to figure out:**

**Strong name Public/Private key**

**Background workstation GC**

**Background server GC**

**Unmanaged resources**

**IDisposable**

**Product Line Engineering**

* **Genbrug**
* **Variationspunkter (Open-Closed princip)**
* **Kræver meget at holde styr på dokumenter**
  + **Godt ved firmaer med meget overhead (f.eks. Terma)**

1. **Redegør for ideerne bag komponentbaseret programudvikling, og tilhørende designprincipper.**

**Hvad er en komponent?**

* **Del af en helhed**
* **Kan erstattes (behøver ikke være helt ens)**
  + **Skal blot overholde et interface (eks. en pære, harddisk)**

**Hvorfor komponenter?**

* **Spare penge!**
  + **Reducer development omkostninger**
  + **Reducer ”*time to market”***
* **Genbrug**
* **Ikke ved det hele så det er smart at bruge noget der er lavet**
* **Flere kan arbejde samtidigt** 
  + **God adskillelse mellem komponenter**

**Software komponent**

* **Uafhængig deployment**
* **Third-party komposition**
* **En enhed på run-time**
  + **.exe, .dll/ocx, tabel, fil (config/html)**
* **Krav**
  + **Skal overholde et interface!**
  + **Passer ind i en bestemt komponentarkitektur**
    - **Kan ikke bruge komponenter fra andre teknologier (eks. .Net til java)**
    - **Kan med interoperability**
* **Anderledes end hardware da man køber et ’blueprint’ i stedet for fysisk enhed**

**Custom-made vs standard**

* **Custom**
  + **Pro: Optimalt design, fordel over konkurrenter**
  + **Con: Lang tid at lave, dyrere**
* **Standard**
  + **Pro: Billigere, hurtigere**
  + **Con: Bruge tid på at forstå, kan kræve omstrukturering**
* **Mixed: Optimalt cost/benefit!**

**Designprincipper**

* **DIP: High-level moduler skal ikke være afhængig af konkrete low-level**
  + **Fix: Interfaces**
* **REP (Reuse/Release Equivalency Principle)**
  + **Tracking system:**
  + **Skal ikke være for småt**
  + **Nogen skal maintain komponenter og notificere**
* **CRP (Common Reuse Principle)**
  + **Man genbruger alle klasser i en komponent**
  + **Klasser der *ikke* genbruges skal ikke i samme komponent**
* **CCP (Common Closure Principle)**
  + **Ændring på en klasse i en komponent ændrer ikke andre komponenter**
  + ***Klasser som muligvis ændrer sig sammen skal være i samme komponent***

**God dependency management**

* **ADP (Acyclic Dependencies Principle)**
  + **Ingen cykliske afhængigheder -> lav til én pakke!**
  + **Kan løses ved DIP eller lave ny pakke som der refereres til**
* **SDP (Stable Dependencies Principle)**
  + **Afhæng af *mere* stabile komponenter (sandsynligvis ikke ændrer sig i forholdt til komponenten selv)**
  + **Mange afhængigheder: sandsynligvis ustabil**
* **SAP (Stable-Abstraction Principle)**
  + **En pakke skal være så abstrakt som den er stabil**
  + **Ustabil pakke: Konkret klasse**
  + **Meget stabil pakke: Maksimal abstrakt**
  + **A-I Graph: Abstraktion vs Ustabillitet** 
    - **Useless: Abstrakt + ingen der bruger!**
    - **Pain: Mange afhængigheder til ustabil + konkret klasse!**
* **Hvor store, hvor mange, hvad hører sammen etc**

**Metrics**

* **Ce = Afhængigheder fra pakken til andre klasser**
* **Ca = Afhængigheder til pakken fra andre klasser**
* **Na = antal af abstrakte klasser**
* **Nc = antal af totale klasser**

1. **Redegør for ideerne bag komponentbaseret programudvikling og redegør for udvikling og brug af DLLfiler i C++ samt vis og forklar hvorledes man kan udvikle og anvende plugins i C++.**

**Mere kodenært end 1. (DLL og plugins i C++)**

**Forklaring af hvad er komponentbaseret programudvikling (meget af spg 1)**

**Hvad er en DLL?**

* ***Compiler* kompilerer source filer -> får objekt filer**
* ***Linker* sammen**
  + **Funktioner bliver til DLL filer**
  + **Main bliver til .exe**
* ***Loader* sætter dem sammen *run time***

**Hvorfor DLL?**

* **Kan bruges af andre programmer**
* **Ligger kun fysisk én gang i RAM men virtuelt flere gange**
* **Mindre linking (ingen rekompilering)**
* **Kan kaldes fra andre programmeringssprog**

**Load time (implicit linking)**

* **Linkeren skal kende .lib filen**
* **Loader kigger efter DLL filer**
  + **DLL skal ligge med .exe eller en mappe refereret i PATH**

**Run time (eksplicit linking)**

* **Ingen .lib eller header fil behøves**
* **Bruges ved plugins**
* **LoadLibrary**
  + **Navn på DLL (eller sti)**
  + **Giver en handle til DLL**
* **GetProcAddress**
  + **Navn på funktion**
  + **Returnerer adressen, gemmes i en funktionspointer**
* **FactoryMethod (CreateDLLObject(), DeleteDLLObject() ..)**

**Name mangling**

* ***dumpbin* kan bruges for at se hvilke funktioner en DLL eksporterer**
* **C++ manipulerer med funktionsnavne**
  + **Skal supportere funktionsoverload**
* **Linkeren er fra C, kender ikke til funktionsoverload**
* **Bruge *extern ”C”* for at fjerne det**

**Lab**

* **Hovedprogrammet definerer interface som DLL skal implementere**
  + **CreateDLLObject, DeleteDLLObject**
* **CDLLclass implementeres af DLL**
* **CAppUtil interface laves af hovedprogrammet (samt konkret klasse)**
  + **Bruges til at DLL kan tilgå og anvende noget data**
* **Plugin DLL!!**

**Find og øv lab øvelse, i egen øvelse, ikke løsningsforslag**

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

**Spørgsmål med lav karakter! Be aware!**

**.Net**

* **Kræver ikke forarbejde som COM!**
  + **Ingen GUIDS, .IDL filer, HResults, IUknown, AddRef/Release, CoCreateInstance..**
  + **I stedet: Exceptions, ’System’, GC, ’new’ …**
* **’Assembly’: .exe eller .dll (skal loades fra .dll eller .exe)**
  + **Kun loadtime/runtime! Ingen statisk linking**
* **Loades ind i en Application Domain**
  + **Kan sandboxes med bestemte permissions**
  + **Permissions: origin-based, content-based evidence**

**.Net komponentmodel:**

* **Assembly som består af 4 dele:**
* **Manifest**
  + - **Strong name (sker ved signering):**

**Filnavn**

**Versionsnummer**

**Culture**

**Public key + signatur**

**Kode hashet**

**Manifest hashet -> token**

**Kommer ind i hashing algoritme med private key**

**Får digital signatur + public key**

**Kan verificere ved at hashe DLL som skal match PK + dig sig**

* + - **Filer i assemblien og refererede assemblies**
* **Metadata (typeinformationer)**
* **Kode: IL eller native**
* **Optionelt: ressourcer**

**GAC (Global Assembly Cache)**

* **Indeholder delte assemblies (skal være strong named)**

**Garbage Collection**

* **Sletter IKKE!**
* **Roots: reference typer (static field, argumenter, lokale variabler, CPU registre..)**
  + **Hver metode har root table (produceret af JIT compiler)**
* **Kompakter heapen (Mark and Compact)**
  + **Kopierer dem der er i brug ned over dem der ikke bruges**
  + **Dem der er i brug ligger tæt sammen**
  + **Generationer (0, 1, 2)**
  + **Gen 0 har f.eks. 256 KB (cache)**
* **’NextObjPtr’ bruges ved allokering**
* **5% CPU tid på GC… Men hurtigere med Background GC**
  + **Background på Gen 2 (0, 1 er hurtige ..)**
* **Problem: eventhandlers mem. leak**
  + **GC følger ikke Weak Refs (ses i Release mode!!)**
  + **PRISM EvetnAggregator bruger Weak R efs**

**Unmanaged code (Ikke GC’ed):**

* **Finalization (’destructor’)**
  + **Kaldes af GC, før GC overskriver**
  + **GC kører**
  + **Finalization Queue**
  + **Freachable Queue** 
    - * + **Finalizer tråd kalder ’Finalize’**
  + **Overlever en gen -> ikke god performance**
* **Bedre: Dispose pattern**
  + **Sætte flag for ’disposed’ for at clean managed res.**
  + **’using’ statement kalder Dispose så finalizer ikke skal**
* **Finalizer (en form for destructor)**
  + **Kører på finalizer tråd**
  + **Holder øje med unmanaged kode**
  + **Overlever en GC pga finalizer**
  + **Når den er færdig bliver den overskrevet**
  + **Ikke hensigtsmæssigt: derfor Dispose Pattern**
    - **Selv aktivt kan kalde dispose i stedet for GC skal kalde finalize**
    - **Husk at kalde GC.SupressFinalization() for at signalere man har ryddet op som gøres i Dispose**
    - **Hvis man har kaldt Dispose eksplicit kan man også kalde Dispose hvis man har nogle managed ressourcer som man har nogle referencer til. Har man eksplicit kaldt Dispose skal man også rydde dem op. Hvis Dispose bliver kaldt fra finalizeren har brugeren glemt at kalde Dispose. Så er det ikke sikkert at der er nogen der har referencer? Pointere kan potentielt være null, derfor må man ikke kalde Dispose på managed ressourcer? Derfor et tjek inde i Dispose.**

1. **Redegør for begrebet dependency injection og brugen af IoC-containere. Giv et overblik over Microsofts forskellige dependency injection og/eller Extensibility Frameworks. Redegør for de grundlæggende begreber i MEF og vis et eksempel på brug af MEF eller Unity**

**Løsning på tæt kobling**

* **Interfaces**
* **Klasse A bruger instans af klasse B**
  + **A instantierer ikke direkte B**
  + **Nemmere at teste**
  + **Muligt at udskifte**
* **Kan bruge Factory**

**Dependency Injection**

* **Constructor, setter injection**

**IoC containers (Inversion of Control)**

* **Håndterer objekt management (lifetime og konfigurering)**
  + **Dependencies!**
* **Sker fx i Controllers i ASP.NET (man new’er ikke selv)**

**Unity (IoC container)**

* **Kan instantiere objekter** 
  + **Begrænset, da den ikke kan håndtere interfaces automatisk, kun konkret**
* **Kan vide typer vha. RegisterType**
* **Transient (ny instans hver gang), Singleton**

**MEF (Managed Extensibility Framework)**

* **Dynamically composed: Understøtte plug inns**
* **Open-Closed principle**

**MEF usage**

* **Contract: ISomething**
* **Klasse: Export(typeof(ISomething)]**
  + **Er en streng (typenavn)**
* **Catalog: forskellige typer (Type, Assembly, Directory, Aggregating)**
  + **AssemblyCatalog (imports/exports i en given assembly)**
* **CompositionContainer: kræver AssemblyCatalog med Exports**
  + **ComposeParts(this): Connects import/export (dep. Injection)**
* **Host: [Import] på sin property** 
  + **Constructor [ImportingConstructor]**
  + **Multiple classes exports: [ImportMany]**
  + **Dep. injecter fra CompContainer**
* **Lifetime ved Export/Import: Shared/NonShared**
* **MEF understøtter <Lazy>**
* **Custom attributes med [MetadataAttribute]**
  + **ExportAttribute**
  + **[ExportAnimal(Eatmeat = true)]**
* **ExportProvider: compose enkelte containere**
* **RegistrationBuilder: kan bruges hvis Export/Import sættes på eksterne klasser**

**MAF**

**Dependency Injection og IoC containere generelt**

**Lidt om forskellige muligheder med microsoft framworket med Unity MAF og MEF. Fordele og ulemper. Gå i kødet med en af dem (MEF – hvad er grundelementerne og hvordan virker det, hvordan bruger man det)**

**Prism**

* **Grafiske brugergrænseflader**

**System.Add-In**

* **Sikkerhed, Dårlig performance**

1. **Redegør for den grundlæggende arkitektur og begreber i Prism. Hvilke fordele og ulemper er der ved at bruge Prism? Vis et eksempel på brug af Prism.**

**Application framework (MVVM) +mere**

* **God til større applikationer (flere views)**
  + **Forventer lang levetid**
* **Moduler: Indeholder views og services**
  + **Implementerer IModule**
  + **Afkoblet**
  + **Genbrug**
* **Modul katalog (håndterer loading af moduler)**
* **Flere views i en shell (main vindue)**
* **Views: UI**
* **View Model: Indkapsler præsentation logik og state**
* **Model: Data og business logik**
* **Commands: DelegateCommand/CompositeCommand**
* **Region: Område til view**
  + **RegionManager: loader regions**
* **Ændre UI i forhold til user navigering**
* **Bootstrapper: initialize and ceate the shell**

**Navigation**

* **Hvad der vises i region**
* **State-based (toggler mellem views)**
* **View-based (loader views)**
* **NavigationParameters**
* **OnNavigatedTo, OnNavigatedFrom**

**Kommunikation**

* **EventAggregator**
  + **PubSubEvent (typenavnet er en streng)**
  + **Ingen afhængigheder**
  + **Weak Refs – holdes ikke i live!**
    - * + **Kan dog godt gøres hård hvis nødvendig for at perform lidt bedre ved mange events**
  + **Kan køre i baggrunden og tilgå UIThread**

**Dependency Injection (IoC containers)**

* **Ikke implementeret af PRISM selv**
* **Kan fungere med Unity, MEF ..**

**Benytter sig af IoC containere.**

**Bootstrapper metoden som er afhængig af valget af IoC containere. Hvad laver boostrapperen, hvordan virker det, mekanismer. Regions, hvordan defineres de, switch imellem dem. Forskellige måder med view switching logic, message pattern (pub, sub). Weak references.**

1. **Redegør for begrebet "Interoperability" generelt, og redegør for brugen af PInvoke samt interoperability mellem COM og .Net. Vis eksempler på interoperability**

**Hvorfor interoperability?**

* **Bruge eksisterende kode uden at kende til underliggende typer**

**COM interoperability**

* **CLR håndtrer transformation:**
* **New -> CoCreateInstance**
* **Cast operator -> QueryInterface**
* **Memory mgmt -> Reference counting**
* **Exceptions -> Hresults**

**Runtime callable wrapper**

* **Dannes ved at referere COM fil**
* **Har finalizer som kalder release i COM objekt**
* **Nogle typer kan ikke mappes (fx C++ string, char\* kan)**

**PInvoke (Platform Invocation service)**

* **Tilgå funktioner i unmanaged fra managed**
* **CLR udfører LoadLibrary() + GetProcAddress()**

**Marshalling**

* **Transformation fra typesystem til en anden (unmanaged <–> managed)**
* **Strings bliver ikke kopieret tilbage -> brug StringBuilder**
* **Custom marshalling: [MarshalAs(Unmanagedtyope.LPArray)]**
* **Pin objekter eller Marshal class** 
  + **Allokere unmanaged memory, kopierer**

**Centralt: Marshal klassen**

**Problem med forskellige memorymodeller:**

* **Fx adresse på array kan flyttes af GC eller memory, ikke ens i C++ og C#**
* **Marshal klasse kan allokere unmanaged memory**
* **Problemer med typesystemer**

**Fare i DLL i C++**

* **Distribuering af nye DLL’er med tilføjede variabler** 
  + **Ingen indkapsulering på binær niveau, kun compiler**
  + **Problem hvis programmer stadig bruger gamle**
  + **Men indlæser ny -> overskriver andre adresser!**
  + **Kan løses ved COM**

**COM – veldefineret standard**

* **Deploy som DLL**
* **Separer interface fra implementering med wrapper**
  + **Peger på et objekt af klassen (konstant størrelse)**
* **Implementerer dynamisk cast (egentlig statisk)**
* **Bruger reference counting ved flere instanser**

**COM**

* **Interfaces: IUnknown, IDispatch, ISomething**
  + **IUnknown: AddRef, Release, QueryInterface**
  + **IDispatch: GetIDSOfNames, DispInvoke**
* **Interfaces må ikke ændres!**

**Typesystemer simpelt**

* **Primitive og interfaces med medlemsmetoder**
* **Ingen statiske mellem Com og .Net og parameterløs constructor**

1. **Redegør for problemer og muligheder for cross platform development på .Net platformen. Vis et eksempel på en cross platform component. Redegør for hvorledes man designer og implementerer Windows RT komponenter**

**Problem**

* **Forskellig kode for forskellige platforme**

**.Net Core**

* **Cross platform**
* **Fælles base class library**
* **Bliver delivered som NuGet pakker**

**Portable Class Library – hvorfor**

* **Understøtte forskellige platforme**

**.Net Standard**

* **Specifikation af API’er som implementeres af .Net Framework, .Net Core, Xamarin**
* **.Net Standard Lib: lag som mapper til Base Class Libs af platforme**
* **Erstatning for PCL da den kun kører mod specifik version**
* **.Net Standard løser: Nemt at tilføje platforme! (skal blot overholde *API version)***

**Primært**

**Hvad er problemet og løsning?**

**Portable Class Library og nye .Net Standard**

**Windows RT**

* **Windows Store apps**
* **Fået en del af generelle .NET funktioner**
* **COM komponenter med .NET typesystem**
* **Operativsystem**
* **Objektorienteret**
* **Kører i sandbox**
* **Driver Model**
* **Komponentmodel: COM**
* **Projections**
  + **CLR og Framework**
* **CCW and RCW**
* **Ingen System.Object – men ligner pga. projection**
* **WinRT komp: Kaldbar fra Javascript, C/C++, C#, CB**
  + **Public typer skal være WinRT**

**Dynamic Language Runtime**

* **Bedre dynamic language compability**
* **Easy code**
* **Langsommere end statisk. Hurtigere end reflection**
* **Lettere COM interop**

**Bygger på COM med .Nets typesystem**

**CLR bytter om på nogle ting og tilføjer (fx ligner det nedarver fra System.Object men gør ikke) med projections**

1. **Giv et overblik over de forskellige muligheder for cross platform udvikling for Android og iOS enheder. Redegør for Xamarins arkitektur og virkemåde for de forskellige understøttede platforme. Samt redegør for både fordele og udfordringer ved brug af Xamarin til App udvikling. Vis et eksempel på brug af Xamarin**

**Xamarin.Forms:**

* **.Net, C#**
* **Definer layout én gang**
* **MVVM**
* **Renderer som native code på hver platform**
  + **Ingen performance loss**
* **Nemt at bygge native user interfaces med shared code**
* **Abstraktion over iOS, Android, Windows Phone, fælles API**

**iOS kompilering**

* **Ahead of Time (AOT)**
  + **Native ARM executable**
* **kræver dog lige man har en Mac man kan bygge på**

**Android kompilering**

* **Mono - .Net runtime**
* **ART – android runtime**
* **Build process**
* **ACW (Android Callable Wrapper)**

**Pages, Controls som mapper til de underliggende platforme**

* **Fx ”Tabbed” Page har forskellige layouts, men Xamarin styrer dette**

**Fordele: Code reuse!! (avg. 75%) hvor resten er UI**

**Udfordringer: Har man mange customizations kræver det mere arbejde**

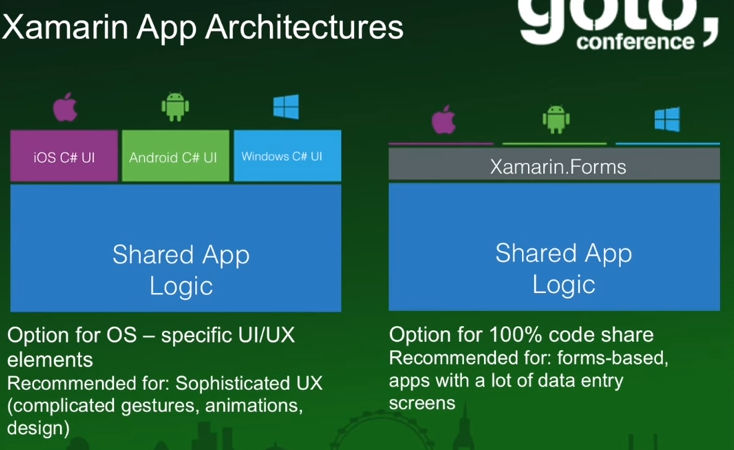
**Xamarin styrer:**

* **iOS: Marshaling fra .Net til iOS**
* **Android: Android bindings**
  + **Bygger APK ud fra .dex filer**

**Xamarin App Architectures:**

**Traditional Xamarin vs Xamarin.Forms:**

* **Forms:**
  + **Pages, Layouts, Controls (Content, Master Detail etc)**
  + **Data Binding**
  + **Navigation**
  + **MVVM fx Prism**



**Portable Class Library**

* **Forms-projektet ligger som PCL**
* **Lave interface i Portable Class Library**
* **Impl. i iOS, Android, Windows Phone kode**
* **DependencyService:**
  + **For at implementere Platform afhængigt**

**Shared Asset Library**

* **Kode et sted**
* **Kopieres til alle platforme**
* **Kompilerer kun platform specifikt**

**Native på iOS, kører Monos virtuelle maskine ved siden af Androids**

**Laver type Marshaling ligesom COM interop. Alle native api på iOs eller Android oversættes til C#’s (.Net) typesprog så alt ses i C#, men er egentlig native. Overbygning med Xamarin.Forms som har en række layout panels og controls. Samme brugergrænseflade men konverteres til native controls på de respektive platforme. Stadig native men kodes i samme sprog! Ekstra abstraktionslag. Kan være lidt tungt? Lidt mere overhead. Køres native brugergrænseflade er det samme performance ved Xamarin som hvis man kørte Swift på iOS eller Java på Android Studio.**

**Native vil sige native API’er man kalder, man koder stadig i C# uanset platform.**

1. **Redegør for begrebet ”Services”, og redegør for hvorledes man designer og implementerer en Windowsservice ved brug af .Net og C#. Vis et eksempel på en windowsservice.**

**Services:**

* **Ingen brugergrænseflade**

**Windows service kører ikke i en user session. Kan starte op når en pc (system) starter, fx webserver. Har ingen brugergrænseflade og må ikke tilgås! Kan bruge Windows Event Log for at logge fejl fra servicen**

**Kan implementeres ved TopShelf eller VS**

* **Kan ikke testes i VS, skal installeres**
* **Kan udskrive events i Logbog**

**I VS:**

* **ServiceBase**
* **Alle funktioner skal returnere inden 30 sekunder**
* **Svært at debug? (eller ikke)**

**Service Control Manager (SCM):**

* **Sørger for at kalde LoadService fra .Net Runtime som starter Main**
* **RunService -> onStart (fx lave ny baggrundstråd)**
* **StopService -> onStop (stopper tråd)**

**Controlling a service from an application:**

* **ServiceController**
  + **Kan hente alle services og finde den service man ønsker**
  + **Kan sende meddelser til servicen**
    - **Start, Stop, Pause …**
  + **Sender enums (128-255)**
  + **Kan ikke modtage parametre:**
    - **Send til fil eller**
    - **Registreringsdatabasen**

1. **Redegør for brugen af komponenter ved client side webudvikling. Hvad skal browseren understøtte for at man kan lave UIkomponenter, som er uafhængige af et JavaScript framework. Vis et eksempel på en web komponent, som er udviklet uden brug af framework eller evt. ved brug af LitElement.**

**NOT A CURRENT QUESTION!!!!!!!!!:**

**Redegør for COMs arkitektur og terminologi samt forklar hvorledes man kan udvikle COM komponenter ved brug af ATL:**

**Arkitektur:**

* **Binære indkapsling**
* **Alt er igennem et interface**
* **Ingen parameter til constructor**
* **Ingen implementeringsdetaljer, kun interface**
* **Indpakket komponent i en DLL (eller exe)**
  + **Skal have 4 metoder som der skal implemeteres i en rigtig rækkefølge**
* **Skaber forbindelse fra et program til den med COM-funktionalitet i Windows, bruger ProcID, slår den op i registeringsdatabasen (er vigtig!)**

**Den der så laver arbejdet med finde komponenterne og få dem instantieret er service kontrol manageren. Samme med Windows Service som bruges til at aktivere COM ting**

**Lavet en COM:**

* **Registeringdatabasen**
* **Loader ind**
* **Får adgang til factory som returnerer class factory**
* **Via den kan få lavet instanser**
* **Har man den kalder man den direkte via custom interfaces**

**Memory model:**

**Reference counted**

**IUnknown interface: query interface**

**Hele fundamentet af COM teknologi! (VIGTIGT)**

**Standard kode som alle komponenter skal implementere, ligge i template library som man kan nedarve fra. Derfor har man ATL (Active Template Library) som er en samling af template filer som har boiler plate kode som implementerer mange standard COM interfaces**