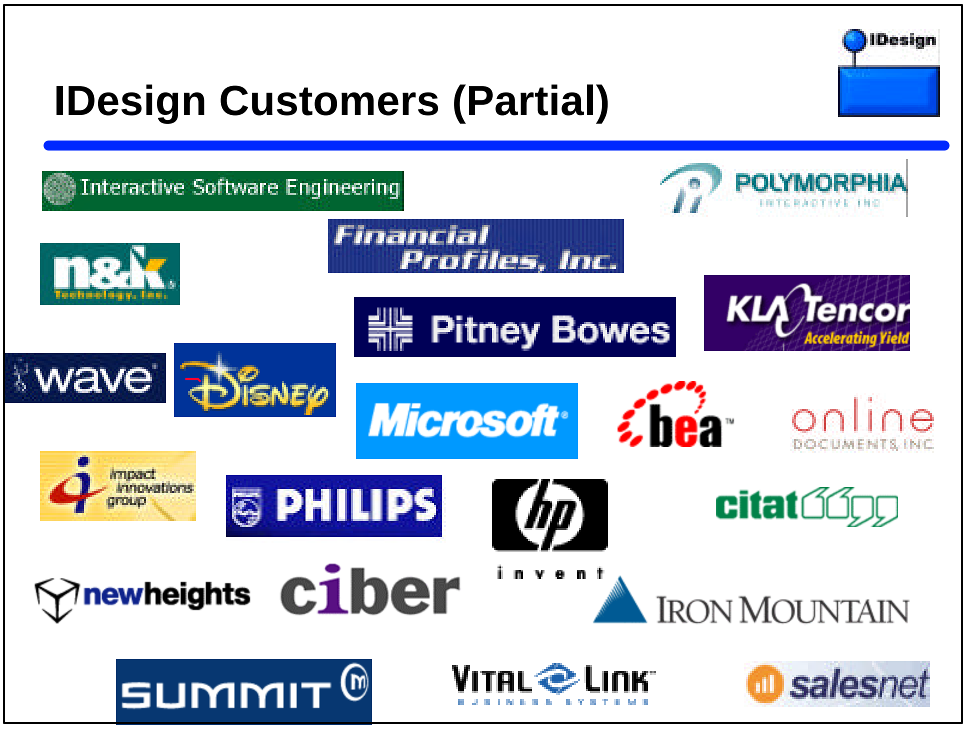
.NET Component-Oriented Development

Juval Löwy
www.idesign.net

About IDesign

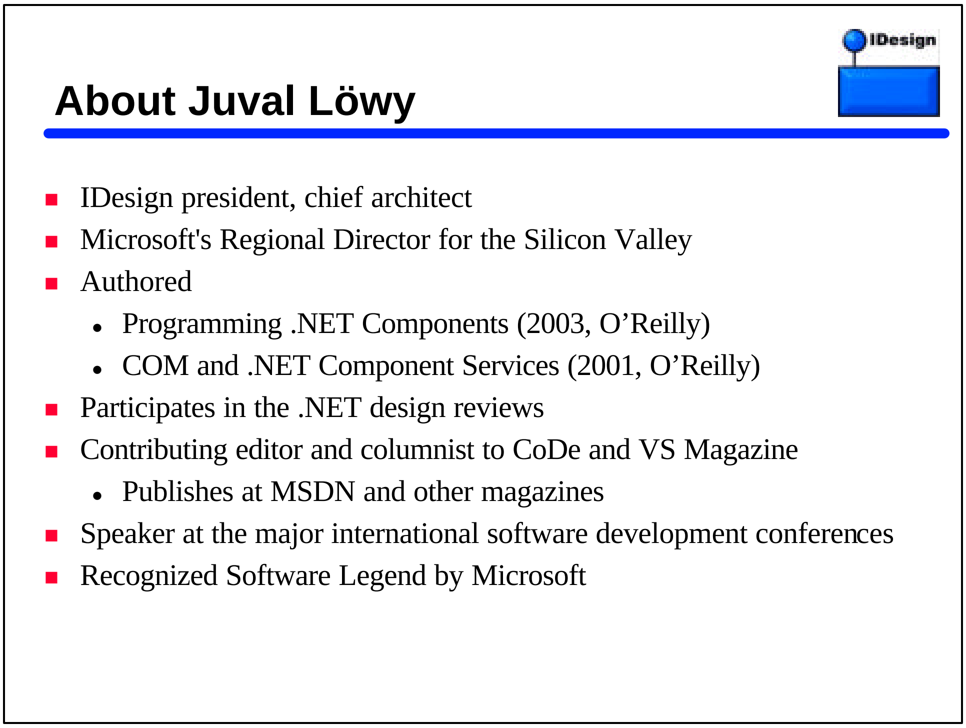
- .NET architecture consulting, training and process improvement
- Comprised of leading world-class experts
  - Authors, speakers, veterans
  - Work closely with Microsoft
    - Strategic reviews
    - High end resource for the local office
- Community involvement
- Multiple awards
- www.idesign.net



IDesign

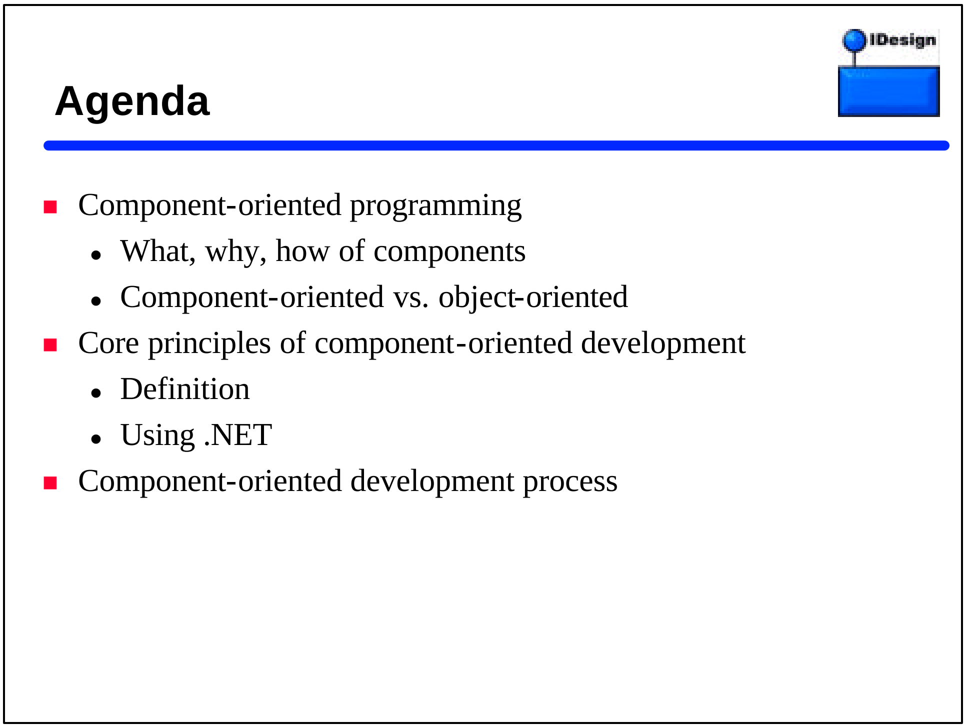
IDesign Customers (Partial)

Interactive Software Engineering
POLYMORPHIA INTERACTIVE INC
Financial Profiles, Inc.
n&k Technology, Inc.
Pitney Bowes
KLA Tencor Accelerating Yield
wave
Disney
Microsoft
bea
online DOCUMENTS, INC
impact innovations group
PHILIPS
hp invent
citat
newheights
ciber
IRON MOUNTAIN
SUMMIT
VITAL LINK BUSINESS SYSTEMS
salesnet



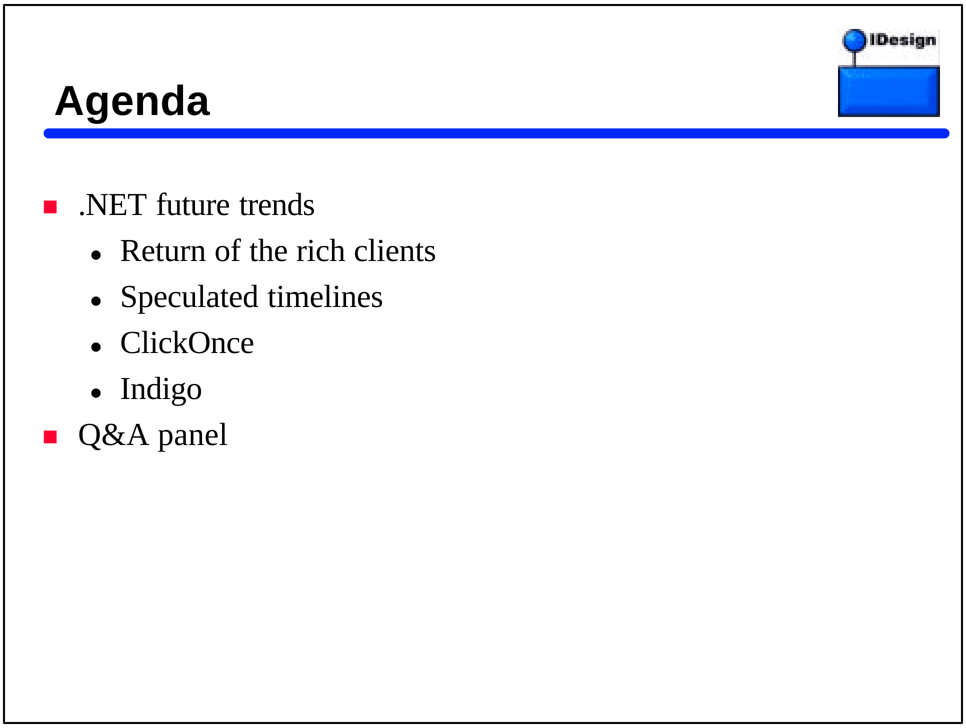
About Juval Löwy

- IDesign president, chief architect
- Microsoft's Regional Director for the Silicon Valley
- Authored
  - Programming .NET Components (2003, O’Reilly)
  - COM and .NET Component Services (2001, O’Reilly)
- Participates in the .NET design reviews
- Contributing editor and columnist to CoDe and VS Magazine
  - Publishes at MSDN and other magazines
- Speaker at the major international software development conferences
- Recognized Software Legend by Microsoft



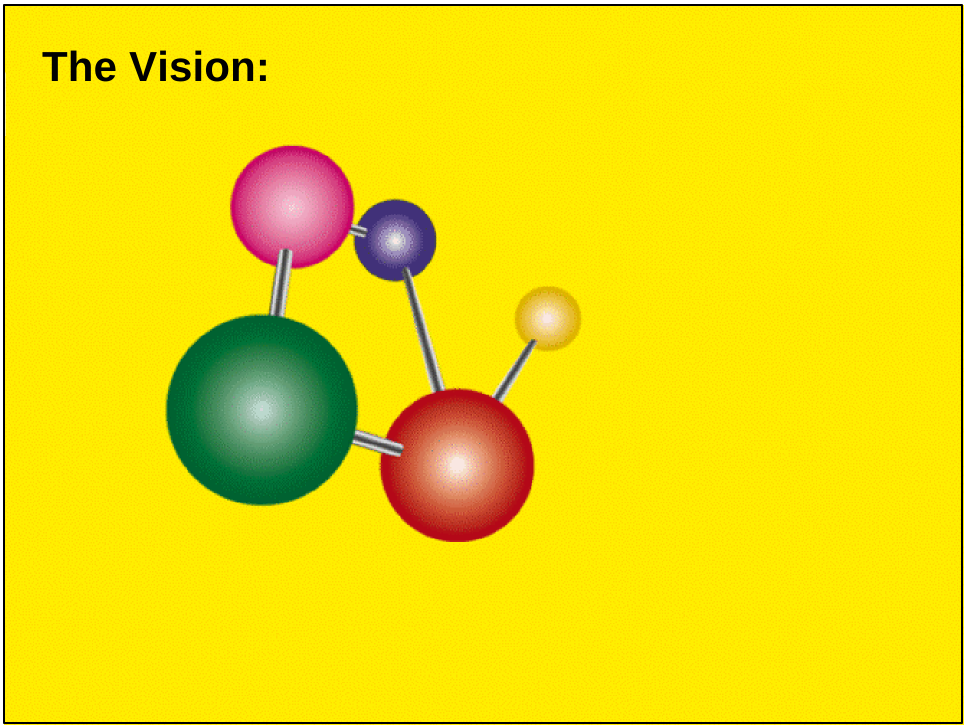
Agenda

- Component-oriented programming
  - What, why, how of components
  - Component-oriented vs. object-oriented
- Core principles of component-oriented development
  - Definition
  - Using .NET
- Component-oriented development process

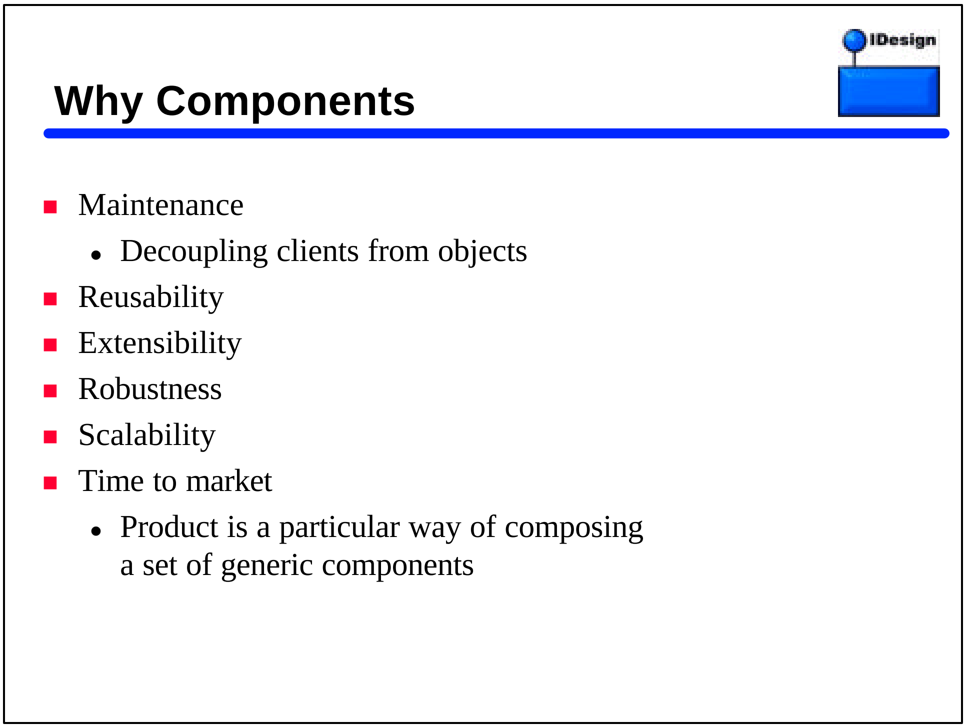


Agenda

- .NET future trends
  - Return of the rich clients
  - Speculated timelines
  - ClickOnce
  - Indigo
- Q&A panel

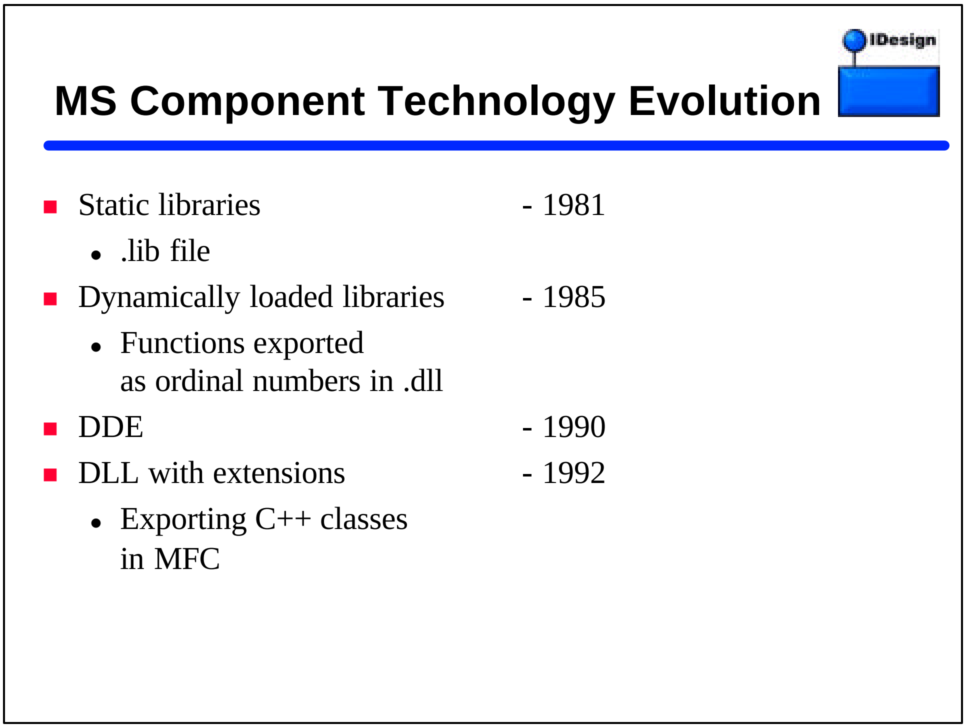


The Vision:



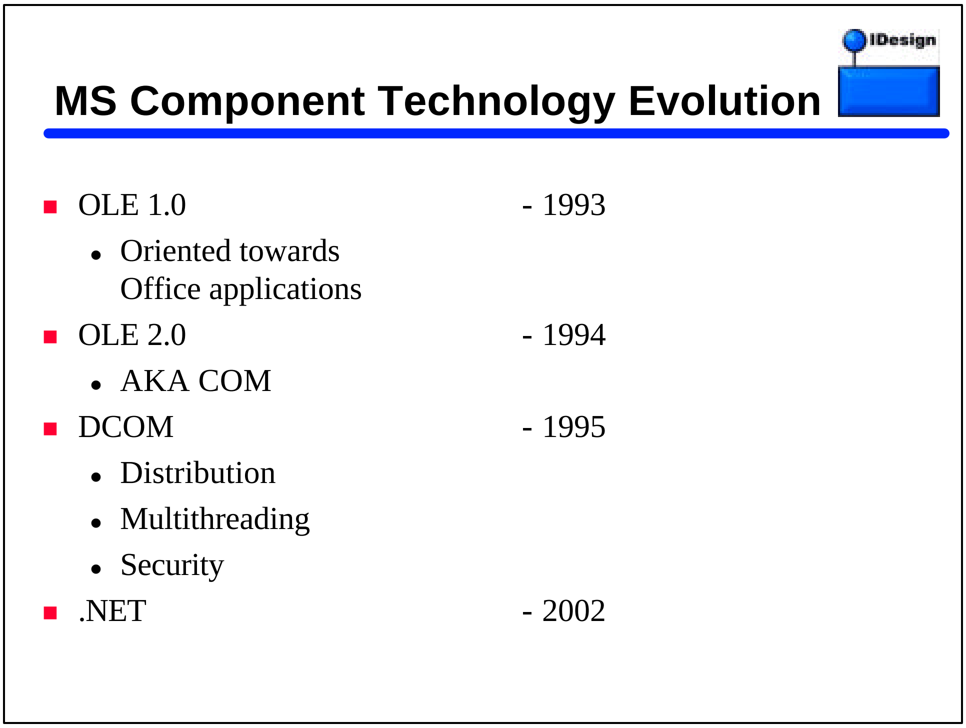
Why Components

- Maintenance
  - Decoupling clients from objects
- Reusability
- Extensibility
- Robustness
- Scalability
- Time to market
  - Product is a particular way of composing a set of generic components



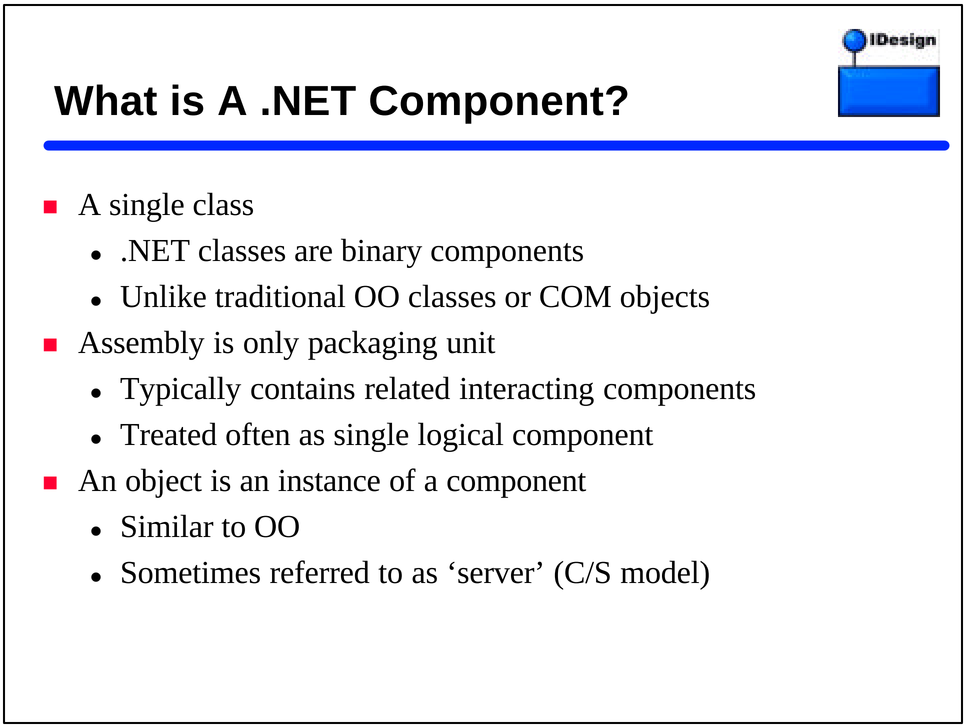
MS Component Technology Evolution

- Static libraries - 1981
  - .lib file
- Dynamically loaded libraries - 1985
  - Functions exported as ordinal numbers in .dll
- DDE - 1990
- DLL with extensions - 1992
  - Exporting C++ classes in MFC



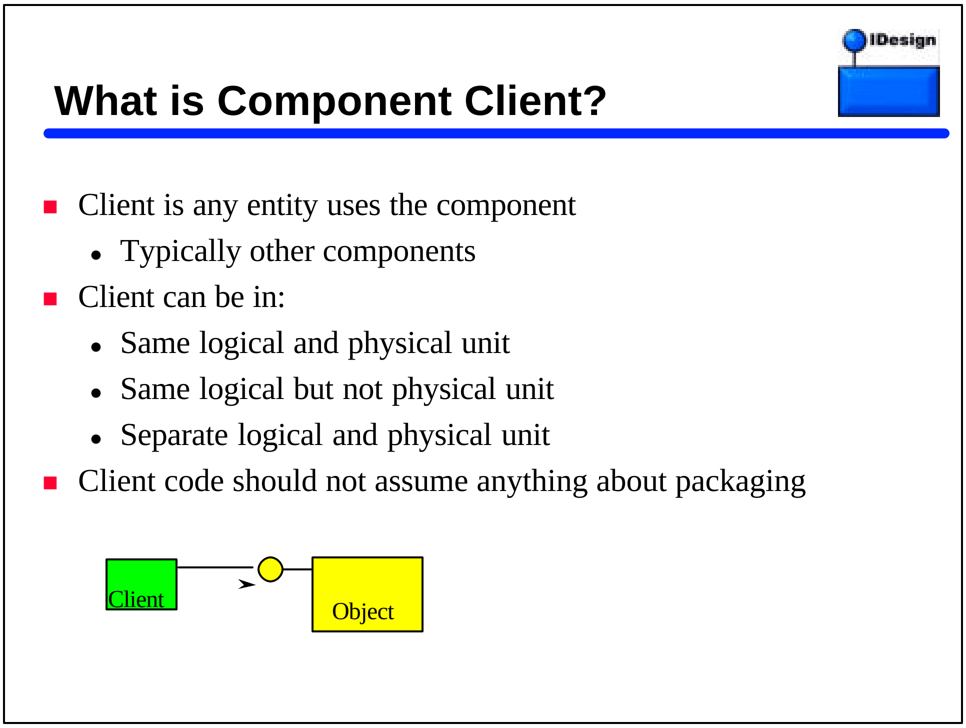
MS Component Technology Evolution

- OLE 1.0 - 1993
  - Oriented towards Office applications
- OLE 2.0 - 1994
  - AKA COM
- DCOM - 1995
  - Distribution
  - Multithreading
  - Security
- .NET - 2002



What is A .NET Component?

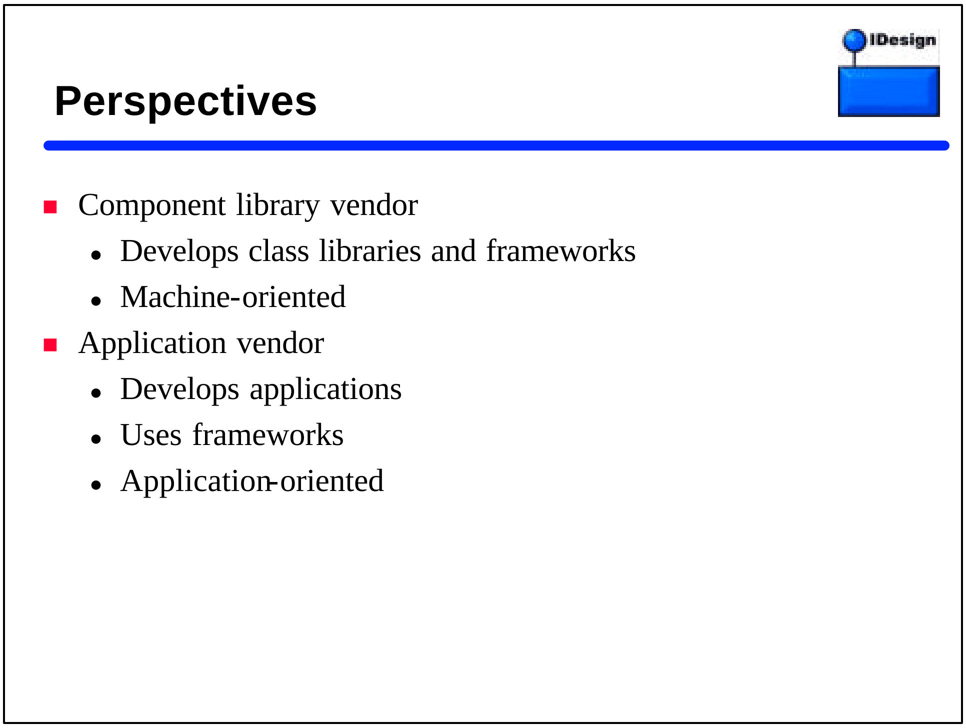
- A single class
  - .NET classes are binary components
  - Unlike traditional OO classes or COM objects
- Assembly is only packaging unit
  - Typically contains related interacting components
  - Treated often as single logical component
- An object is an instance of a component
  - Similar to OO
  - Sometimes referred to as ‘server’ (C/S model)



What is Component Client?

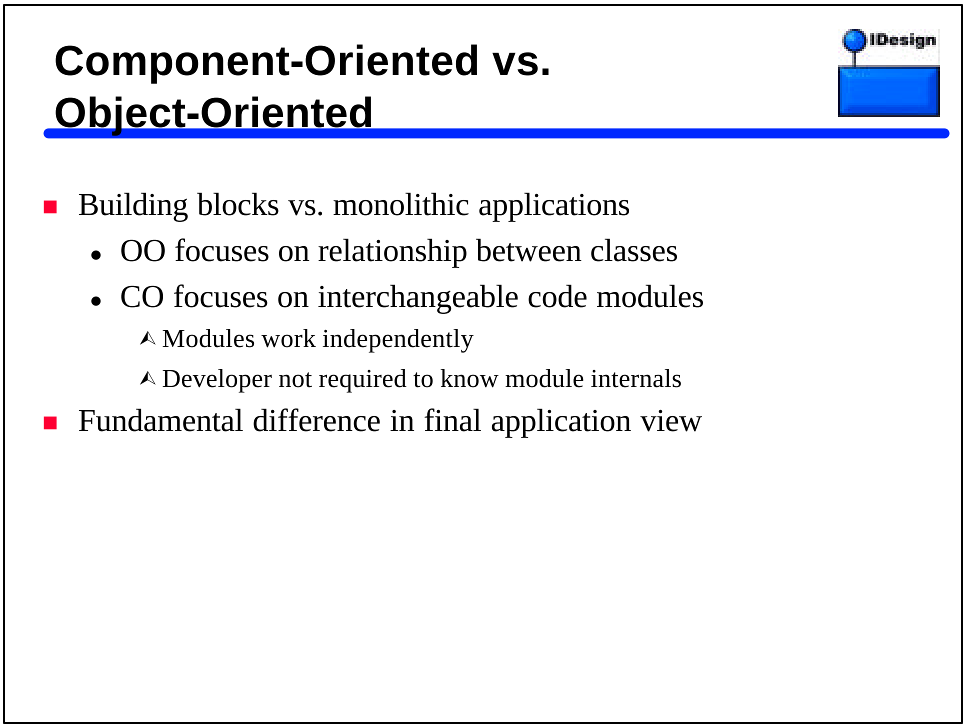
- Client is any entity uses the component
  - Typically other components
- Client can be in:
  - Same logical and physical unit
  - Same logical but not physical unit
  - Separate logical and physical unit
- Client code should not assume anything about packaging

Client
Object



Perspectives

- Component library vendor
  - Develops class libraries and frameworks
  - Machine-oriented
- Application vendor
  - Develops applications
  - Uses frameworks
  - Application-oriented



Component-Oriented vs. Object-Oriented

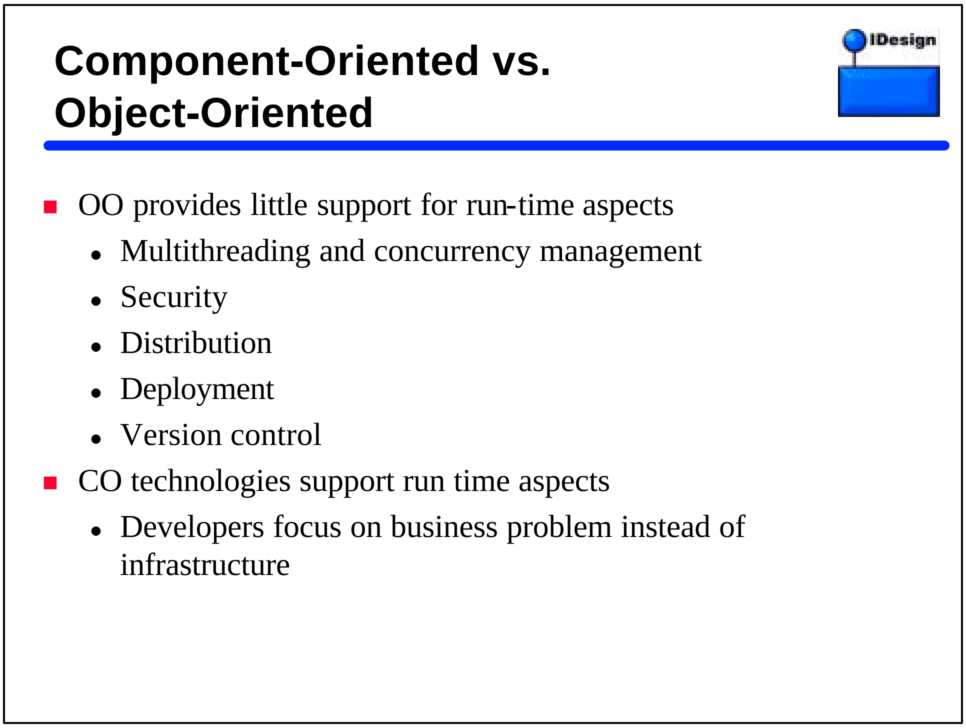
- Building blocks vs. monolithic applications
  - OO focuses on relationship between classes
  - CO focuses on interchangeable code modules
    - Modules work independently
    - Developer not required to know module internals
- Fundamental difference in final application view

Component-Oriented vs. Object-Oriented

- Traditional object-oriented
  - Logic factored to many fine-grained classes
  - Once compiled, result is monolithic binary
  - All classes share same physical deployment unit
  - All classes share same process
    - Same address space
    - Same security privileges
  - Shared source files
    - Single implementation language
  - Change made to one class can trigger massive re-linking
    - Retesting
    - Redeployment

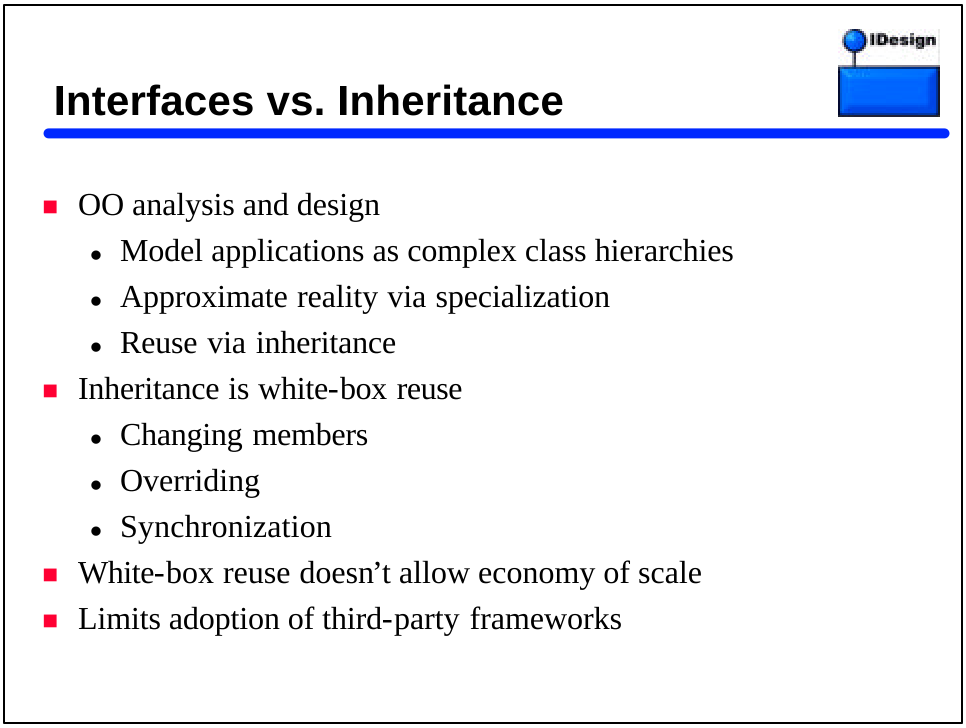
Component-Oriented vs. Object-Oriented

- Component-oriented
  - Application comprises a collection of interacting ***binary*** components
- Particular binary component may not do much
  - Can be general-purpose component
  - Can be highly specialized
- Requirements implemented by gluing individual components
  - Component-enabling technologies provide infrastructure to connect binary components
    - COM, J2EE, CORBA, .NET
  - Distinct in ease of use



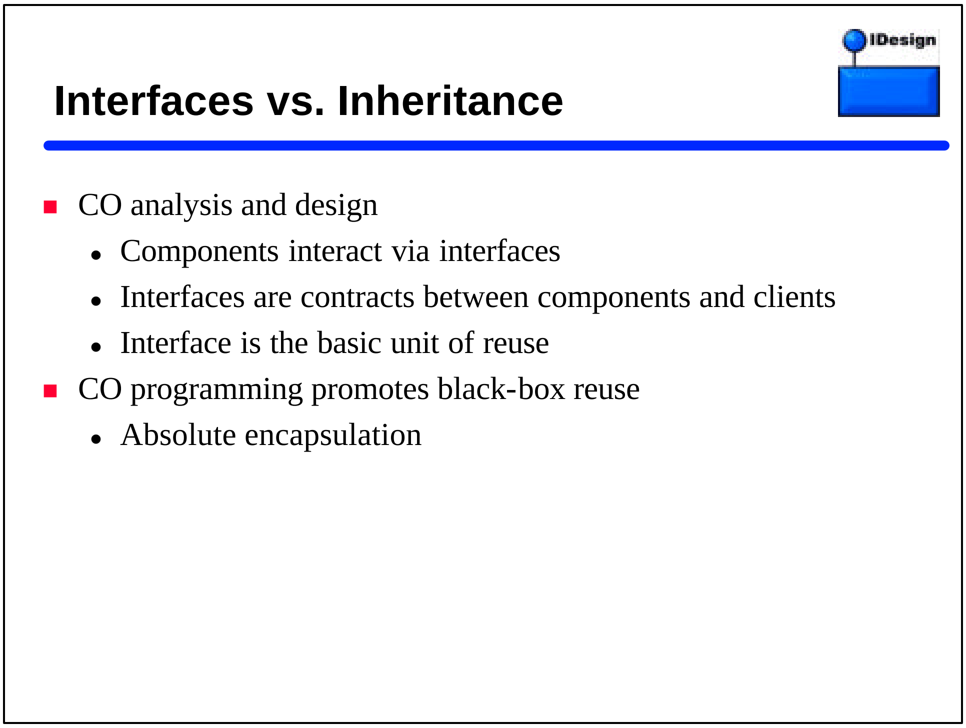
Component-Oriented vs. Object-Oriented

- OO provides little support for run-time aspects
  - Multithreading and concurrency management
  - Security
  - Distribution
  - Deployment
  - Version control
- CO technologies support run time aspects
  - Developers focus on business problem instead of infrastructure



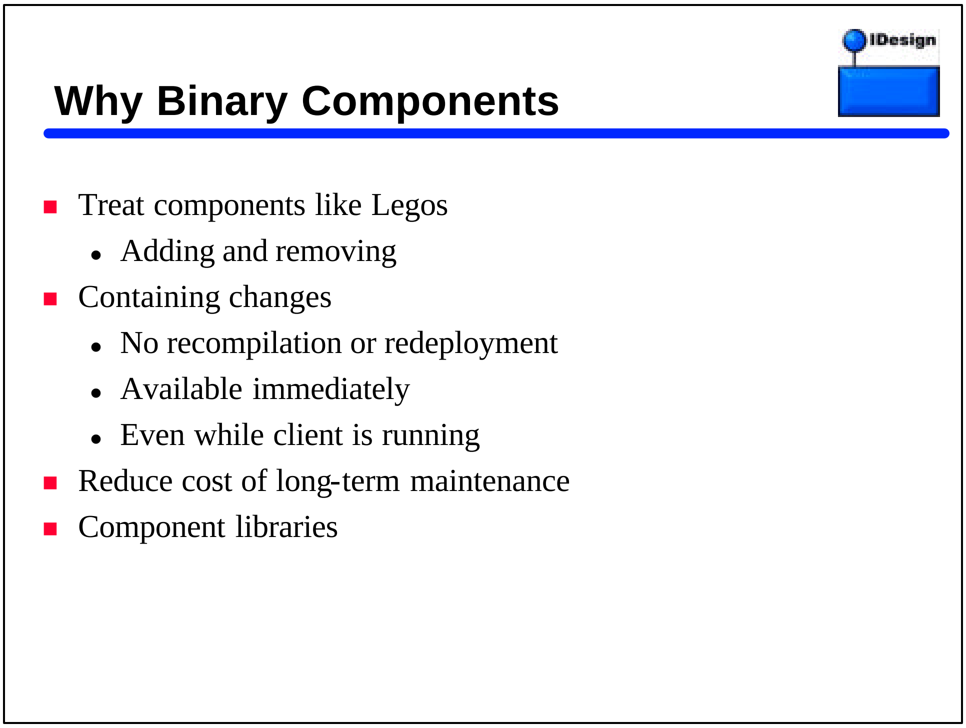
Interfaces vs. Inheritance

- OO analysis and design
  - Model applications as complex class hierarchies
  - Approximate reality via specialization
  - Reuse via inheritance
- Inheritance is white-box reuse
  - Changing members
  - Overriding
  - Synchronization
- White-box reuse doesn’t allow economy of scale
- Limits adoption of third-party frameworks



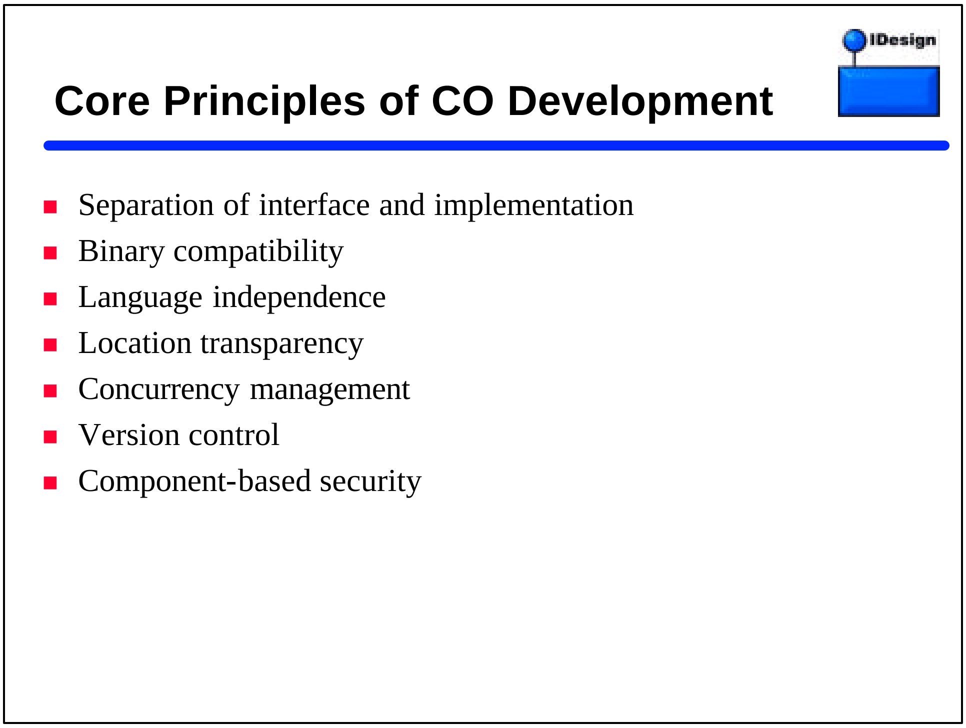
Interfaces vs. Inheritance

- CO analysis and design
  - Components interact via interfaces
  - Interfaces are contracts between components and clients
  - Interface is the basic unit of reuse
- CO programming promotes black-box reuse
  - Absolute encapsulation



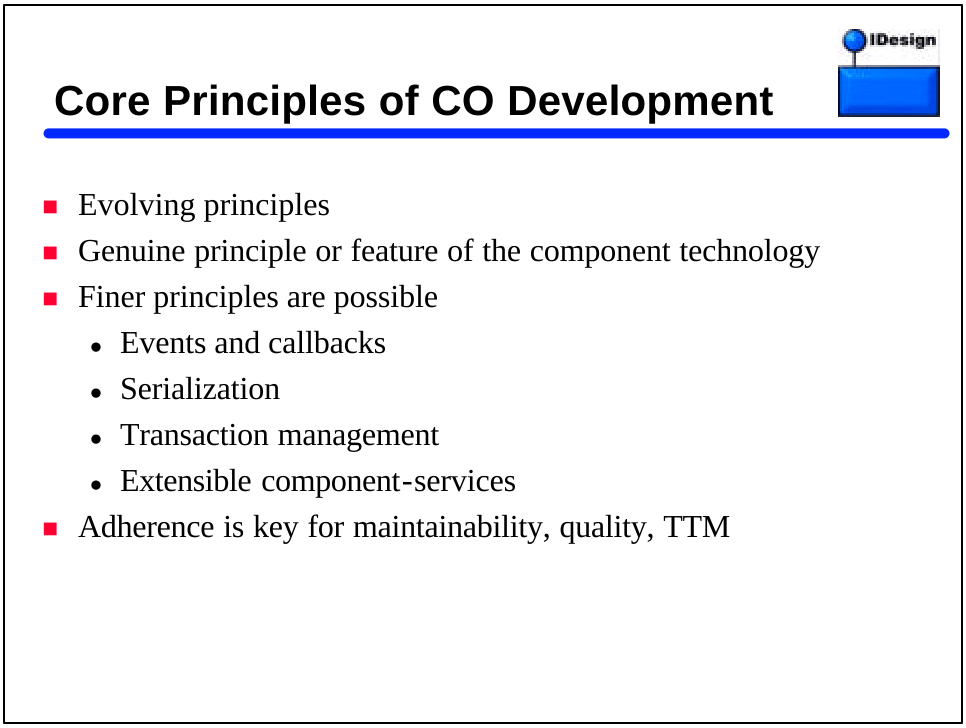
Why Binary Components

- Treat components like Legos
  - Adding and removing
- Containing changes
  - No recompilation or redeployment
  - Available immediately
  - Even while client is running
- Reduce cost of long-term maintenance
- Component libraries



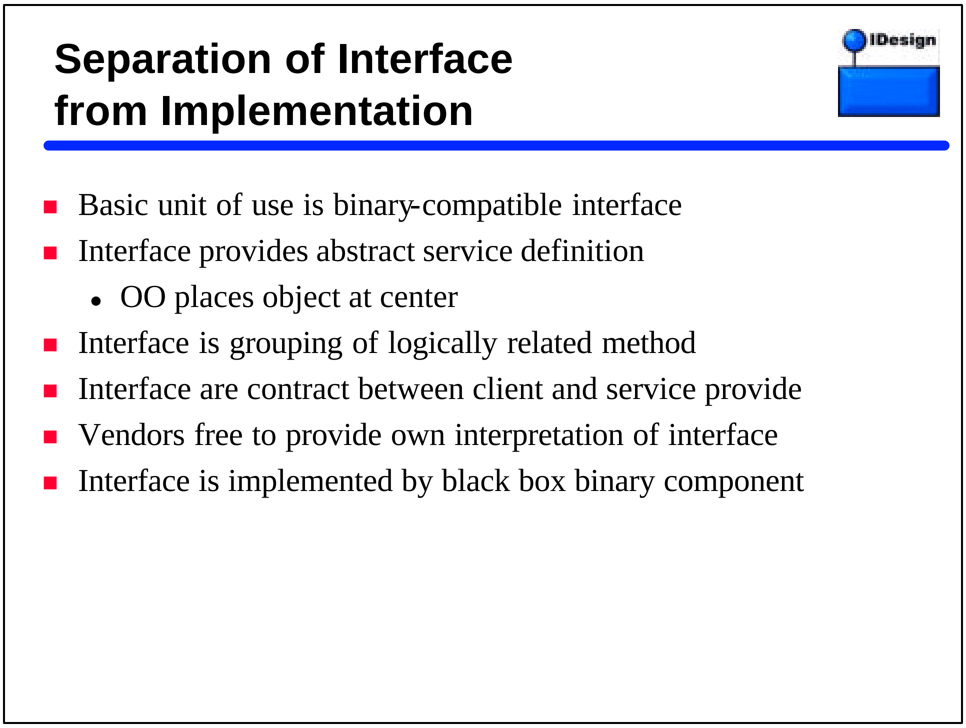
Core Principles of CO Development

- Separation of interface and implementation
- Binary compatibility
- Language independence
- Location transparency
- Concurrency management
- Version control
- Component-based security



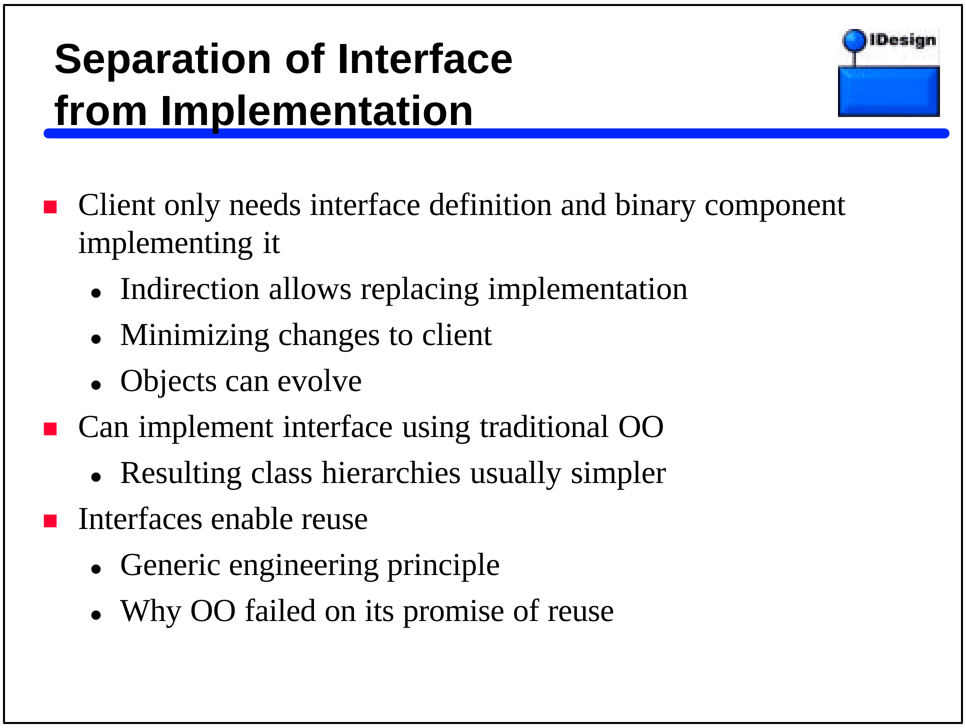
Core Principles of CO Development

- Evolving principles
- Genuine principle or feature of the component technology
- Finer principles are possible
  - Events and callbacks
  - Serialization
  - Transaction management
  - Extensible component-services
- Adherence is key for maintainability, quality, TTM



Separation of Interface from Implementation

- Basic unit of use is binary-compatible interface
- Interface provides abstract service definition
  - OO places object at center
- Interface is grouping of logically related method
- Interface are contract between client and service provide
- Vendors free to provide own interpretation of interface
- Interface is implemented by black box binary component



Separation of Interface from Implementation

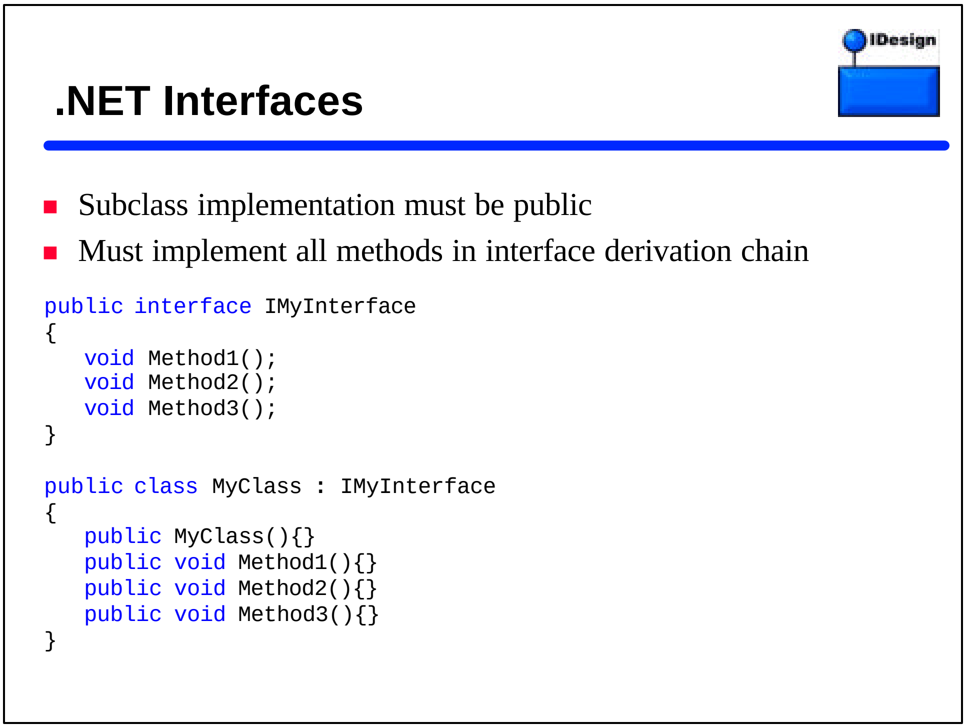
- Client only needs interface definition and binary component implementing it
  - Indirection allows replacing implementation
  - Minimizing changes to client
  - Objects can evolve
- Can implement interface using traditional OO
  - Resulting class hierarchies usually simpler
- Interfaces enable reuse
  - Generic engineering principle
  - Why OO failed on its promise of reuse

Interfaces Vs. Abstract Classes

- Interfaces are not the same as abstract classes
  - Abstract class can still have implementation
  - Class can derive from only one base class
  - Class can derive from multiple interfaces
  - Abstract class can derive from any other class or interface(s)
  - Interface can only derive from other interfaces
  - Abstract class can have non-public members
  - Abstract class can have constructors static members and constants
- Differences are deliberate to provide for a formal public contract

.NET Interfaces

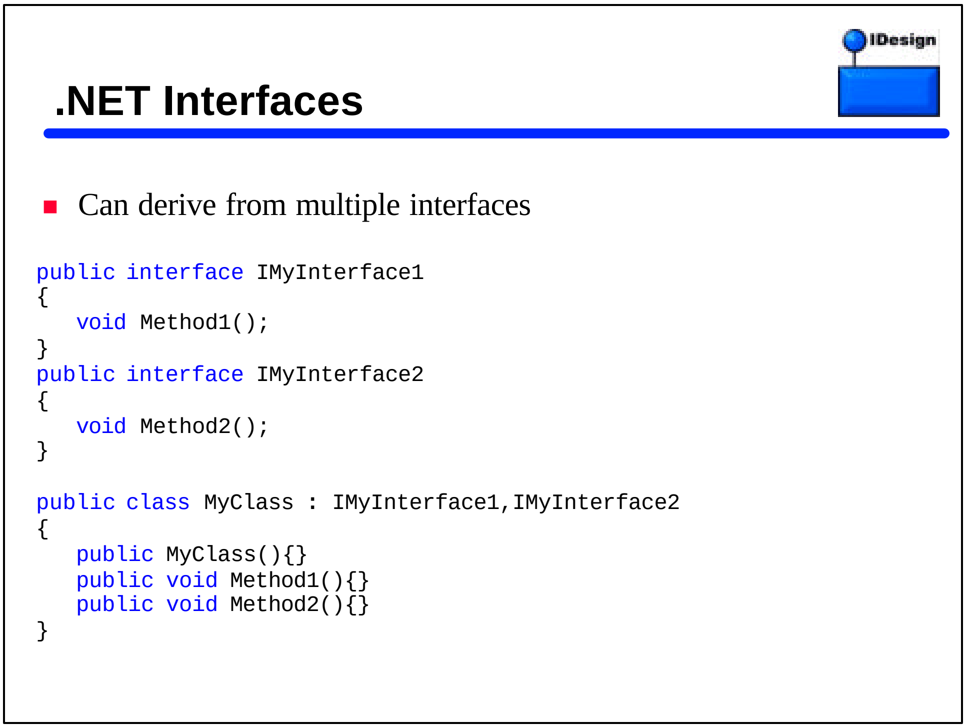
- Interfaces can only derive from other interfaces
- Interface can derive from multiple other interfaces
  - Unlike COM
- All interface methods are public
  - Contract semantics



.NET Interfaces

- Subclass implementation must be public
- Must implement all methods in interface derivation chain

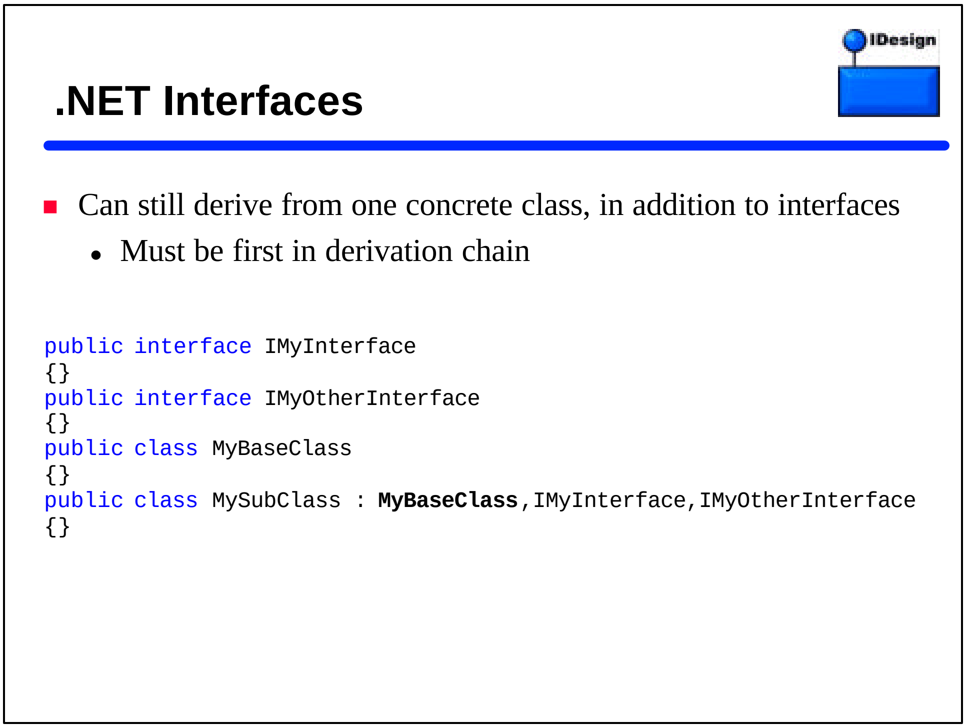
```
public interface IMyInterface
{
   void Method1();
   void Method2();
   void Method3();
}

public class MyClass : IMyInterface
{
   public MyClass(){}
   public void Method1(){}
   public void Method2(){}
   public void Method3(){}
}
```



.NET Interfaces

- Can derive from multiple interfaces

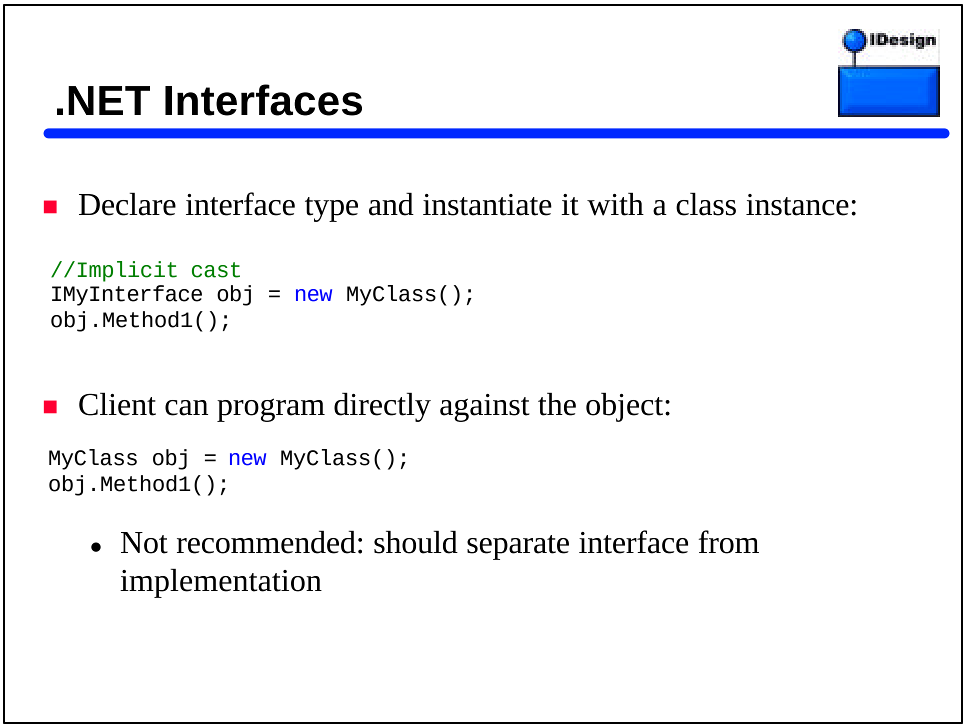
```
public interface IMyInterface1
{
   void Method1();
}
public interface IMyInterface2
{
   void Method2();
}

public class MyClass : IMyInterface1,IMyInterface2
{
   public MyClass(){}
   public void Method1(){}
   public void Method2(){}
}
```



.NET Interfaces

- Can still derive from one concrete class, in addition to interfaces
  - Must be first in derivation chain

```
public interface IMyInterface
{}
public interface IMyOtherInterface
{}
public class MyBaseClass
{}
public class MySubClass : MyBaseClass,IMyInterface,IMyOtherInterface
{}
```



.NET Interfaces

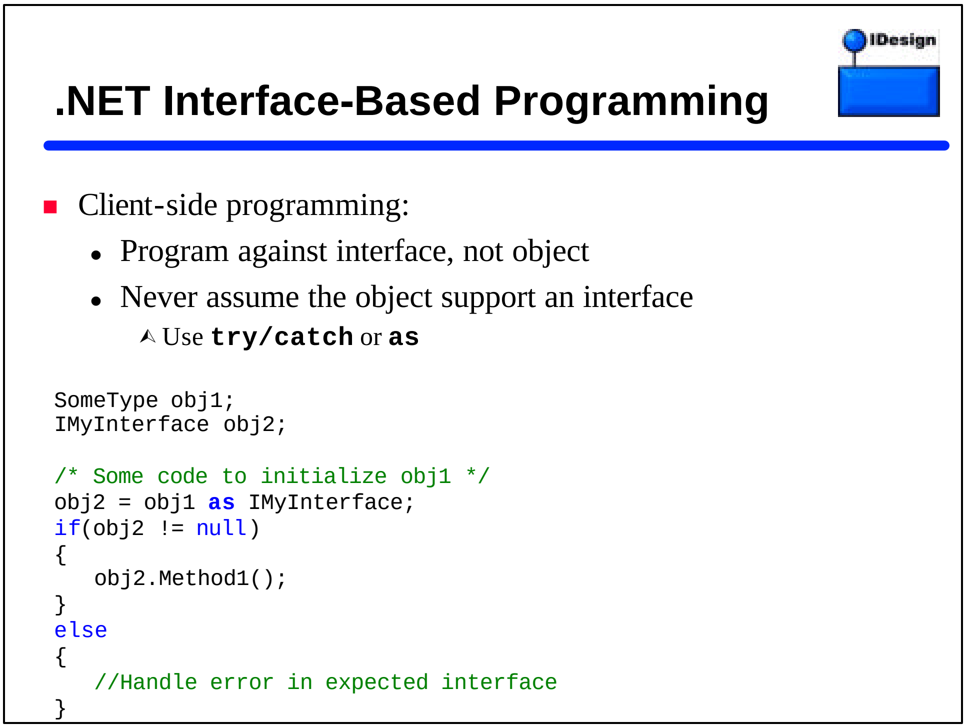
- Declare interface type and instantiate it with a class instance:

```
//Implicit cast
IMyInterface obj = new MyClass();
obj.Method1();
```

- Client can program directly against the object:

```
MyClass obj = new MyClass();
obj.Method1();
```

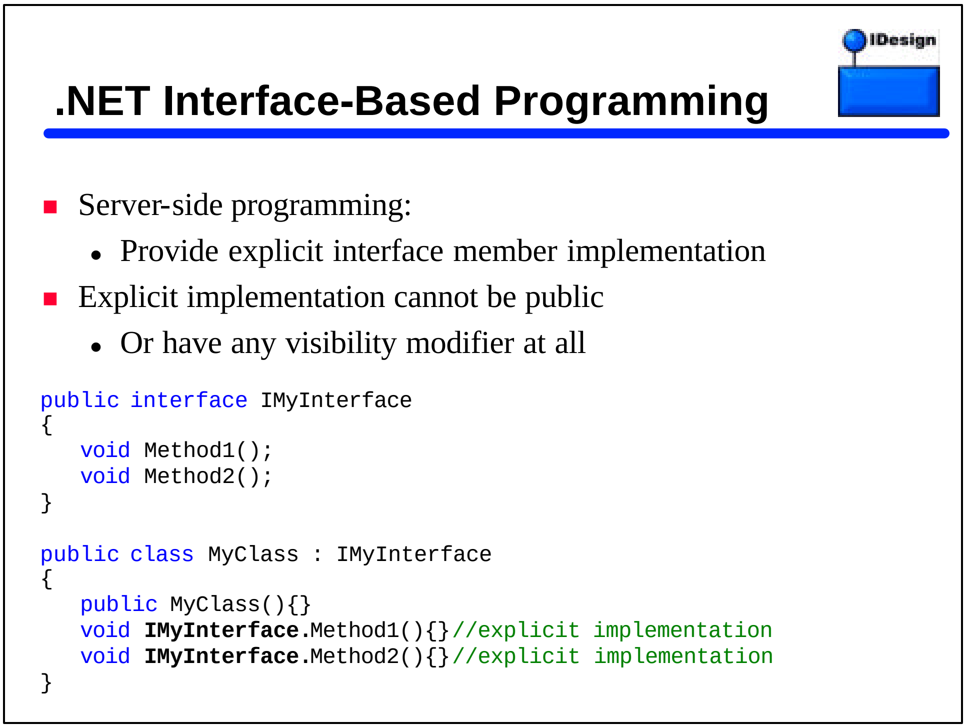
  - Not recommended: should separate interface from implementation



.NET Interface-Based Programming

- Client-side programming:
  - Program against interface, not object
  - Never assume the object support an interface
    - Use **try/catch** or **as**

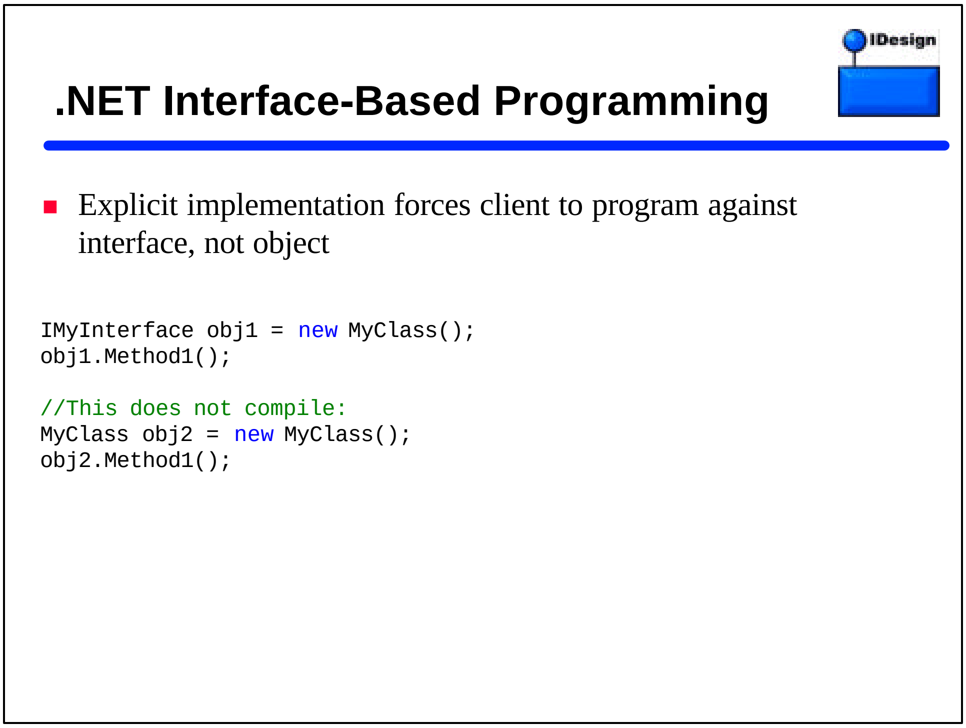
```
SomeType obj1;
IMyInterface obj2;

/* Some code to initialize obj1 */
obj2 = obj1 as IMyInterface;
if(obj2 != null)
{
   obj2.Method1();
}
else
{
   //Handle error in expected interface
}
```



.NET Interface-Based Programming

- Server-side programming:
  - Provide explicit interface member implementation
- Explicit implementation cannot be public
  - Or have any visibility modifier at all

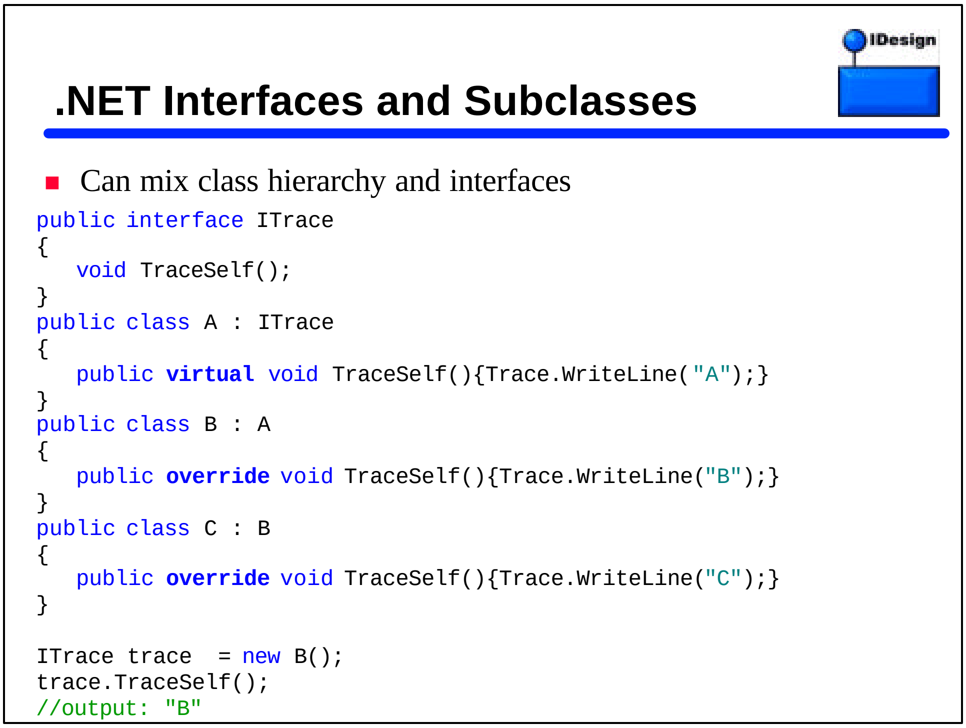
```
public interface IMyInterface
{
   void Method1();
   void Method2();
}

public class MyClass : IMyInterface
{
   public MyClass(){}
   void IMyInterface.Method1(){}//explicit implementation
   void IMyInterface.Method2(){}//explicit implementation
}
```



.NET Interface-Based Programming

- Explicit implementation forces client to program against interface, not object

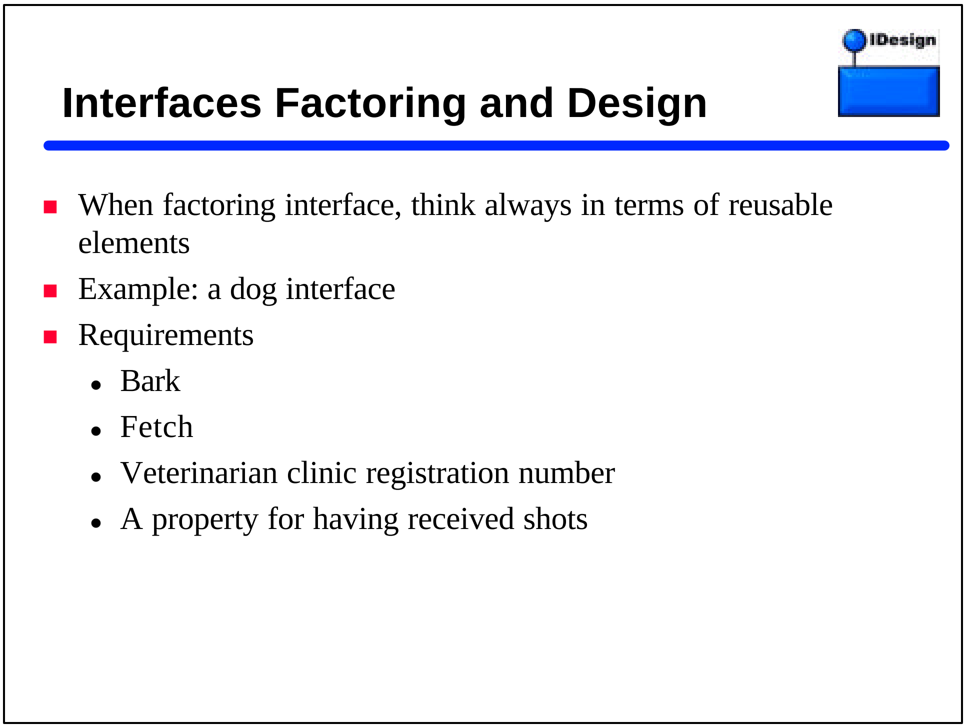
```
IMyInterface obj1 = new MyClass();
obj1.Method1();

//This does not compile:
MyClass obj2 = new MyClass();
obj2.Method1();
```



.NET Interfaces and Subclasses

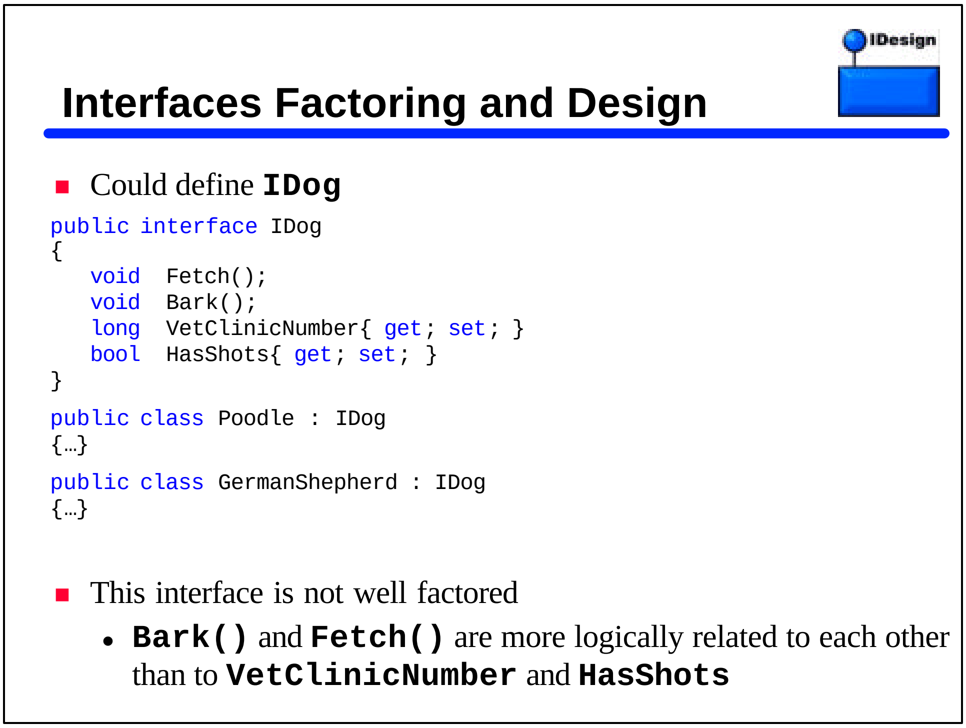
- Can mix class hierarchy and interfaces

```
public interface ITrace
{
   void TraceSelf();
}
public class A : ITrace
{
   public virtual void TraceSelf(){Trace.WriteLine("A");}
}
public class B : A
{
   public override void TraceSelf(){Trace.WriteLine("B");}
}
public class C : B
{
   public override void TraceSelf(){Trace.WriteLine("C");}
}

ITrace trace  = new B();
trace.TraceSelf();
//output: "B"
```



Interfaces Factoring and Design

- When factoring interface, think always in terms of reusable elements
- Example: a dog interface
- Requirements
  - Bark
  - Fetch
  - Veterinarian clinic registration number
  - A property for having received shots

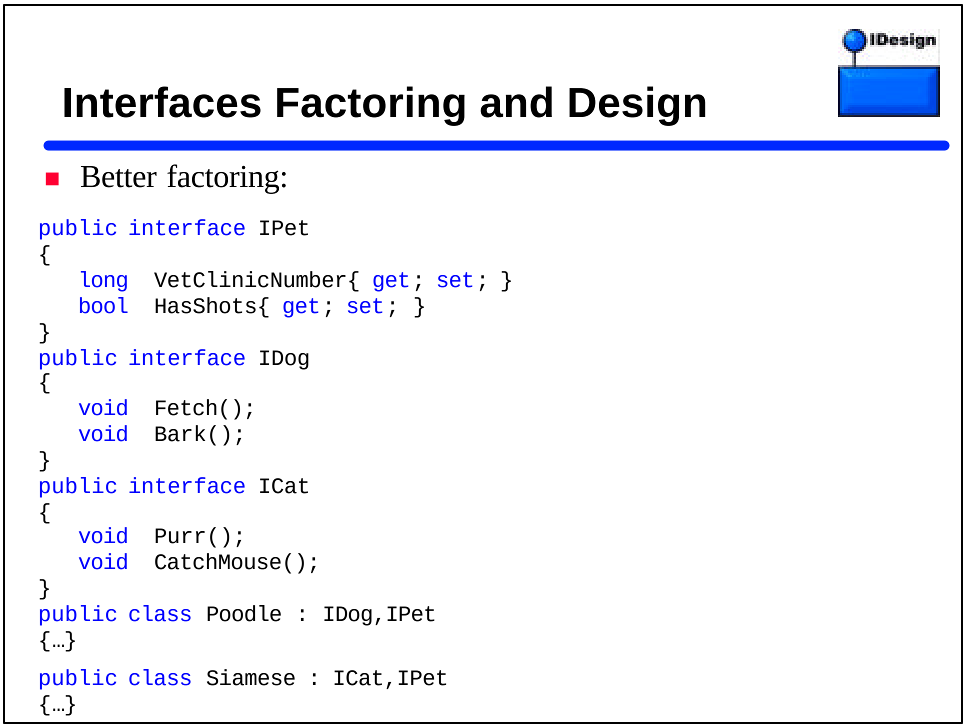


Interfaces Factoring and Design

- Could define **`IDog`**

```
public interface IDog
{
   void  Fetch();
   void  Bark();
   long  VetClinicNumber{ get; set; }
   bool  HasShots{ get; set; }
}
public class Poodle : IDog
{…}
public class GermanShepherd : IDog
{…}
```

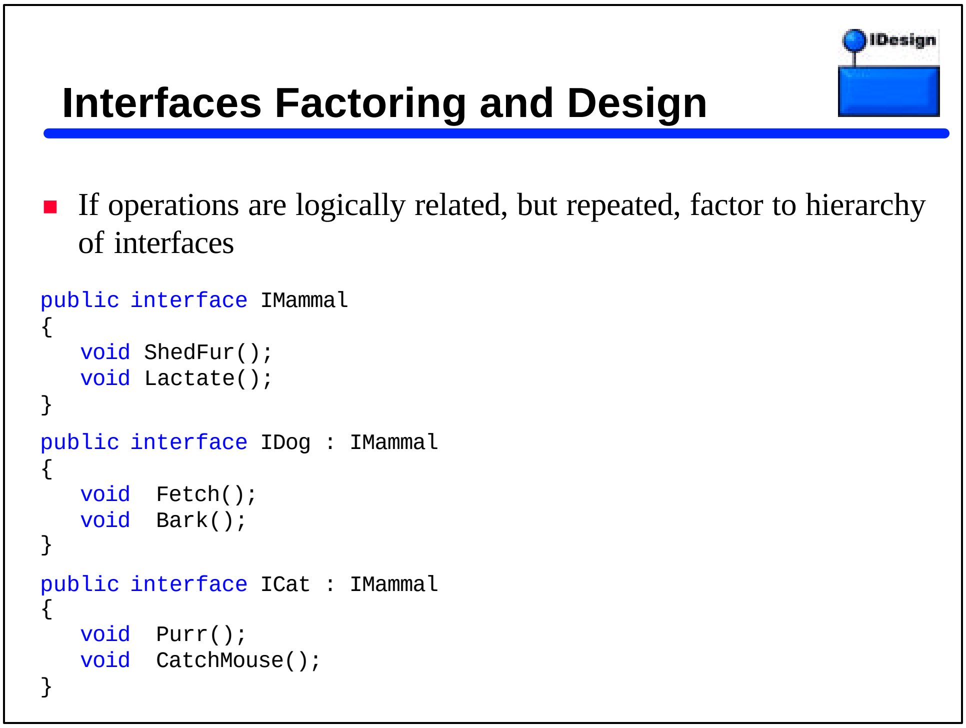
- This interface is not well factored
  - **`Bark()`** and **`Fetch()`** are more logically related to each other than to **`VetClinicNumber`** and **`HasShots`**



Interfaces Factoring and Design

- Better factoring:

```
public interface IPet
{
   long  VetClinicNumber{ get; set; }
   bool  HasShots{ get; set; }
}
public interface IDog
{
   void  Fetch();
   void  Bark();
}
public interface ICat
{
   void  Purr();
   void  CatchMouse();
}
public class Poodle : IDog,IPet
{…}
public class Siamese : ICat,IPet
{…}
```



Interfaces Factoring and Design

- If operations are logically related, but repeated, factor to hierarchy of interfaces

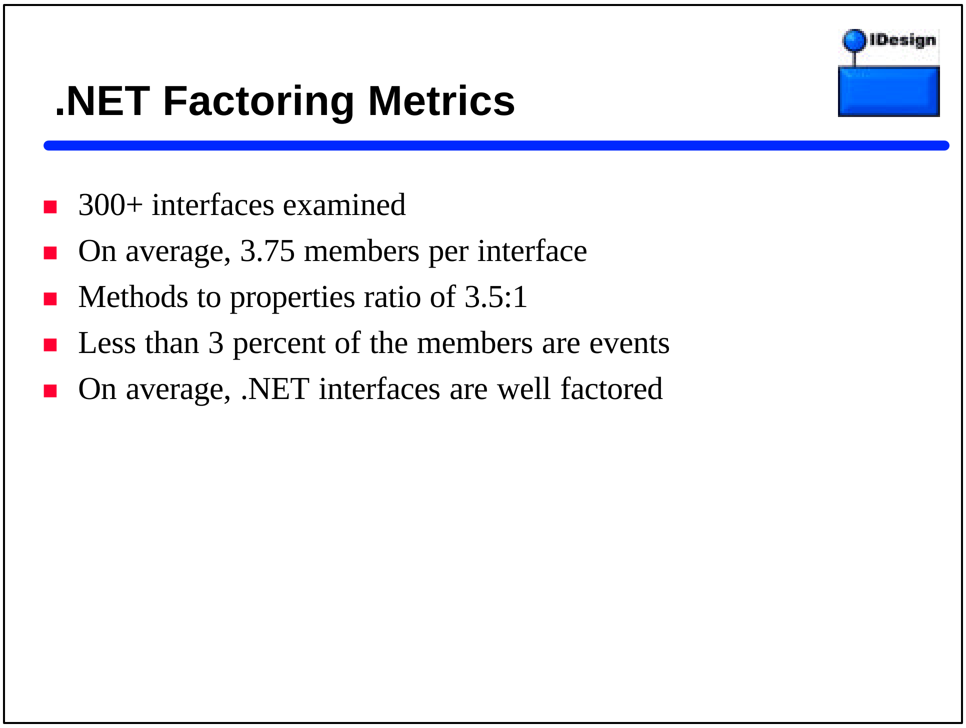
```
public interface IMammal
{
   void ShedFur();
   void Lactate();
}
public interface IDog : IMammal
{
   void  Fetch();
   void  Bark();
}
public interface ICat : IMammal
{
   void  Purr();
   void  CatchMouse();
}
```

Interface Factoring Metrics

- Interface factoring results in interfaces with fewer members
- Balance out two counter forces
  - Too many granular interfaces Vs few complex, poorly factored interfaces
- Just one member is possible, but avoid it
  - Dull facet
  - Too many parameters
  - Too coarse: should be factored into several methods
  - Refactor into an existing interface
- Optimal number 3 to 5
- No more than 20 (12)

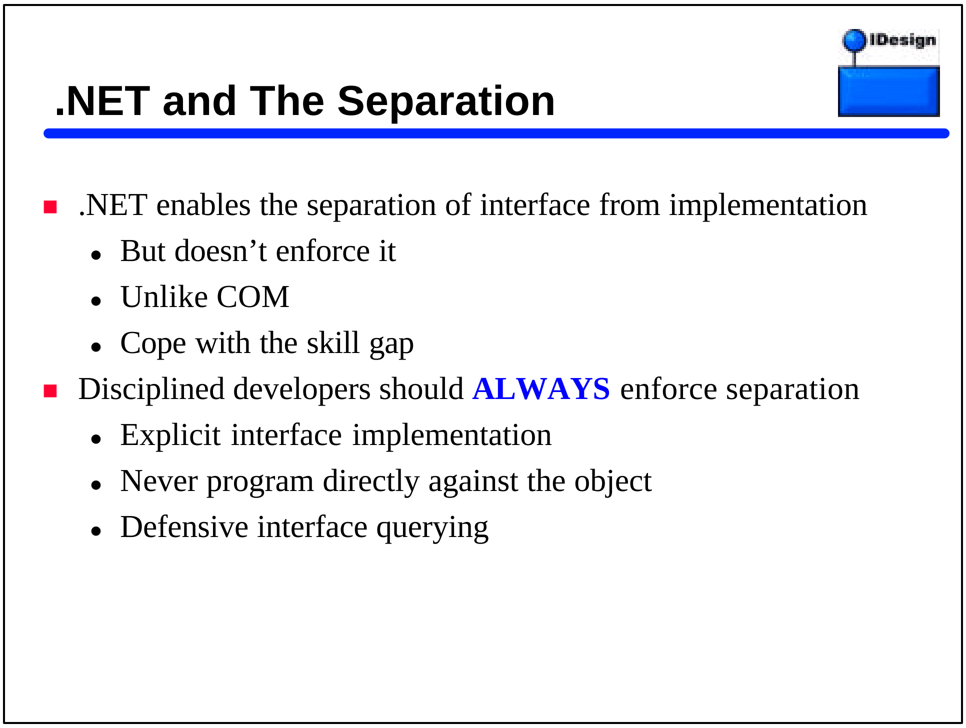
Interface Factoring Metrics

- Ratio of methods, properties and events
  - Interfaces should have more methods than properties
  - Just-enough-encapsulation
  - Ratio of at least 2:1
  - Exception is interfaces with properties only
    - Should have no methods
  - Avoid defining events



.NET Factoring Metrics

- 300+ interfaces examined
- On average, 3.75 members per interface
- Methods to properties ratio of 3.5:1
- Less than 3 percent of the members are events
- On average, .NET interfaces are well factored



.NET and The Separation

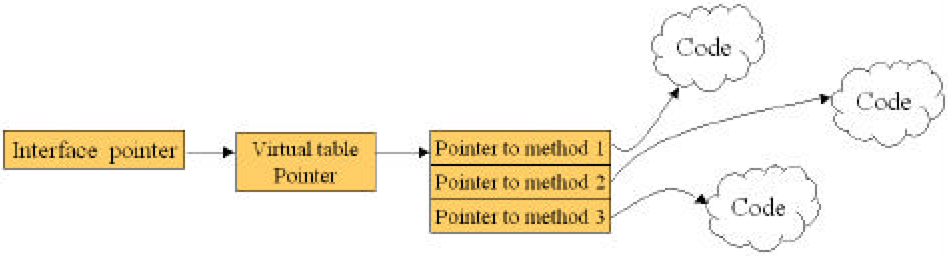
- .NET enables the separation of interface from implementation
  - But doesn’t enforce it
  - Unlike COM
  - Cope with the skill gap
- Disciplined developers should **ALWAYS** enforce separation
  - Explicit interface implementation
  - Never program directly against the object
  - Defensive interface querying

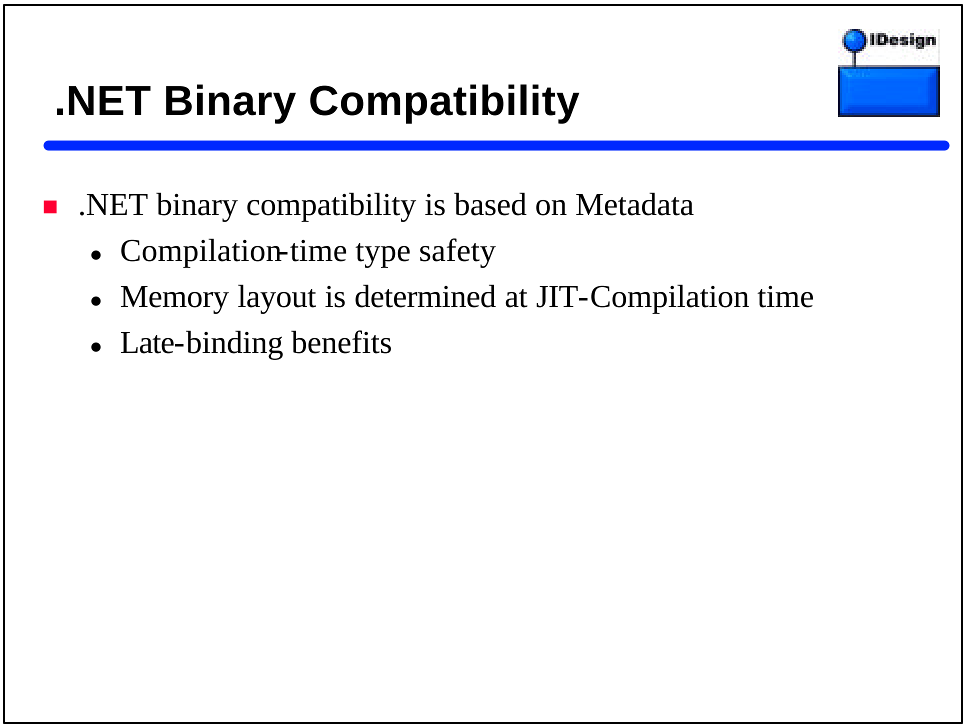
Binary Compatibility

- Traditional OO requires clients and servers to be in one monolithic application
  - Compiler bakes into client code address of server entry points
- CO packages code into binary building blocks
  - Replace and plug new binary versions of server
  - Implies binary compatibility between client and server
  - Client must interact at run time with exactly what it expects in binary layout in memory

Binary Compatibility

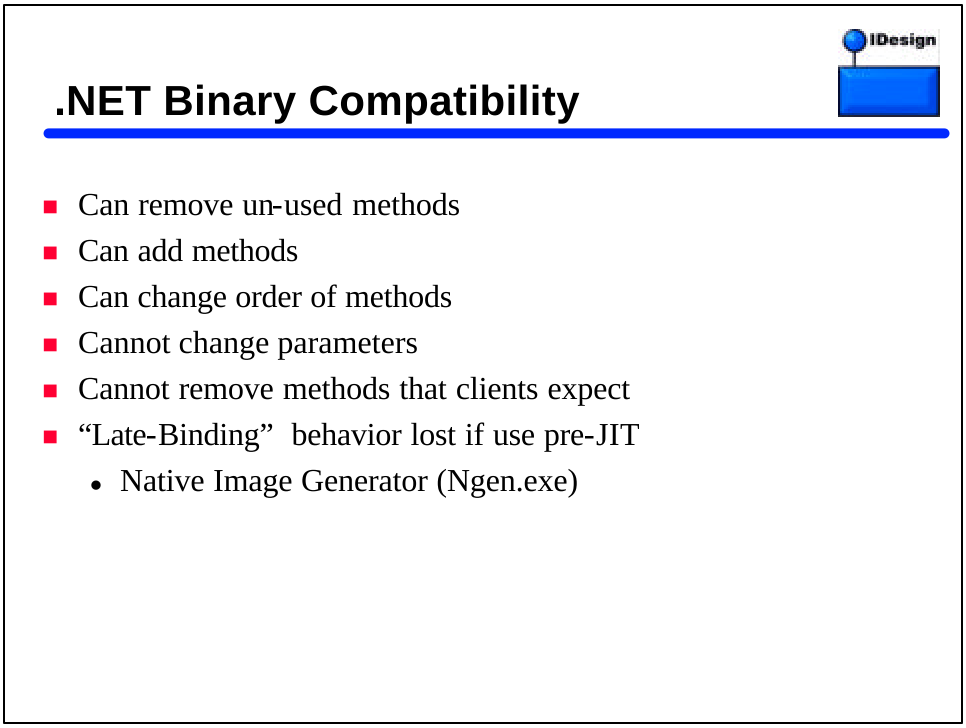
- Binary compatibility is the basis for the contract
- Client compiled once against interface or class definition
  - As long as contract maintained, server-side implementation can change
- COM binary compatibility is based on V-Tables
  - Memory layout





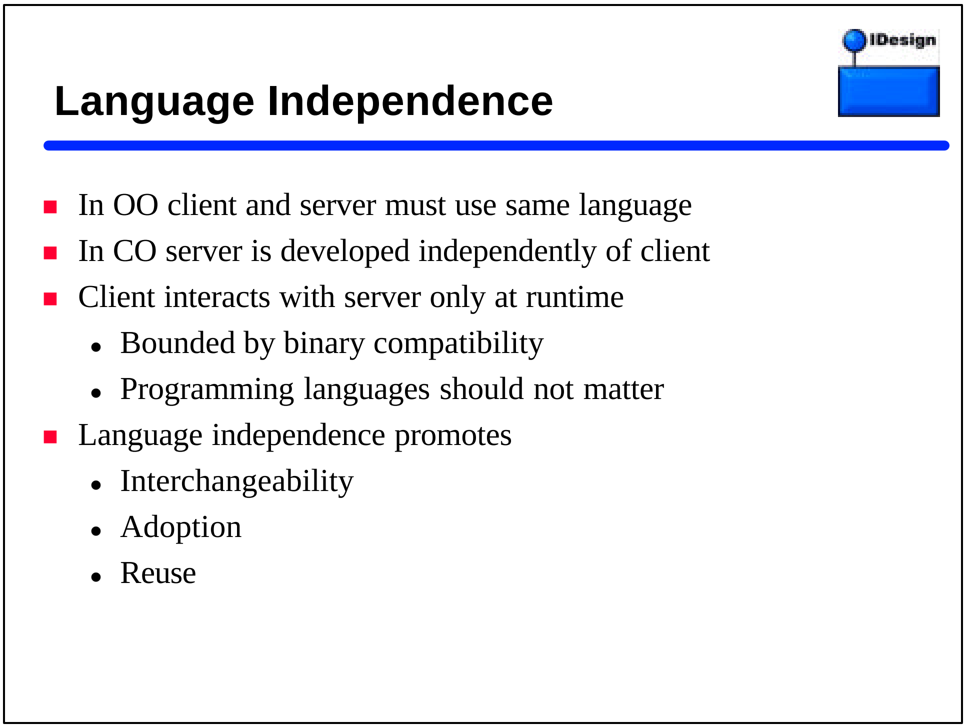
.NET Binary Compatibility

- .NET binary compatibility is based on Metadata
  - Compilation-time type safety
  - Memory layout is determined at JIT-Compilation time
  - Late-binding benefits



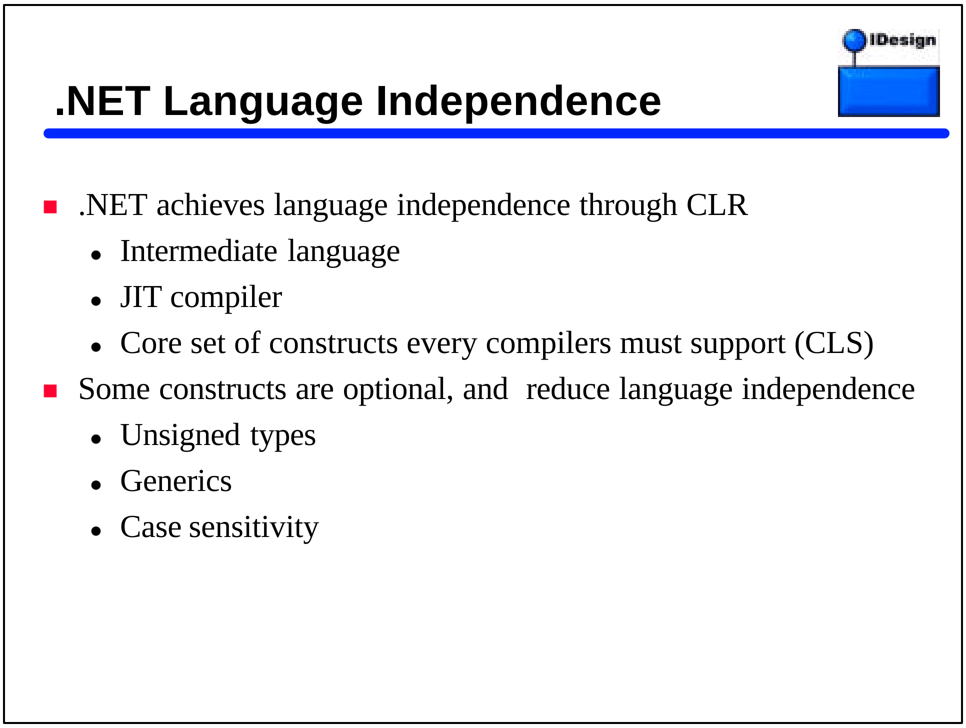
.NET Binary Compatibility

- Can remove un-used methods
- Can add methods
- Can change order of methods
- Cannot change parameters
- Cannot remove methods that clients expect
- “Late-Binding” behavior lost if use pre-JIT
  - Native Image Generator (Ngen.exe)



Language Independence

- In OO client and server must use same language
- In CO server is developed independently of client
- Client interacts with server only at runtime
  - Bounded by binary compatibility
  - Programming languages should not matter
- Language independence promotes
  - Interchangeability
  - Adoption
  - Reuse



.NET Language Independence

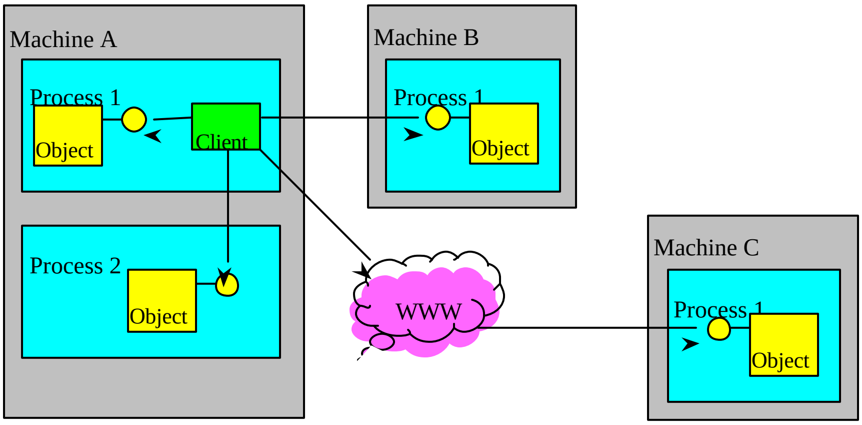
- .NET achieves language independence through CLR
  - Intermediate language
  - JIT compiler
  - Core set of constructs every compilers must support (CLS)
- Some constructs are optional, and reduce language independence
  - Unsigned types
  - Generics
  - Case sensitivity

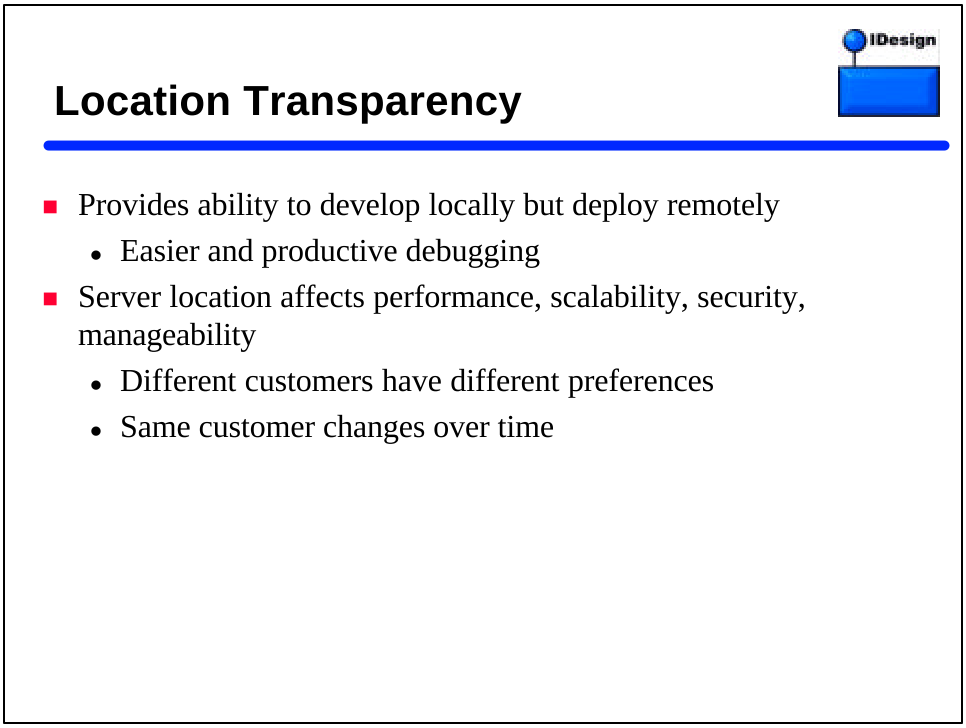
Location Transparency

- Client and component can be
  - In same process
  - Different processes on same machine
  - Different machines on network
  - Across the Internet
- Client code should be independent of actual location of object
  - Nothing pertaining to where the object executes
  - Same client code should handle all cases of object location
- Client should be able to insist on a specific location
- Ultimately, server-side location transparency is impossible

Location Transparency

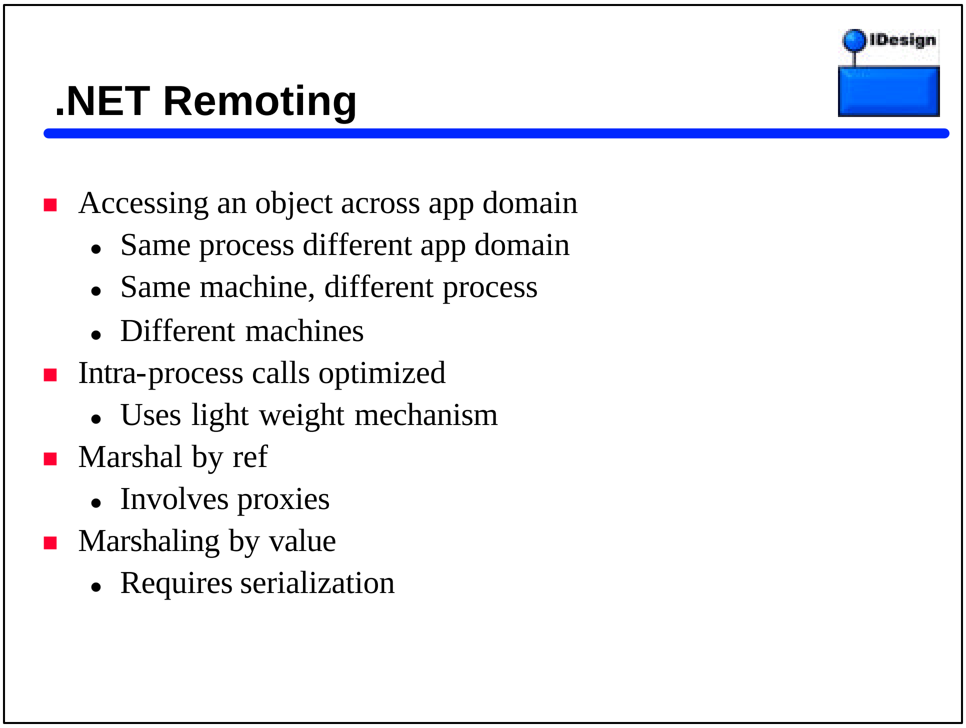
- Same client code handles all locations





Location Transparency

- Provides ability to develop locally but deploy remotely
  - Easier and productive debugging
- Server location affects performance, scalability, security, manageability
  - Different customers have different preferences
  - Same customer changes over time

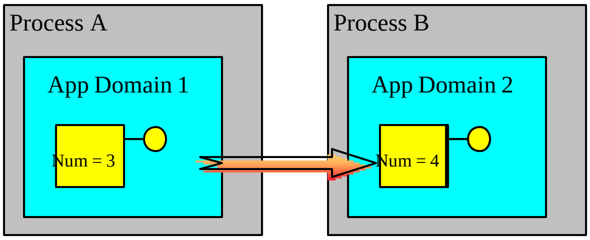


.NET Remoting

- Accessing an object across app domain
  - Same process different app domain
  - Same machine, different process
  - Different machines
- Intra-process calls optimized
  - Uses light weight mechanism
- Marshal by ref
  - Involves proxies
- Marshaling by value
  - Requires serialization

Marshal by Value

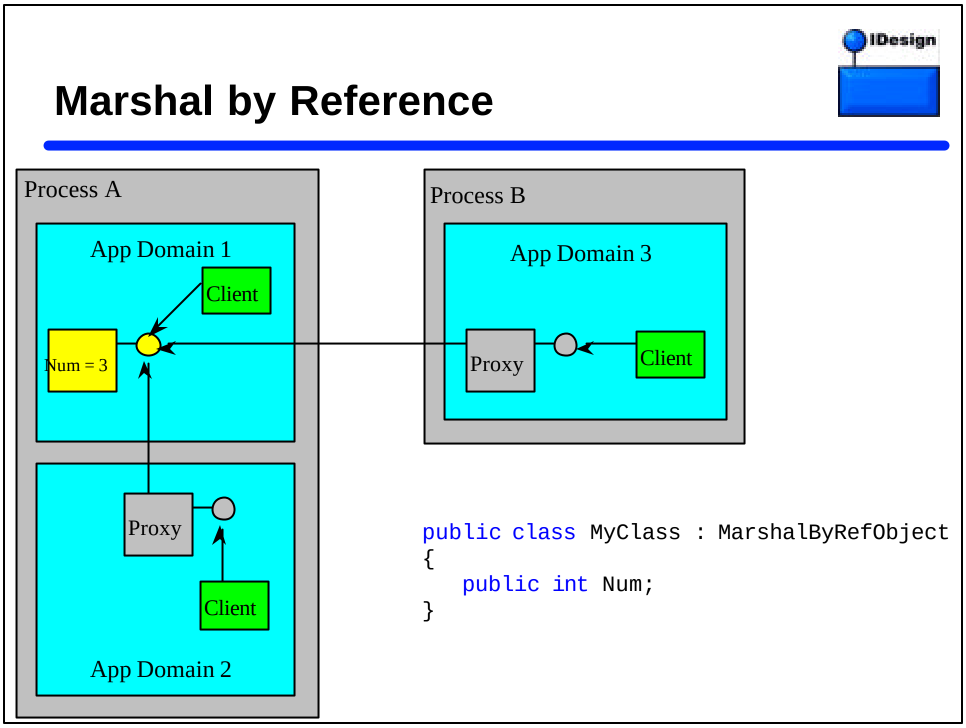
- Once marshaled, the two copies are distinct, and change state independently



```
[Serializable]
public class MyClass
{
   public int Num;
}
```

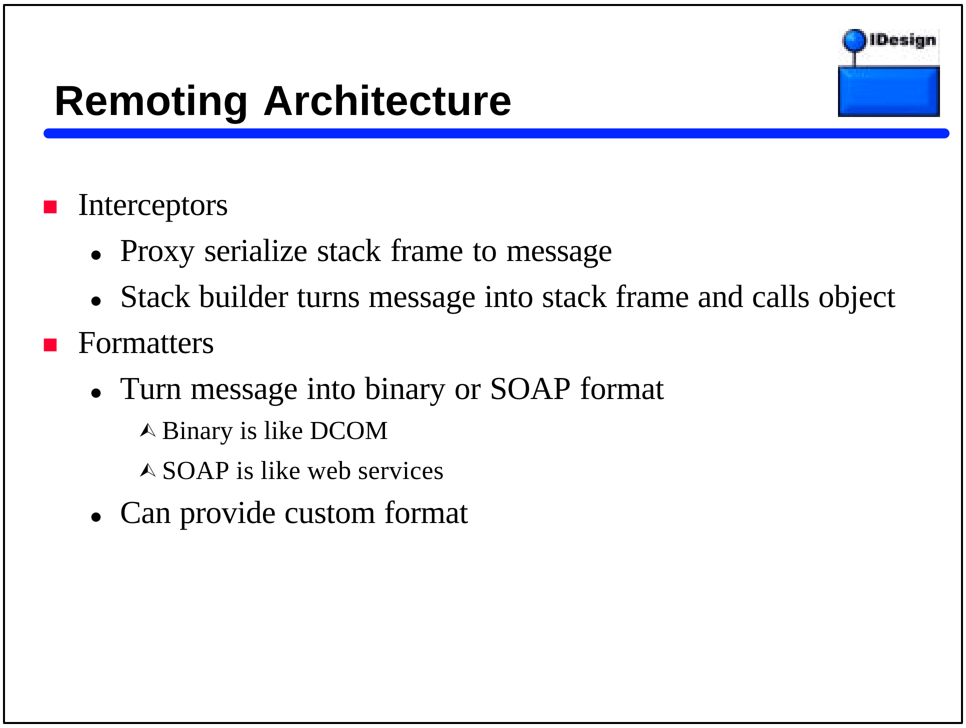
Marshal by Reference

- All access to object across app domain done via proxies
  - MBR across process and machine boundary as well
- All references point to same object
- Class must derive directly or indirectly from **`MarshalByRefObject`**
- If object is serializable as well, still marshaled by ref
- Intra-app domain calls use direct access



Marshal by Reference

```
public class MyClass : MarshalByRefObject
{
   public int Num;
}
```



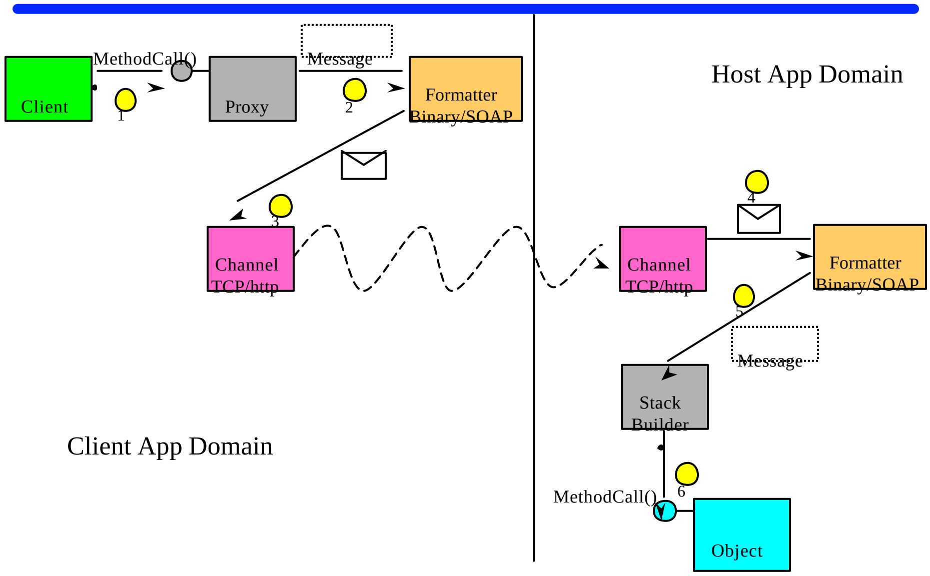
Remoting Architecture

- Interceptors
  - Proxy serialize stack frame to message
  - Stack builder turns message into stack frame and calls object
- Formatters
  - Turn message into binary or SOAP format
    - Binary is like DCOM
    - SOAP is like web services
  - Can provide custom format

Remoting Architecture

- Channels
  - Dispatch message using TCP/HTTP transport protocols
  - Can provide custom channel
  - Object can accept calls on multiple registered channels
- Can combine any format with any channel
- Almost all points in the architecture provide hooks and sinks
  - Extensibility
  - Security
  - Proprietary

Remoting Architecture

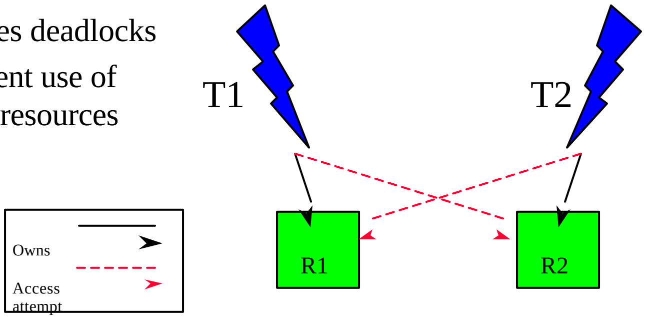


Location Transparency in .NET Remoting

- True location transparency only with **`new`**
  - Config file required
  - Registration sets up channels and identifies types to load remotely
- **`GetObject()`** does not require object registration
  - Only server activation
  - No location transparency
- Leasing and sponsorship requires client programming

Concurrency Management

- Component can be used by multiple threads
  - Vendor must assume it will
  - Vendor must provide synchronization
- Component can contain its own lock
  - Promotes deadlocks
  - Inefficient use of system resources

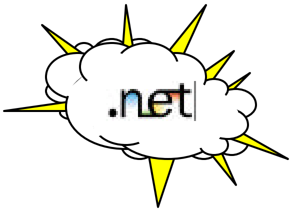


Concurrency Management

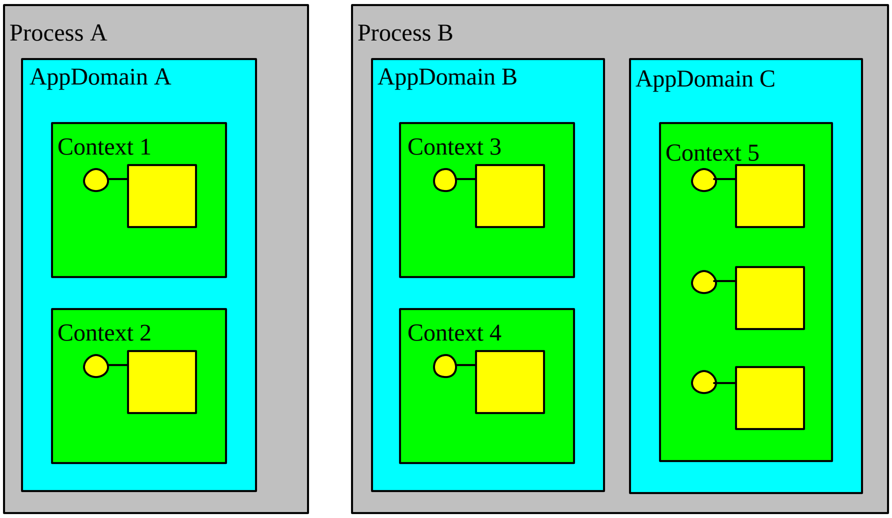
- The component technology must provide concurrency management service
  - Participate in application-wide synchronization mechanism
  - Even when components developed separately

.NET Concurrency Management

- .NET provides concurrency management via synchronization domains
  - Only for context-bound objects
  - Undocumented/poorly documented out of the box
- .NET supports manual synchronization

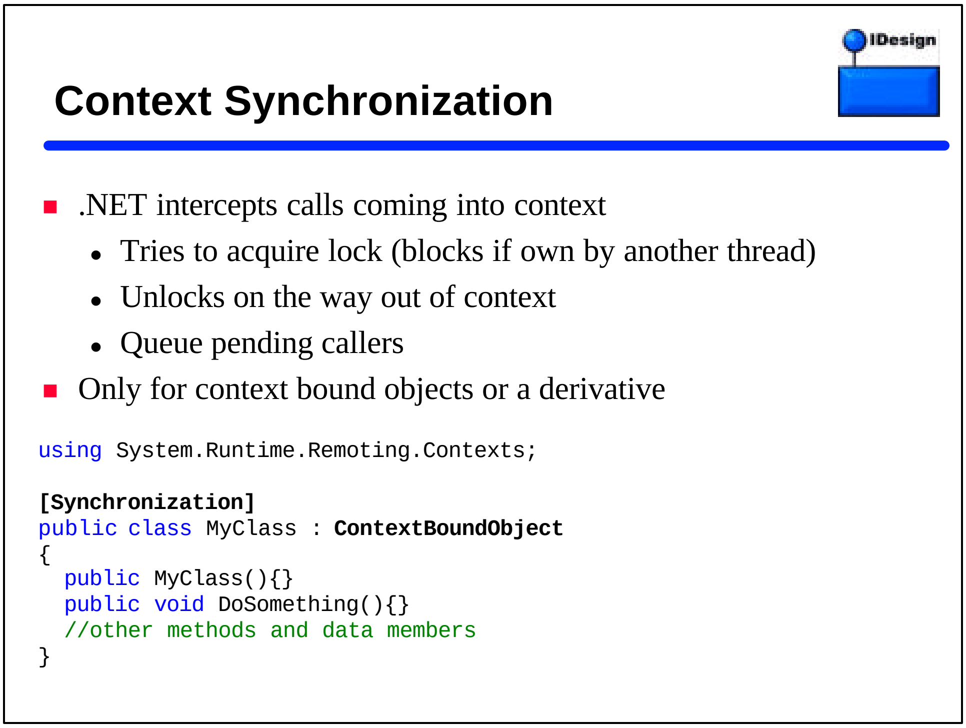


App Domain & Context



Context Synchronization

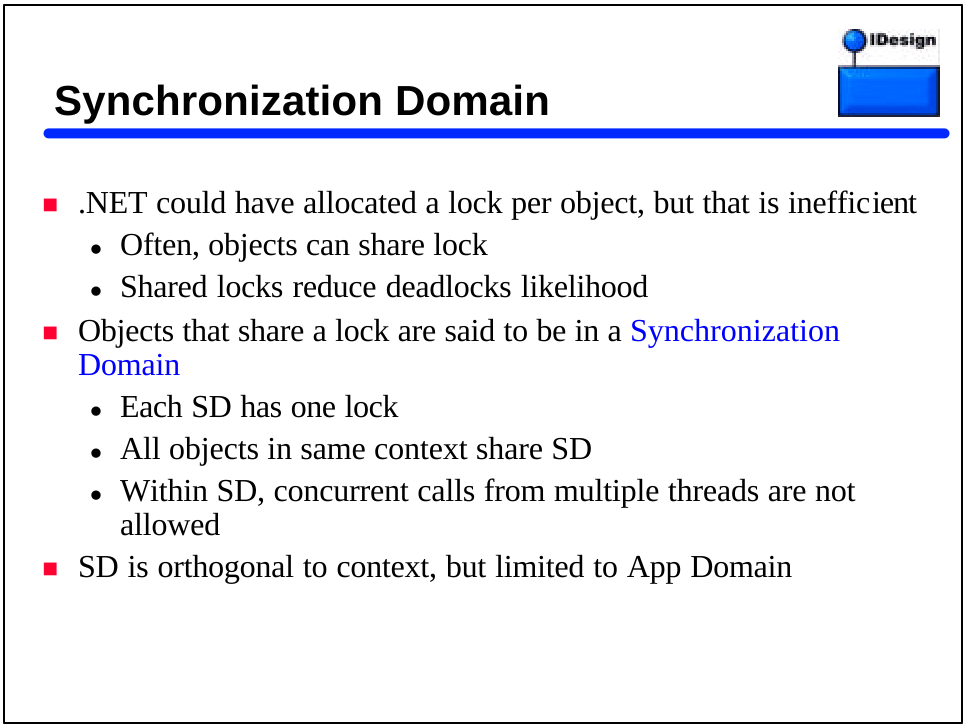
- **`[Synchronization]`** attribute
  - **`System.Runtime.Remoting.Contexts`**
  - Only for context-bound objects
- .NET associates object with a lock
  - Locks whole object during access
- Easiest synchronization mechanism to use
- A modern synchronization option that formally eliminate synchronization problems and the developer’s need to code around them



Context Synchronization

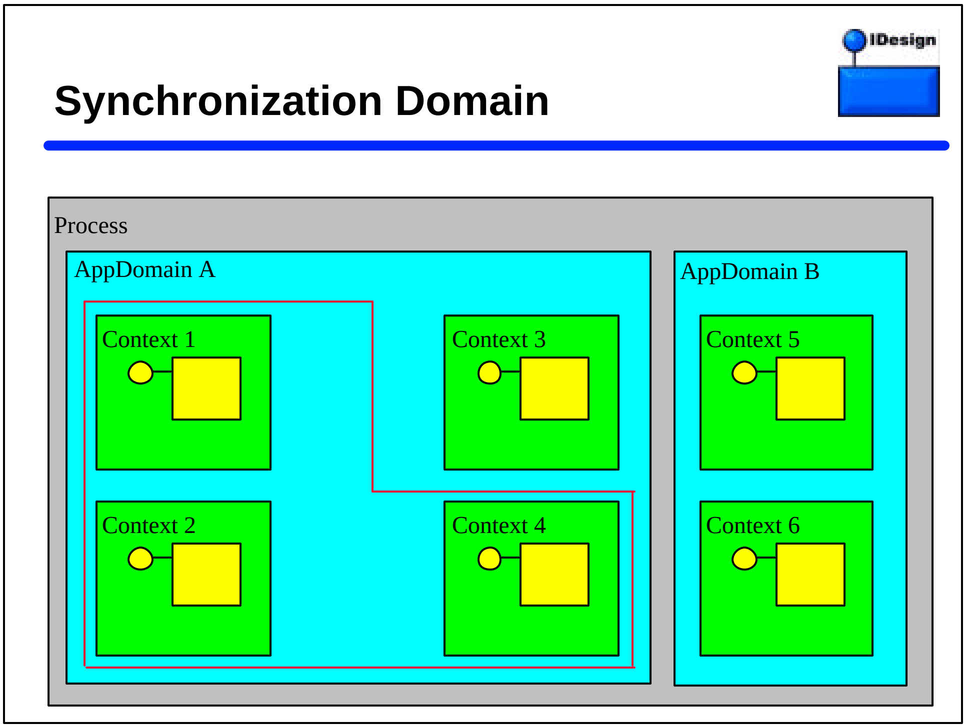
- .NET intercepts calls coming into context
  - Tries to acquire lock (blocks if own by another thread)
  - Unlocks on the way out of context
  - Queue pending callers
- Only for context bound objects or a derivative

```
using System.Runtime.Remoting.Contexts;

[Synchronization]
public class MyClass : ContextBoundObject
{
  public MyClass(){}
  public void DoSomething(){}
  //other methods and data members
}
```

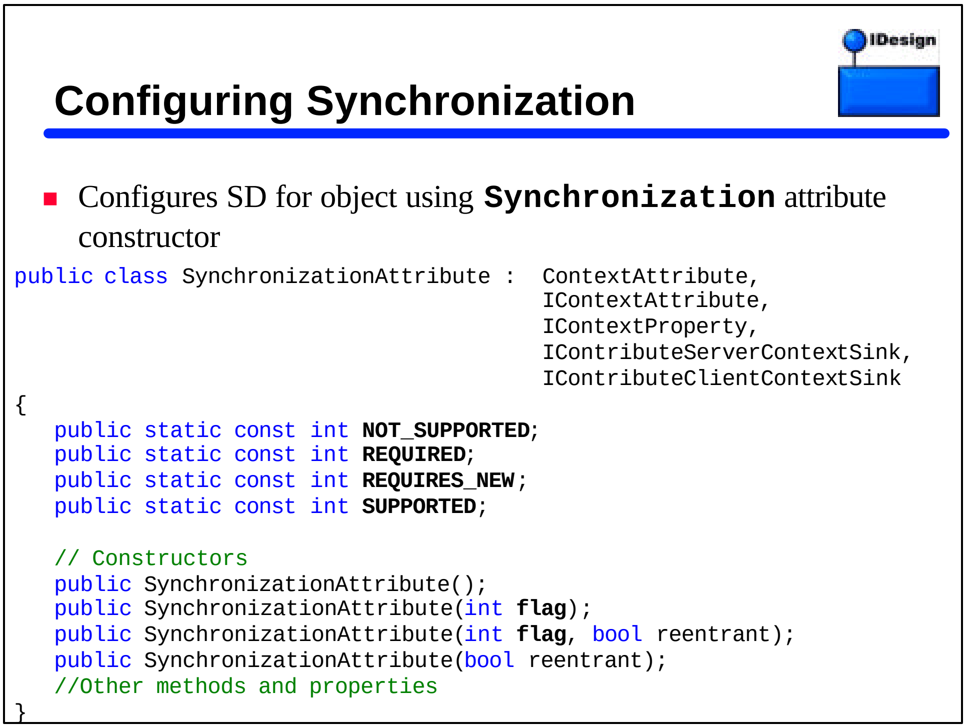


Synchronization Domain

- .NET could have allocated a lock per object, but that is inefficient
  - Often, objects can share lock
  - Shared locks reduce deadlocks likelihood
- Objects that share a lock are said to be in a Synchronization Domain
  - Each SD has one lock
  - All objects in same context share SD
  - Within SD, concurrent calls from multiple threads are not allowed
- SD is orthogonal to context, but limited to App Domain



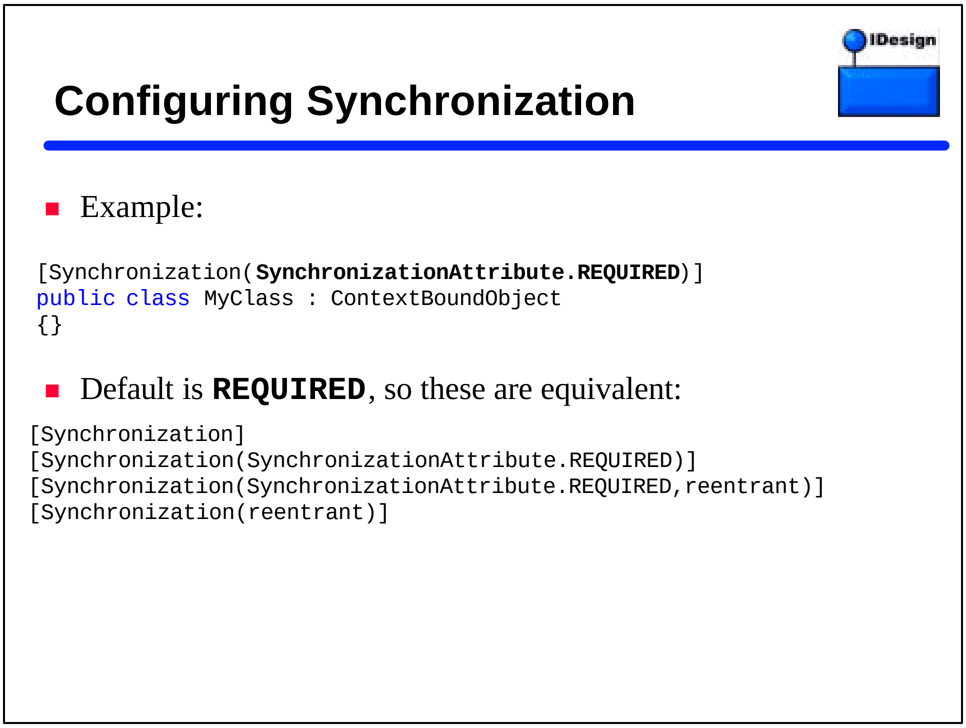
Synchronization Domain



Configuring Synchronization

- Configures SD for object using **`Synchronization`** attribute constructor

```
public class SynchronizationAttribute :  ContextAttribute,
                                         IContextAttribute,
                                         IContextProperty,
                                         IContributeServerContextSink,
                                         IContributeClientContextSink
{
   public static const int NOT_SUPPORTED;
   public static const int REQUIRED;
   public static const int REQUIRES_NEW;
   public static const int SUPPORTED;

   // Constructors
   public SynchronizationAttribute();
   public SynchronizationAttribute(int flag);
   public SynchronizationAttribute(int flag, bool reentrant);
   public SynchronizationAttribute(bool reentrant);
   //Other methods and properties
}
```



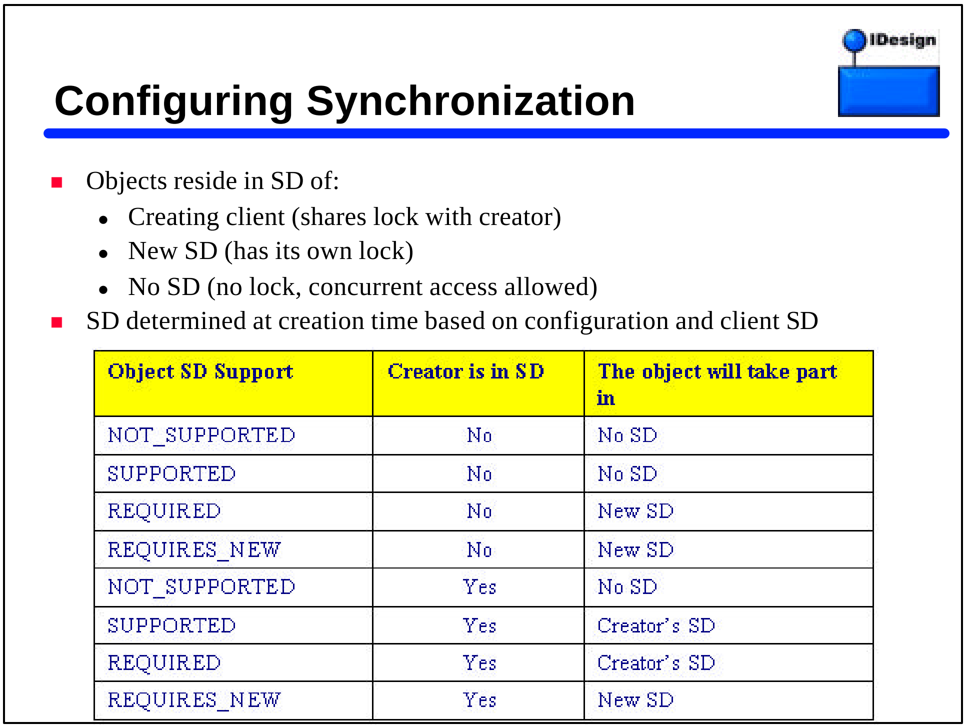
Configuring Synchronization

- Example:

```
[Synchronization(SynchronizationAttribute.REQUIRED)]
public class MyClass : ContextBoundObject
{}
```

- Default is **REQUIRED**, so these are equivalent:

```
[Synchronization]
[Synchronization(SynchronizationAttribute.REQUIRED)]
[Synchronization(SynchronizationAttribute.REQUIRED,reentrant)]
[Synchronization(reentrant)]
```



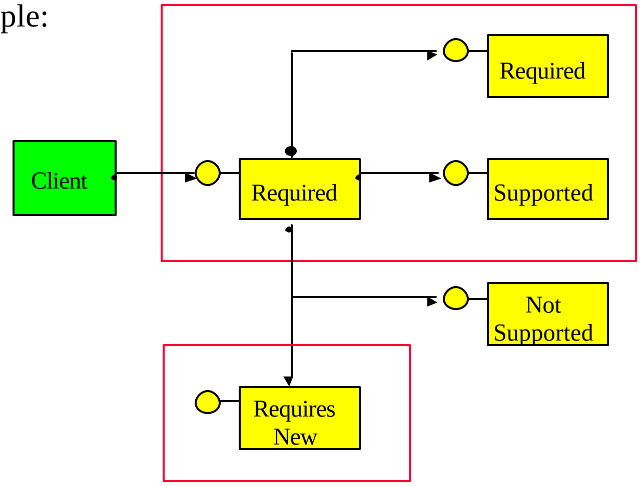
Configuring Synchronization

- Objects reside in SD of:
  - Creating client (shares lock with creator)
  - New SD (has its own lock)
  - No SD (no lock, concurrent access allowed)
- SD determined at creation time based on configuration and client SD

Object SD Support	Creator is in SD	The object will take part in
NOT_SUPPORTED	No	No SD
SUPPORTED	No	No SD
REQUIRED	No	New SD
REQUIRES_NEW	No	New SD
NOT_SUPPORTED	Yes	No SD
SUPPORTED	Yes	Creator's SD
REQUIRED	Yes	Creator's SD
REQUIRES_NEW	Yes	New SD

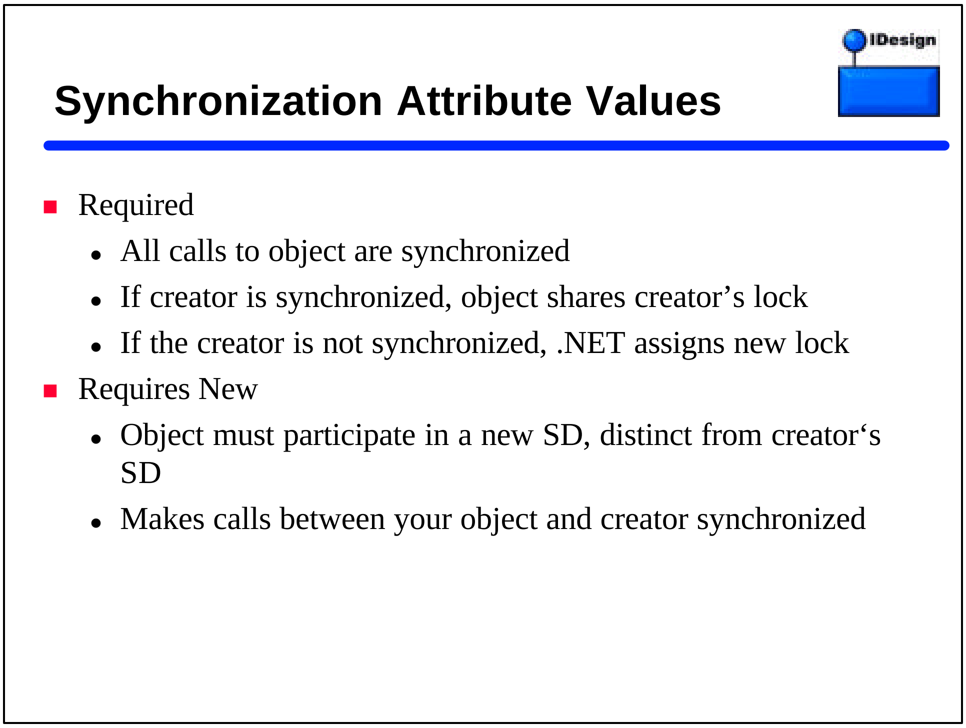
Configuring Synchronization

- Example:



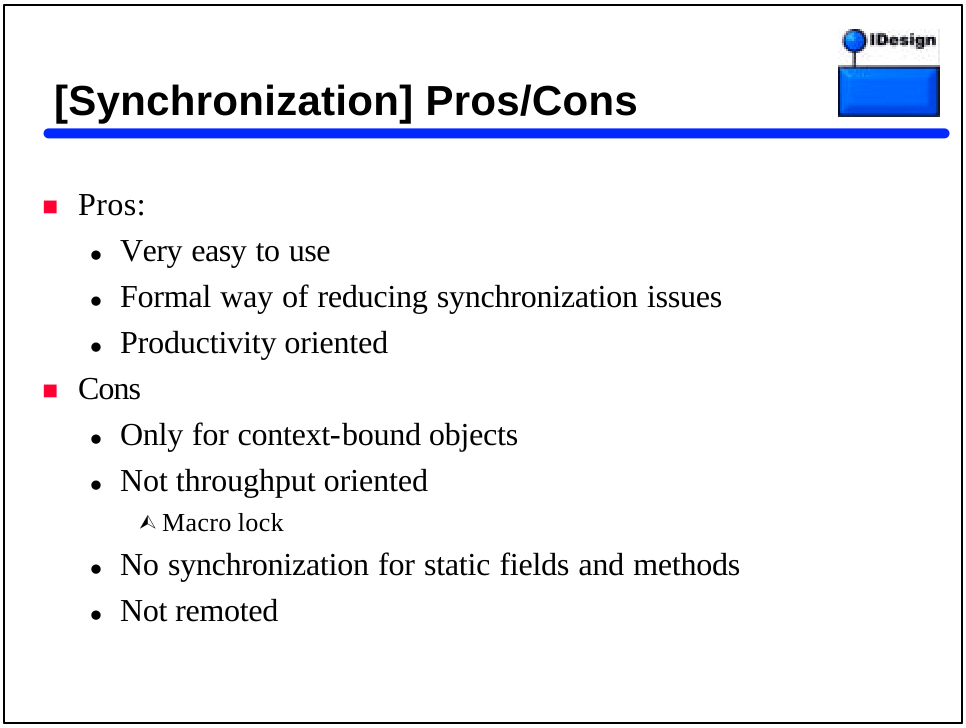
Synchronization Attribute Values

- Not Supported
  - Object never participates in synchronization, regardless of status of creator
  - Object must provide synchronization
  - Avoid it
- Supported
  - Object participates in SD if it exists
  - More difficult to code (have to handle case of concurrent access with no synchronization)
  - Used when need to propagate SD



Synchronization Attribute Values

- Required
  - All calls to object are synchronized
  - If creator is synchronized, object shares creator’s lock
  - If the creator is not synchronized, .NET assigns new lock
- Requires New
  - Object must participate in a new SD, distinct from creator‘s SD
  - Makes calls between your object and creator synchronized



[Synchronization] Pros/Cons

- Pros:
  - Very easy to use
  - Formal way of reducing synchronization issues
  - Productivity oriented
- Cons
  - Only for context-bound objects
  - Not throughput oriented
    - Macro lock
  - No synchronization for static fields and methods
  - Not remoted

Versioning Support

- Clients and components must evolve separately
  - Vendor should deploy new versions (or just fixes) without affecting existing clients
  - Client developers should deploy new versions and interact with older components
- The component technology should support versioning
  - Allow components to evolve along different paths
  - Allow for side-by-side deployment
  - Should detect incompatibility as soon as possible

.NET Versioning Support

- Assemblies can be private or shared
- A private assembly resides in the app directory
- A shared assembly is in a known location, called the global assembly cache (GAC)
- Shared assembly used for:
  - Sharing
  - Side-by-side execution

.NET Versioning Support

- Shared assemblies must have a unique name
  - Called **Strong Name**
- Strong name authenticates assembly's origin and identity
  - Shared assembly implies trust
- Strong name cannot be generated by a party other than the original publisher
- Strong name is based on public/private keys pair

.NET Versioning Support

- Digitally signs the assembly to verify origin
  - Encrypt manifest using the private key
  - Append signature to manifest
  - Incorporated public key into the assembly
- To verify authenticity
  - .NET loader generates the hash
  - Decrypts the manifest-stored hash
  - Compare
- Can only call signed assembly from within signed assemblies
  - Friendly name assemblies can call both

Resolving Version

- When trying to use a shared assembly
  - Possible many versions of the same assembly in the GAC
- Client always gets assembly with exact version match
  - Can provide custom policy
- Developers must be disciplined
  - Release procedures

Resolving Version

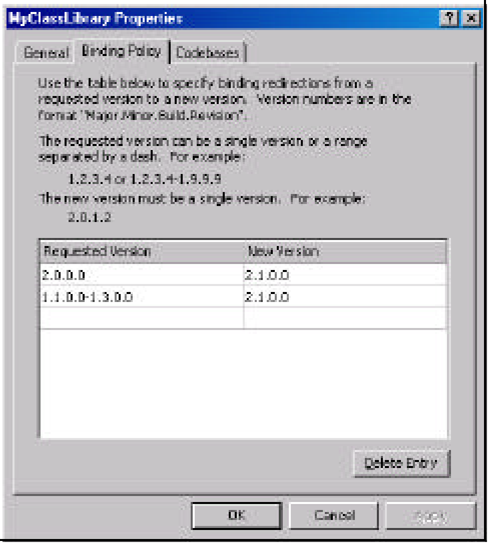
- Private assembly can be strongly named
- .NET ignores version of private assemblies with friendly name only
- .NET enforces version compatibility of private assembly with strong name

Custom Version Policy

- Application can provide version binding policy
  - Override default policy
  - For shared and private assemblies
- Can deploy machine-wide policy
- .NET configuration tool
  - MMC snap-in

Custom Version Binding

- Binding policy:



Use the table below to specify binding redirections from a requested version to a new version. Version numbers are in the format "Major.Minor.Build.Revision".

The requested version can be a single version or a range separated by a dash. For example:

1.2.3.4 or 1.2.3.4-1.9.9.9

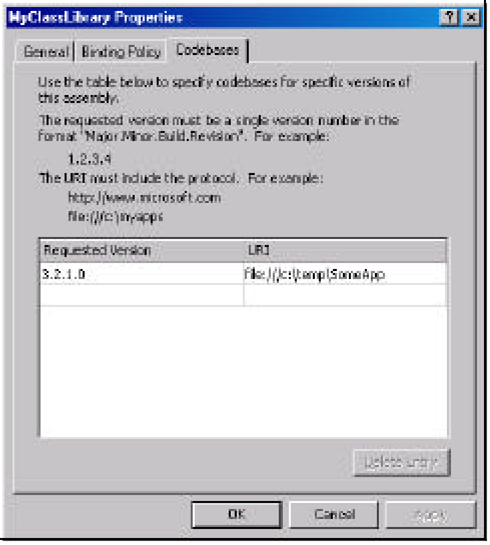
The new version must be a single version. For example:

2.0.1.2

Requested Version	New Version
2.0.0.0	2.1.0.0
1.1.0.0-1.3.0.0	2.1.0.0

Custom Version Binding

- Codebase policy:
  - Redirect to new location



Use the table below to specify codebases for specific versions of this assembly.

The requested version must be a single version number in the format 'Major.Minor.Build.Revision'. For example:

1.2.3.4

The URI must include the protocol. For example:

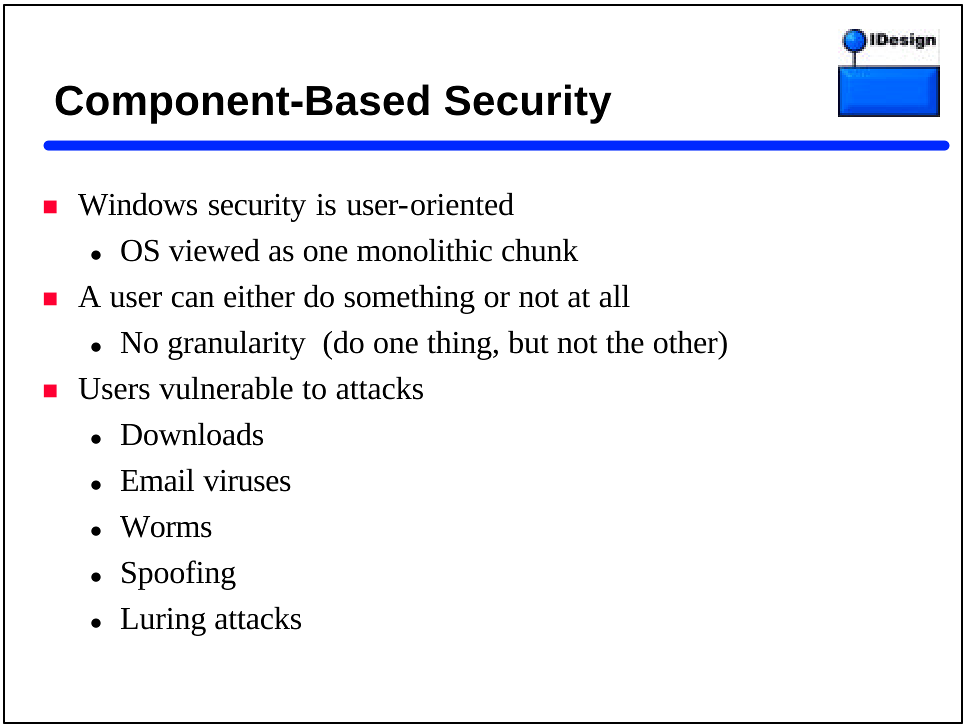
http://www.microsoft.com

file:///c:\myapps

Requested Version	URI
3.2.1.0	file:///c:\temp\SomeApp

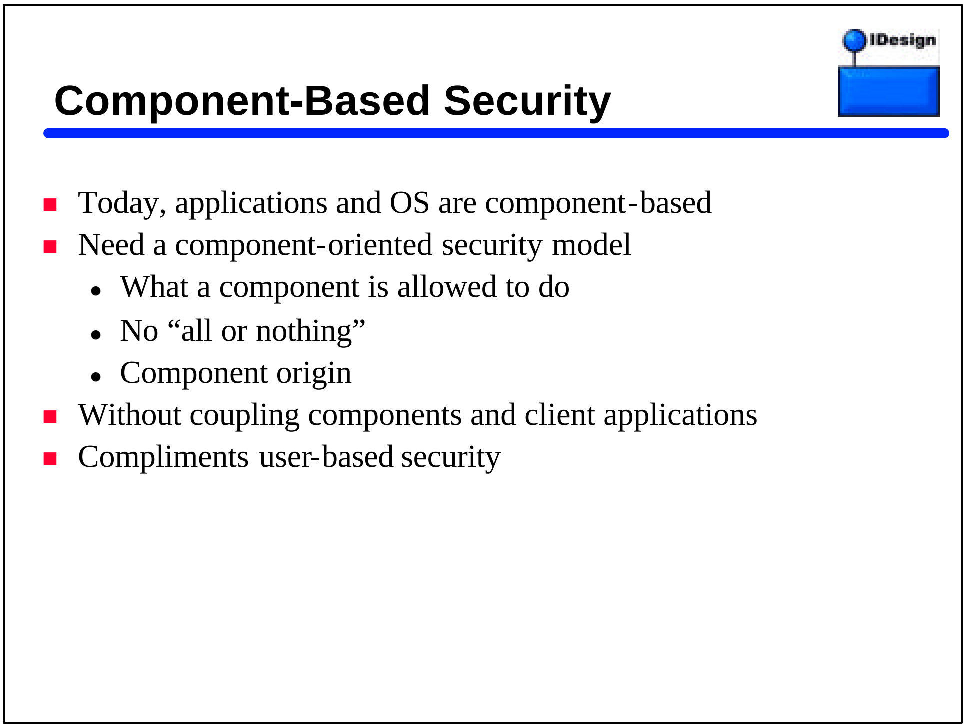
.NET Versioning Support

- Versioning support only for strongly-named assemblies
- Side-by-side in the GAC
- Default policy enforces compatibility
- Can deploy custom application policy
  - Typically by application vendor
- Can deploy custom machine-wide policy
  - Typically by component vendor



Component-Based Security

- Windows security is user-oriented
  - OS viewed as one monolithic chunk
- A user can either do something or not at all
  - No granularity (do one thing, but not the other)
- Users vulnerable to attacks
  - Downloads
  - Email viruses
  - Worms
  - Spoofing
  - Luring attacks

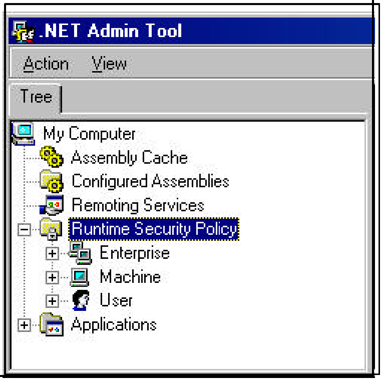


Component-Based Security

- Today, applications and OS are component-based
- Need a component-oriented security model
  - What a component is allowed to do
  - No “all or nothing”
  - Component origin
- Without coupling components and client applications
- Compliments user-based security

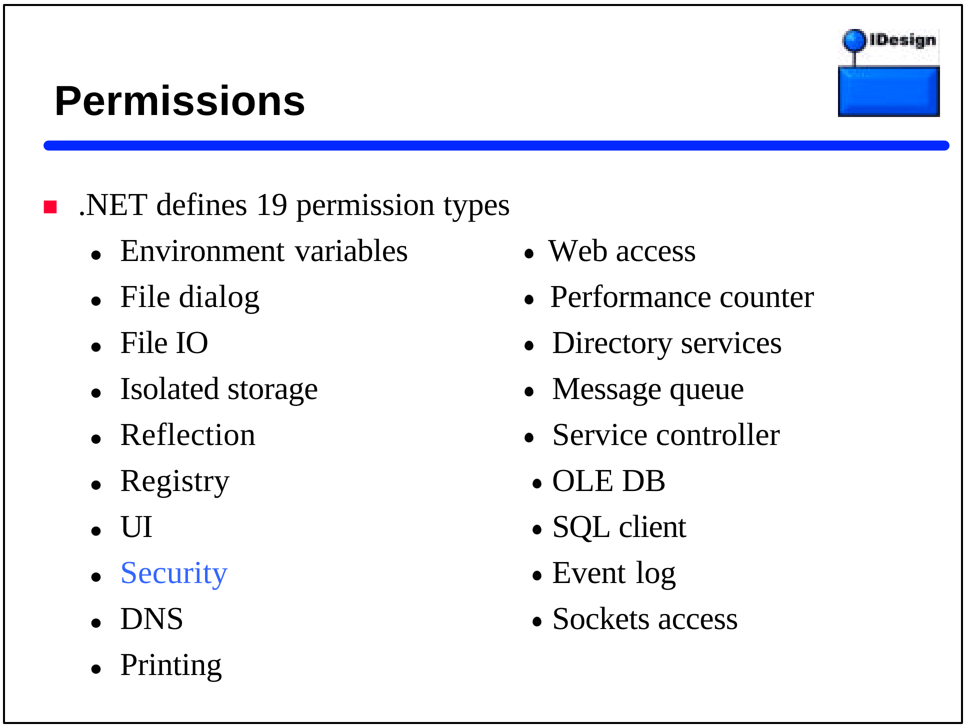
.NET Code-Access Security

- Intricate administrative permissions schema
- Programmatic permissions support
- Role-base security
  - Optional – custom principal



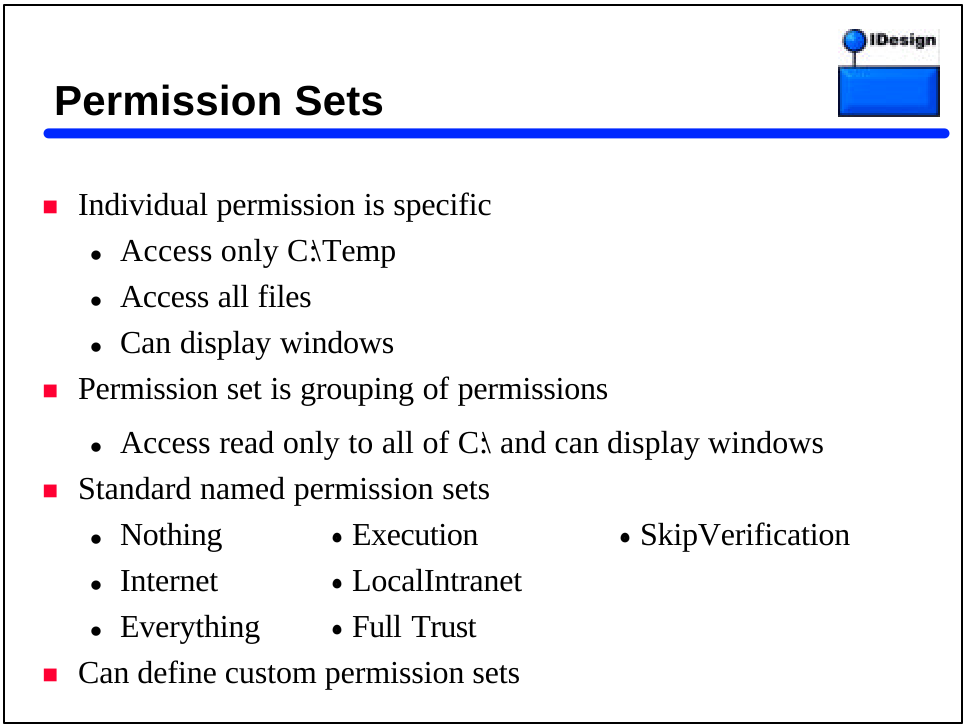
Security Permission

- An individual grant
  - Grants access to a resource
  - Perform operation
- Examples
  - File I/O permission
    - Read, write append data to a **specific** file
  - UI Permission
    - Accessing windows, top level windows, clipboard access
  - Reflection



Permissions

- .NET defines 19 permission types
  - Environment variables
  - File dialog
  - File IO
  - Isolated storage
  - Reflection
  - Registry
  - UI
  - Security
  - DNS
  - Printing
  - Web access
  - Performance counter
  - Directory services
  - Message queue
  - Service controller
  - OLE DB
  - SQL client
  - Event log
  - Sockets access



Permission Sets

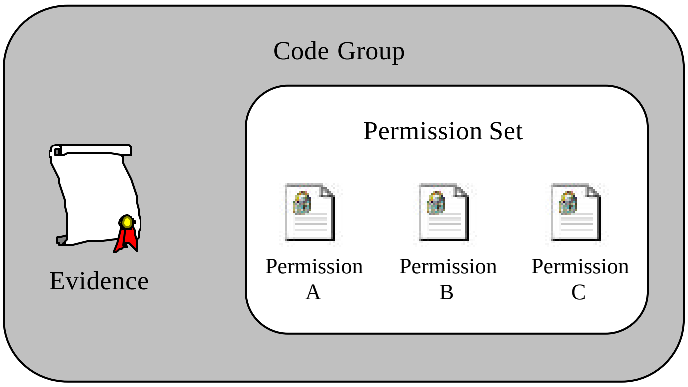
- Individual permission is specific
  - Access only C:\Temp
  - Access all files
  - Can display windows
- Permission set is grouping of permissions
  - Access read only to all of C:\ and can display windows
- Standard named permission sets
  - Nothing
  - Internet
  - Everything
  - Execution
  - LocalIntranet
  - Full Trust
  - SkipVerification
- Can define custom permission sets

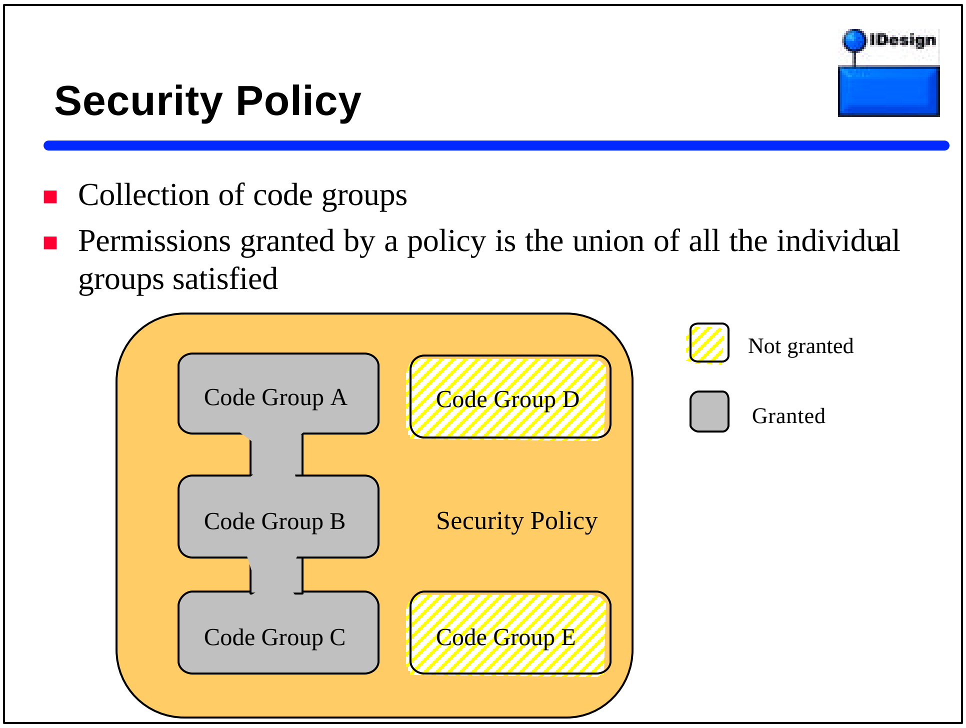
Security Evidence

- Permissions granted based on evidence
- Evidence is some form of proof assembly provides to substantiate identity
  - Origin-based evidences
  - Content-based evidences
- Origin-based evidence
  - Application Directory, Site, URL, Zone
- Content-based evidence
  - Strong name, Publisher certificate, Hash

Code Groups

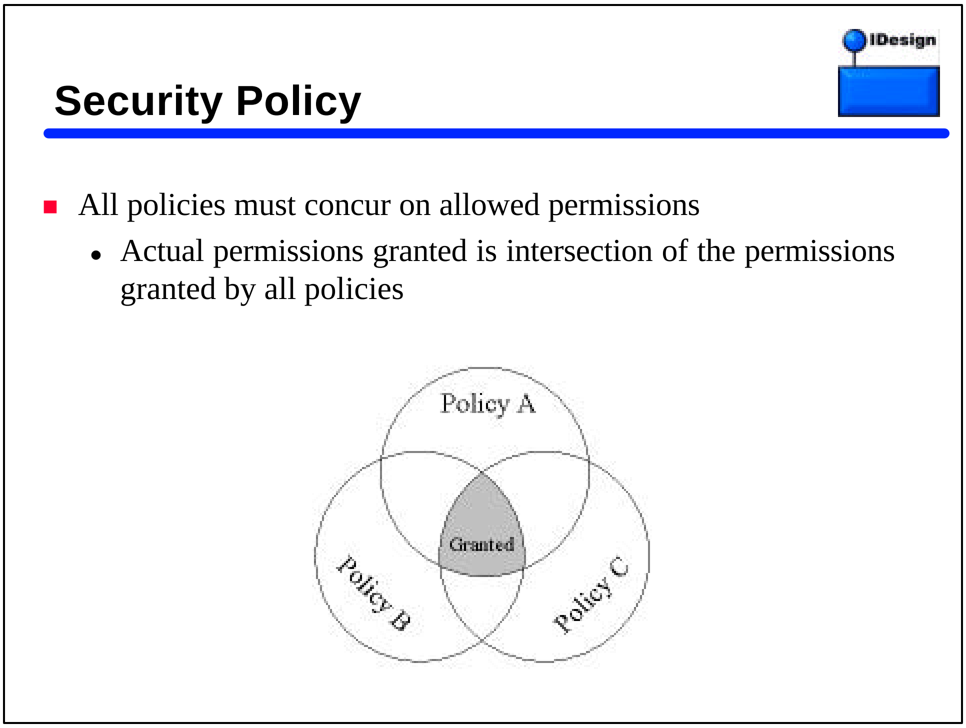
- Binding of a single permission set with a single evidence





Security Policy

- Collection of code groups
- Permissions granted by a policy is the union of all the individual groups satisfied

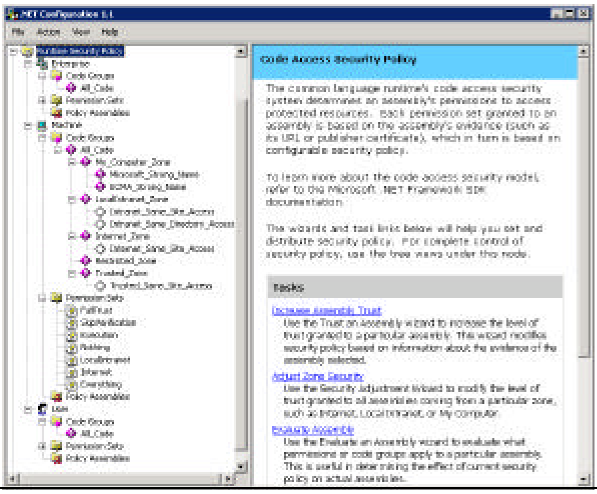


Security Policy

- All policies must concur on allowed permissions
  - Actual permissions granted is intersection of the permissions granted by all policies

Managing Security Policies

- Manage permission using .NET configuration tool



Code Access Security Policy

The common language runtime's code access security system determines an assembly's permissions to access protected resources. Each permission set granted to an assembly is based on the assembly's evidence (such as its URL or publisher certificate), which in turn is based on configurable security policy.

To learn more about the code access security model, refer to the Microsoft .NET Framework SDK documentation.

The wizards and tool links below will help you set and distribute security policy. For complete control of security policy, use the tree views under this node.

Tasks

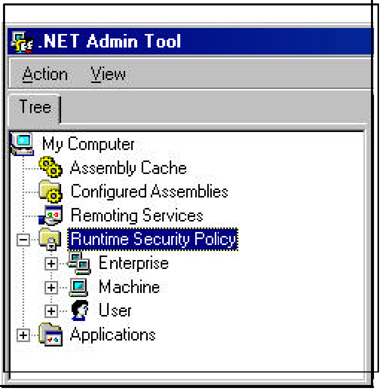
Increase Assembly Trust
Use the Trust an Assembly wizard to increase the level of trust granted to a particular assembly. This wizard modifies security policy based on information about the evidence of the assembly selected.

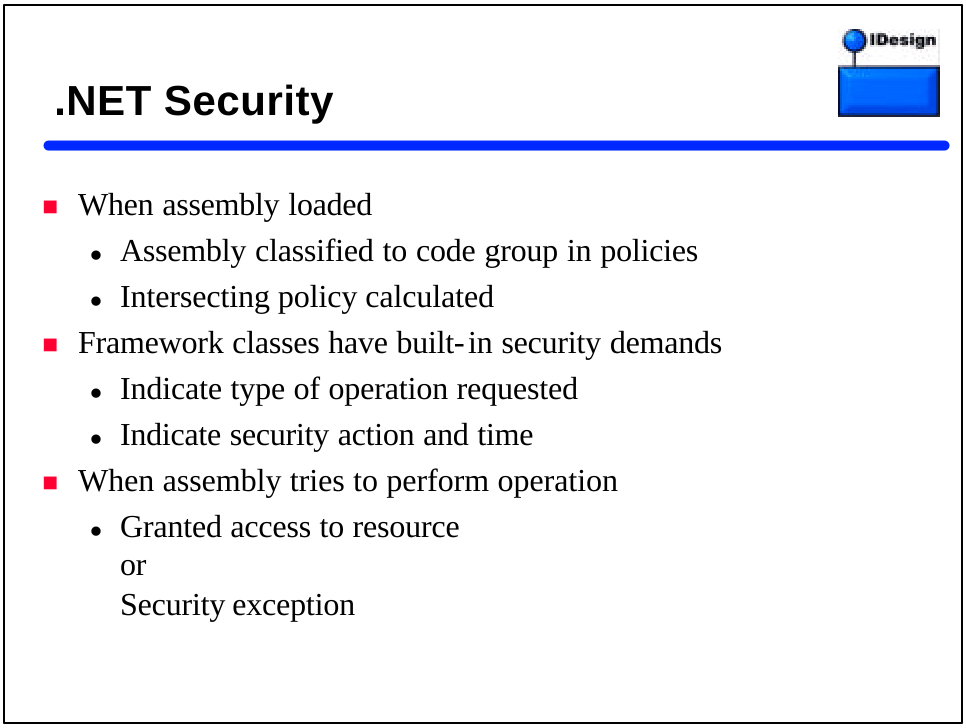
Adjust Zone Security
Use the Security Adjustment Wizard to modify the level of trust granted to all assemblies coming from a particular zone, such as Internet, Local Intranet, or My Computer.

Evaluate Assembly
Use the Evaluate an Assembly wizard to evaluate what permissions or code groups apply to a particular assembly. This is useful in determining the effect of current security policy on actual assemblies.

Security Policy

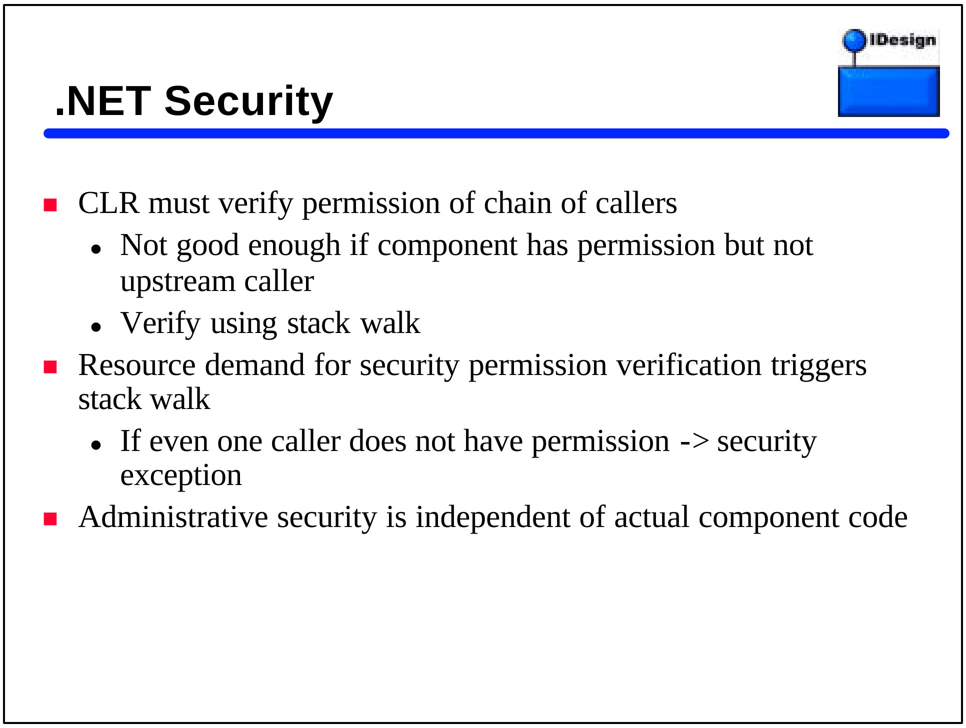
- .NET defines three policies
  - Enterprise
  - Machine
  - User
- Each policy defines code groups
- Each policy defines its own permission sets
- Can customize policies
  - Demo





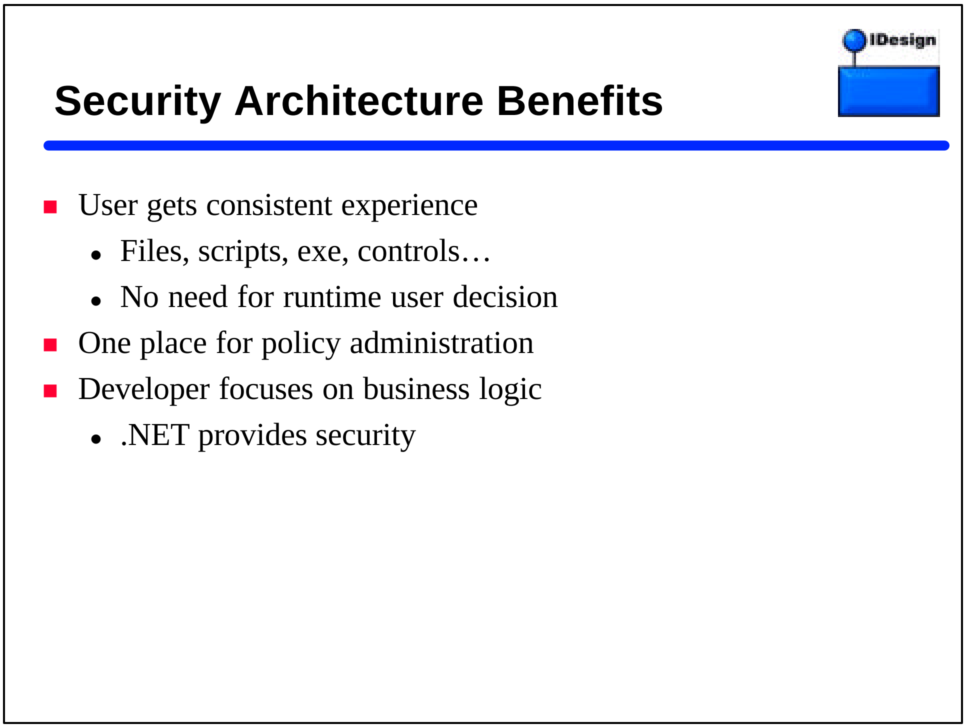
.NET Security

- When assembly loaded
  - Assembly classified to code group in policies
  - Intersecting policy calculated
- Framework classes have built-in security demands
  - Indicate type of operation requested
  - Indicate security action and time
- When assembly tries to perform operation
  - Granted access to resource
    or
    Security exception



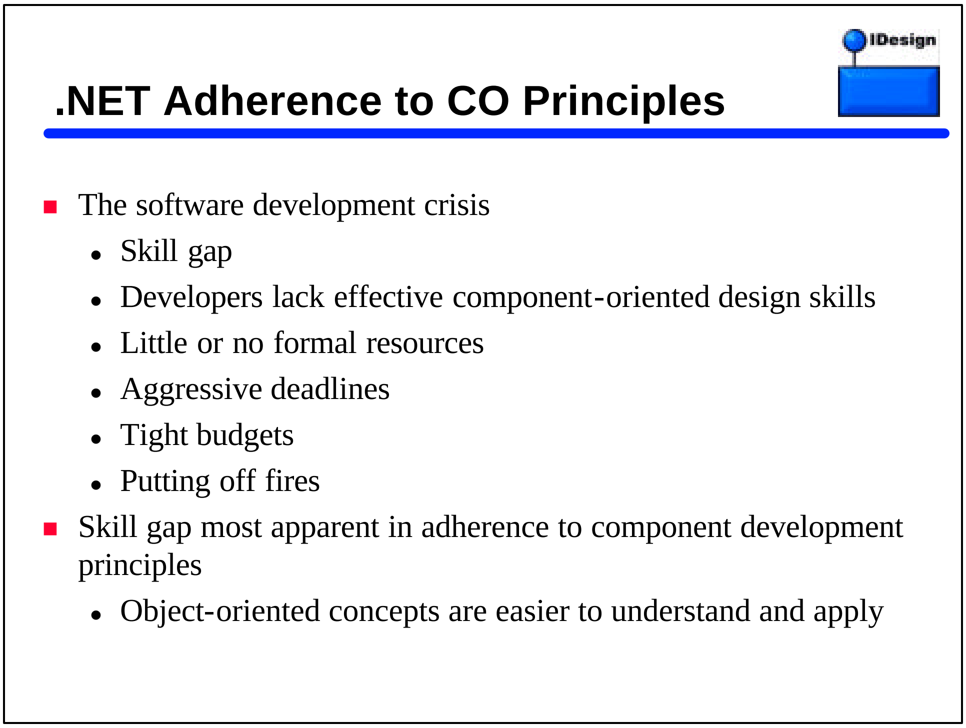
.NET Security

- CLR must verify permission of chain of callers
  - Not good enough if component has permission but not upstream caller
  - Verify using stack walk
- Resource demand for security permission verification triggers stack walk
  - If even one caller does not have permission -> security exception
- Administrative security is independent of actual component code



Security Architecture Benefits

- User gets consistent experience
  - Files, scripts, exe, controls…
  - No need for runtime user decision
- One place for policy administration
- Developer focuses on business logic
  - .NET provides security



.NET Adherence to CO Principles

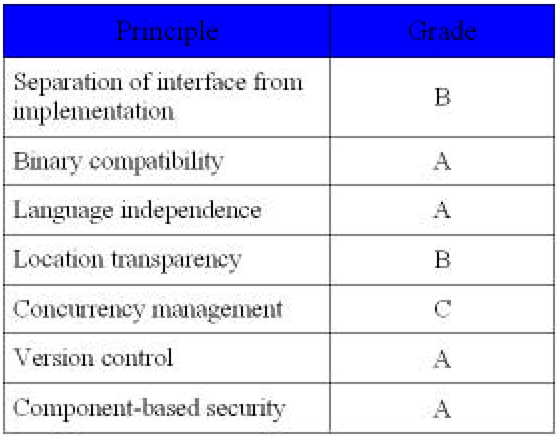
- The software development crisis
  - Skill gap
  - Developers lack effective component-oriented design skills
  - Little or no formal resources
  - Aggressive deadlines
  - Tight budgets
  - Putting off fires
- Skill gap most apparent in adherence to component development principles
  - Object-oriented concepts are easier to understand and apply

.NET Adherence to CO Principles

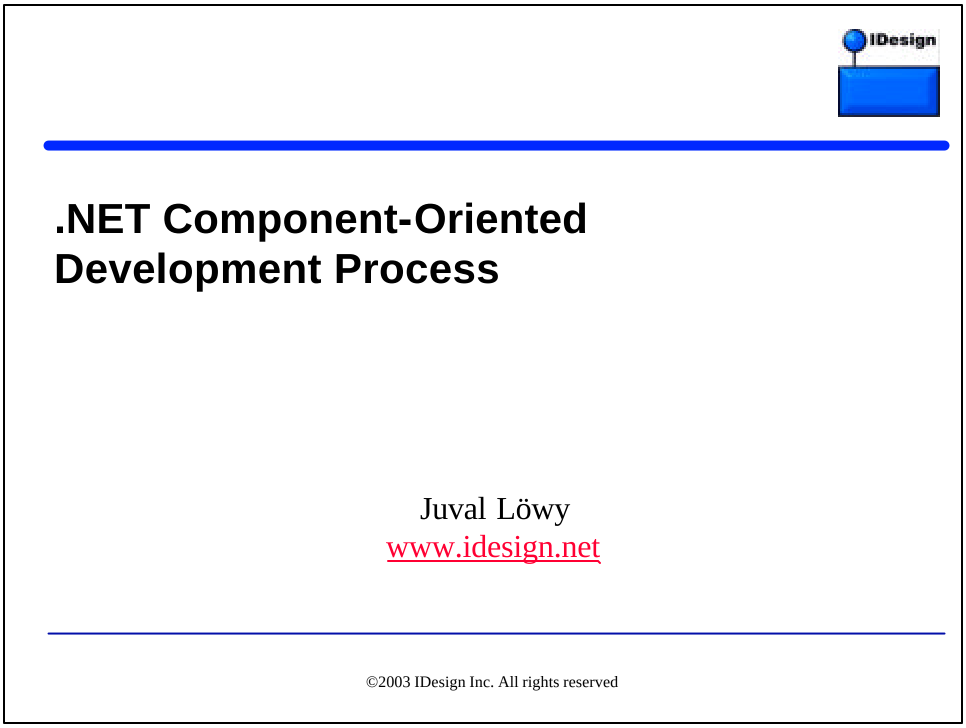
- Primary goal of .NET to simplify development and use of binary components
- .NET does not enforce some of the core principles
  - Separation of interface from implementation
  - Allows binary inheritance of implementation
- .NET enforces few of the concepts and enables the rest
  - Catering to both ends of the skill spectrum

.NET Adherence to CO Principles

- Report card

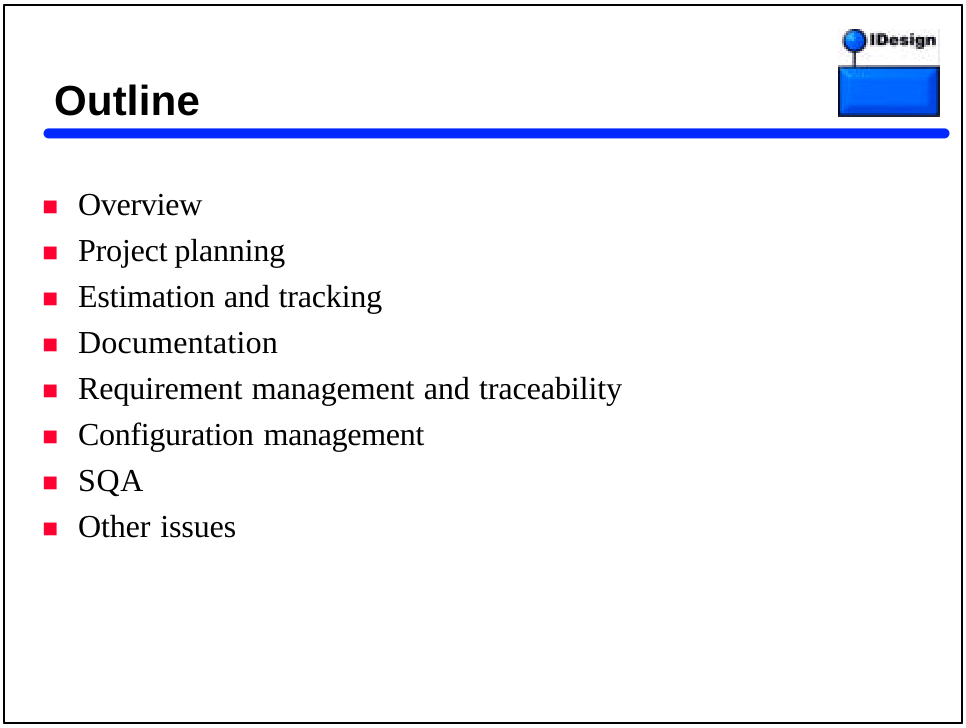


Principle	Grade
Separation of interface from implementation	B
Binary compatibility	A
Language independence	A
Location transparency	B
Concurrency management	C
Version control	A
Component-based security	A



.NET Component-Oriented Development Process

Juval Löwy
www.idesign.net



Outline

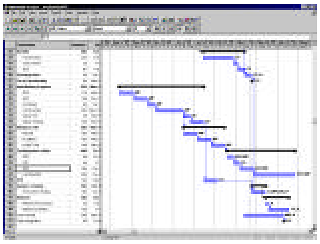
- Overview
- Project planning
- Estimation and tracking
- Documentation
- Requirement management and traceability
- Configuration management
- SQA
- Other issues

Objectives

- Describe only the way key process areas are affected by component oriented product
  - Each area has much more to it
- Process is compatible with CMM level 2-3
- Suitable for small teams (<7)
  - Scaleable ?
- Everything described is practiced in real life
- Metrics and the charts are normalized projects data

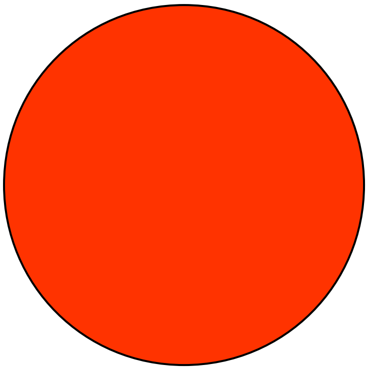
Project Planning

- Staffing
- Product life cycle
- Components integration plan
- Component life cycle



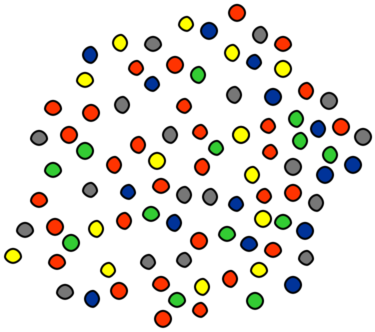
Staffing

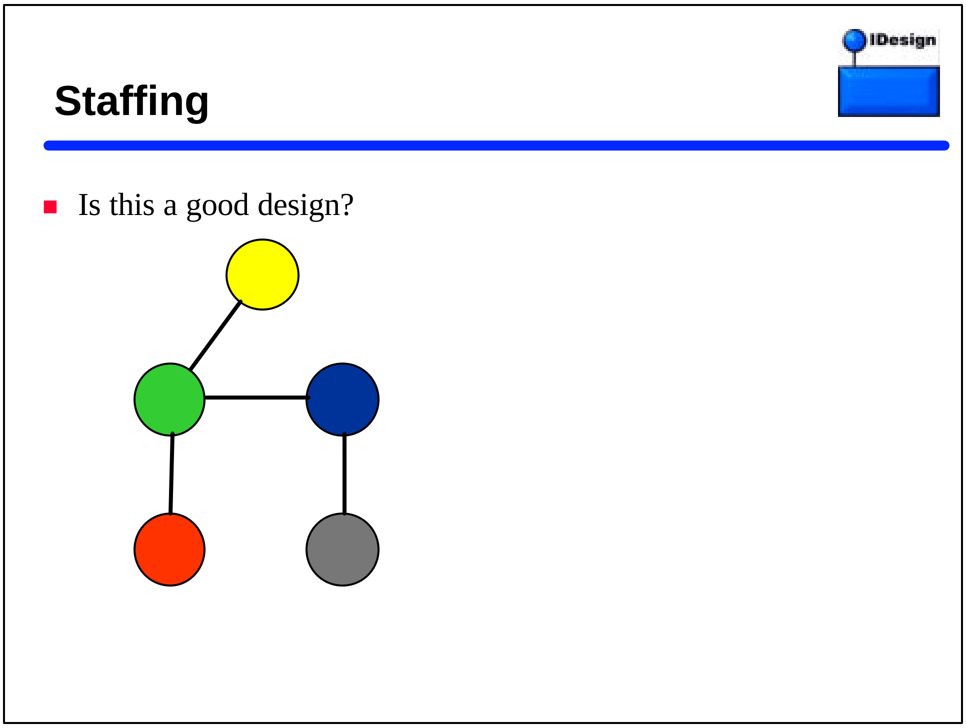
- Is this a good design?



Staffing

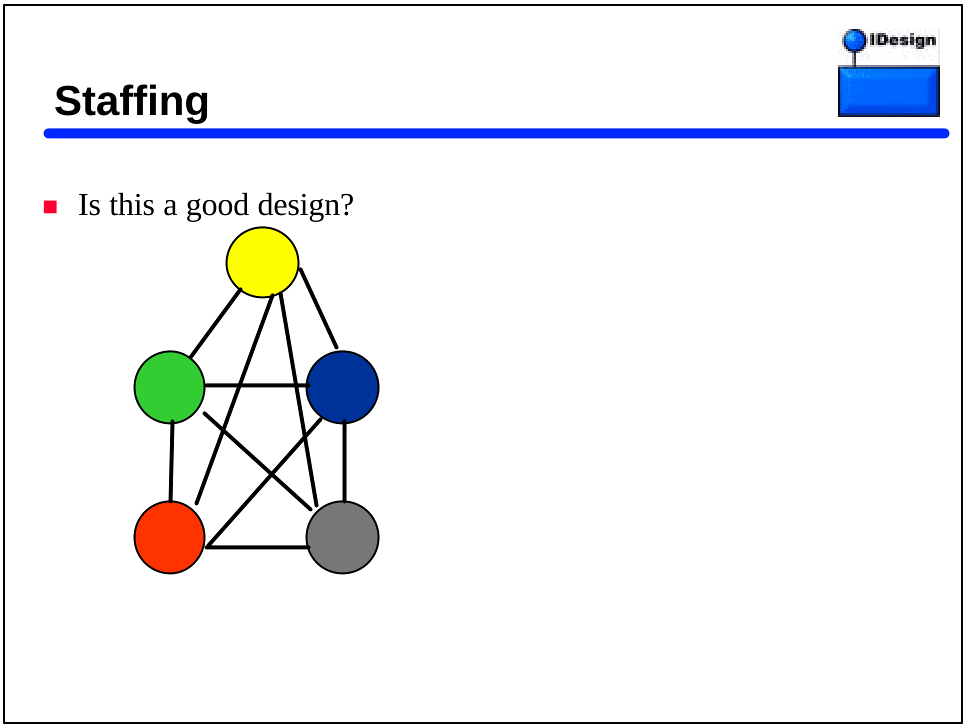
- Is this a good design?





Staffing

- Is this a good design?

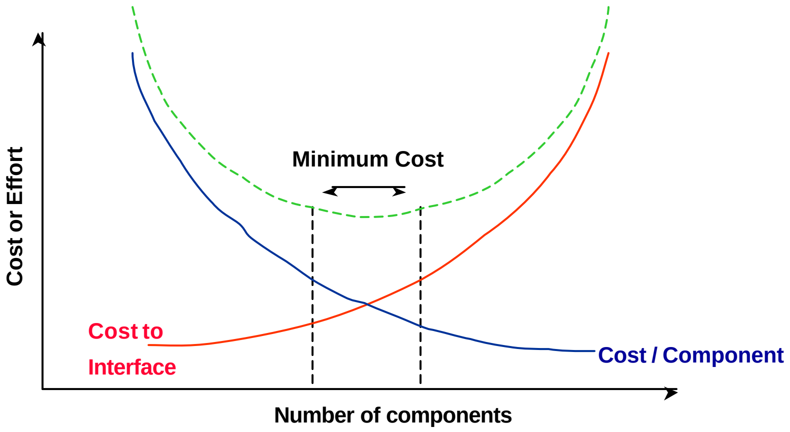


Staffing

- Is this a good design?

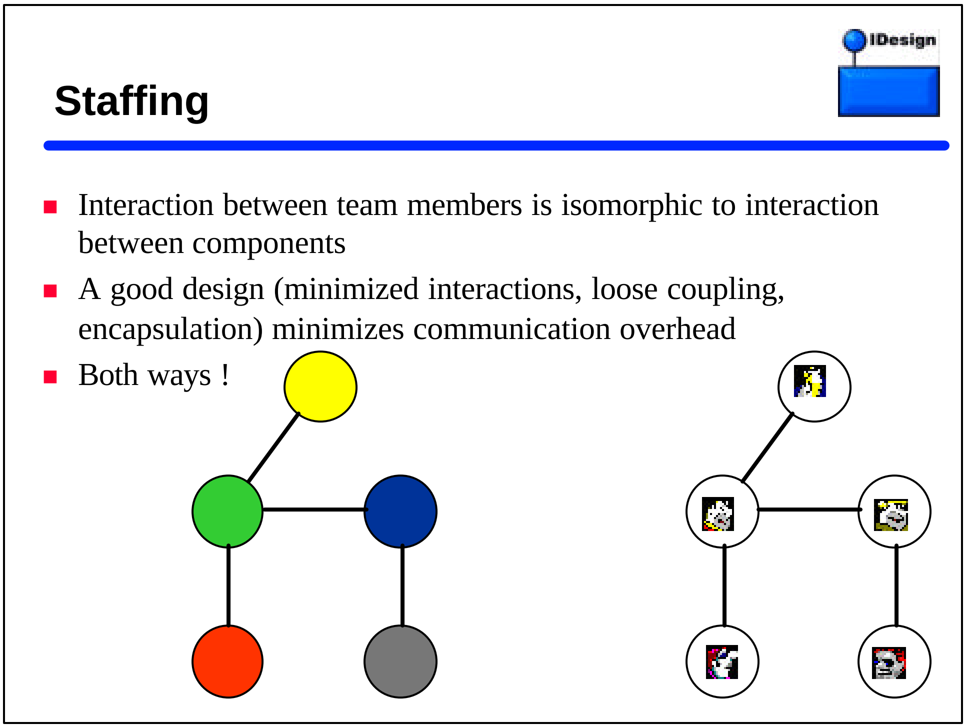
Staffing

- Balance number of components with development effort



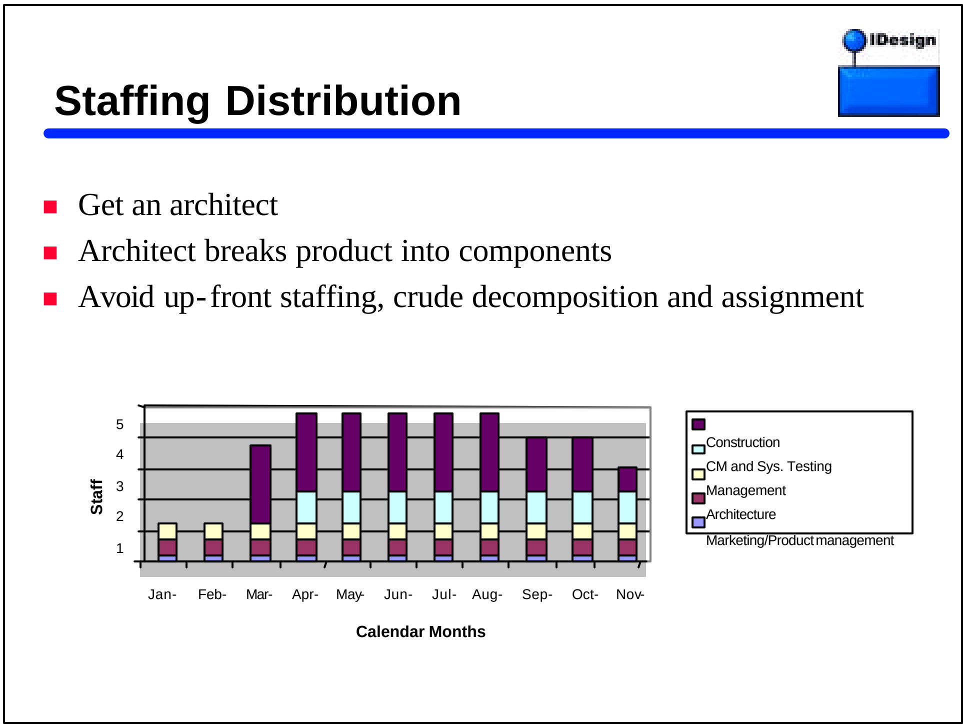
Staffing

- Not having a skilled architect is the #1 technology risk
  - Rather than the technology itself !
- Requirements analysis and architecture are contemplative time consuming activates
  - More firepower does not expedite
  - Single architect usually suffices
  - In large projects, have a senior architect and a junior/apprentice
- Assign a component to individual developer (1:1)
- Assembly boundary is team boundary



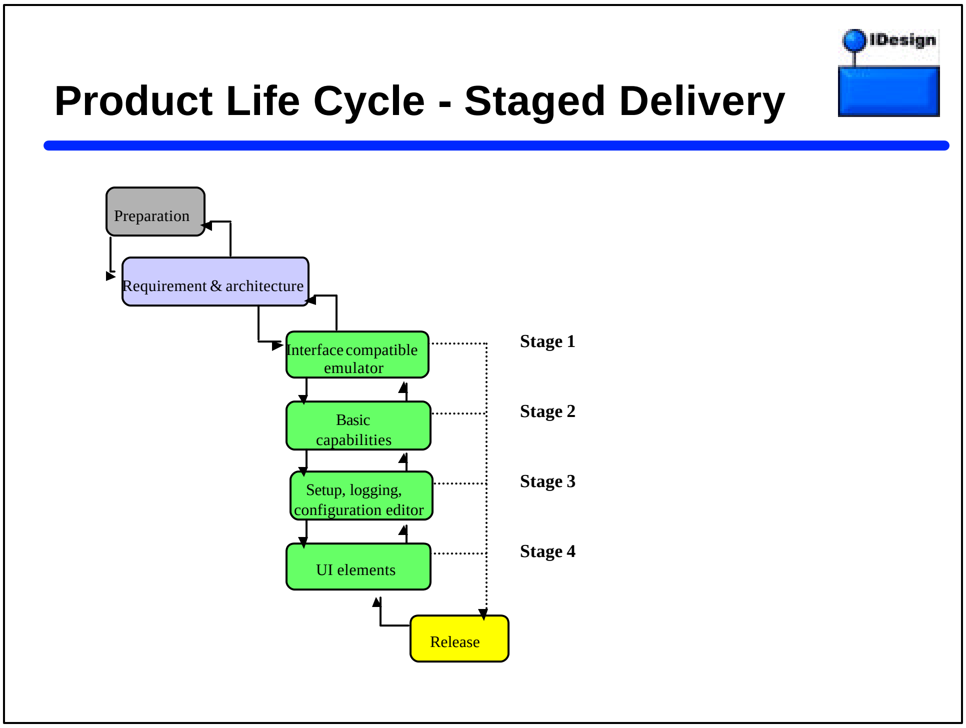
Staffing

- Interaction between team members is isomorphic to interaction between components
- A good design (minimized interactions, loose coupling, encapsulation) minimizes communication overhead
- Both ways !



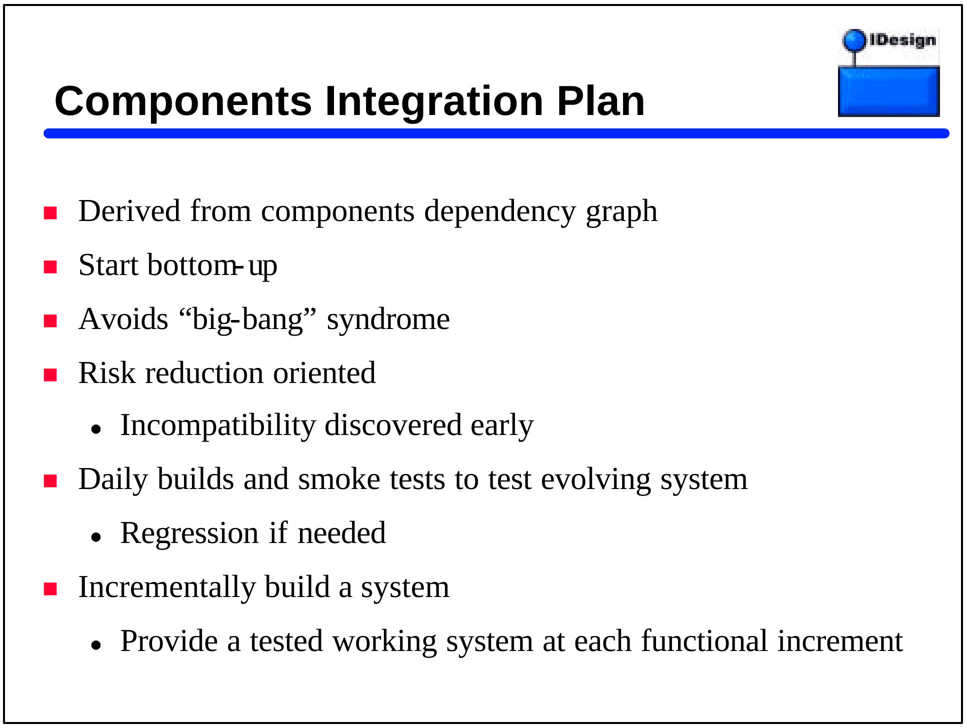
Staffing Distribution

- Get an architect
- Architect breaks product into components
- Avoid up-front staffing, crude decomposition and assignment



Product Life Cycle - Staged Delivery

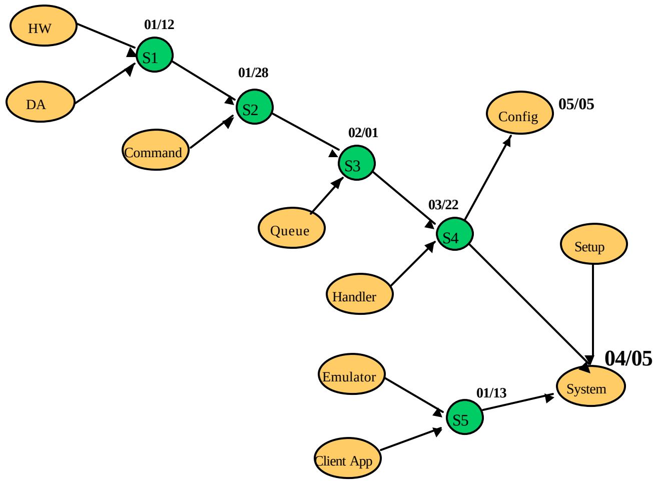
Preparation
Requirement & architecture
Interface compatible emulator
Basic capabilities
Setup, logging, configuration editor
UI elements
Release
Stage 1
Stage 2
Stage 3
Stage 4



Components Integration Plan

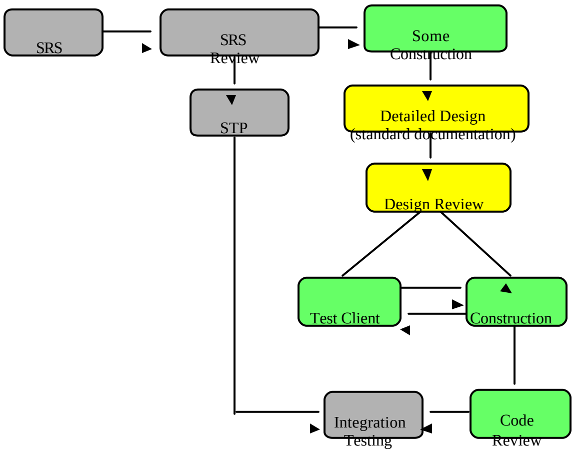
- Derived from components dependency graph
- Start bottom-up
- Avoids “big-bang” syndrome
- Risk reduction oriented
  - Incompatibility discovered early
- Daily builds and smoke tests to test evolving system
  - Regression if needed
- Incrementally build a system
  - Provide a tested working system at each functional increment

Components Integration Plan



HW
DA
Command
01/12
S1
01/28
S2
Queue
02/01
S3
Handler
03/22
S4
Config
05/05
Setup
Emulator
Client App
01/13
S5
04/05
System

Component Life Cycle

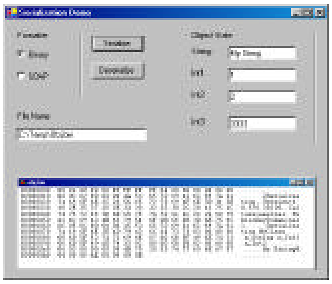


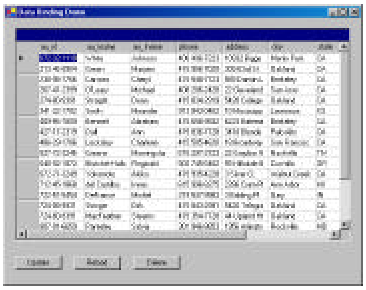
SRS
SRS Review
STP
Some Construction
Detailed Design (standard documentation)
Design Review
Test Client
Construction
Code Review
Integration Testing

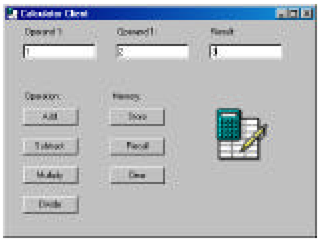
Component Testing

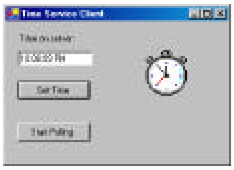
- EVERY component has its own testing environment
- Visible signs of progress to management
- Spice up “boring” testing
- Test all method calls, call backs and errors (white box)
- Fall back to isolate problems
- Assumption - no need to test the ***test*** SW
- System level test SW is provided to customer as well

Component Testing



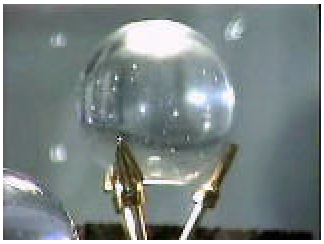






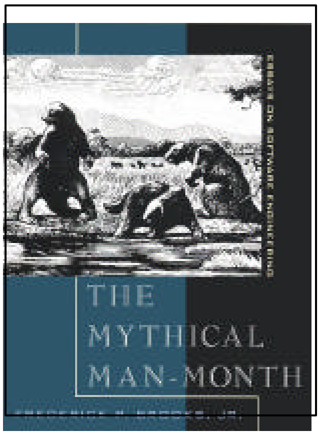
Estimation and Tracking

- Component-based effort estimation
- Component-based earned value tracking



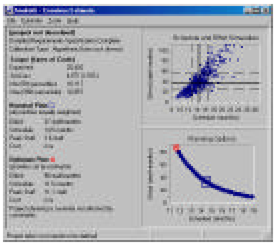
There is No Silver Bullet

- .NET projects do not take less than Windows DNA projects
  - Marginal overall improvement in time to market
  - Applications are more complex



Component-Based Effort Estimation

- Use estimation tools
- Team members participate in estimation
- Itemize lifecycle of all components
  - Do not omit:
    - Learning curves
    - Test clients
    - Installation
    - Integration points
    - Peer reviews
    - Documentation



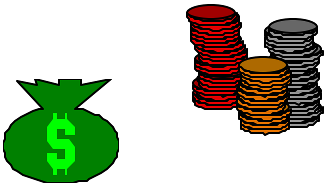
Component Based Effort Estimation

Requirements List and Resouces Allocation

	Owner	Status % Completed	% Remainder	Planning and Design	Construction (code, test)	System testing		Size by LOC
Perform market research	JL	100%	0%	30				
SW development plan	JL	75%	25%	12				
Prototype SW interfaces	JL	100%	0%	7				
SRS	JL	75%	25%	15				
Test plan	QA	0%	0%	10				
Configuration management plan	JL	50%	50%	5				
Build environment	JL	100%	0%	10				
Requirements management	JL	25%	75%	15				
detailed estimation	JL,FN	25%	75%	2				
User manual	MW	25%	75%		30			
Architecture	JL	75%	25%	35				
SW Detailed design	JL	25%	75%		20			
Serial communication	JL	75%	25%		30	10		1355
Emulator	CP	0%	100%		10	5		810
Commands queue	JL	25%	75%		20	10		1236
Modular handler	CP	0%	100%		20	10		2300
Error handling	CP,JL	0%	100%		15	5		500
Error logging	CP	0%	100%		10	3		500
Installation program	KD	0%	100%	15	30	20		2500
Configuration editor	KD	0%	100%	15	30	15		3500
Integration and testing	QA, KD, CP,JL	0%	100%		60	140		
Client application	KD	0%	100%	15	30	10		2000
Commands	CP	0%	100%	5	10	5		820
Save Type	CP	0%	100%		3			2000
Release activities	QA, KD, CP,JL	0%	100%		60	60		
Project Management	FN,JL	5%	95%	81				
Total by Category (man day)				272	378	293	943	
Total by category (man month)				13.6	18.9	24.4	56.9	
Total (man month)								
Total - Zise								17521.0
Training				40				
C++				120				
NT				20				
MFC				120				
Win32 Interface				120				
COM				240				
Advance COM				240				
Domain Knowledge				120				
Total Training				1020				
Total Training (man-month)				51				
Stage 3								
UI Elements				30	40	20		
Wafer ID				10	15	10		
Carried ID				10	15	10		
Wafer Management				30	40	20		
Advanced error logging				10	20	5		
Total				4.3	6.2	3.1		

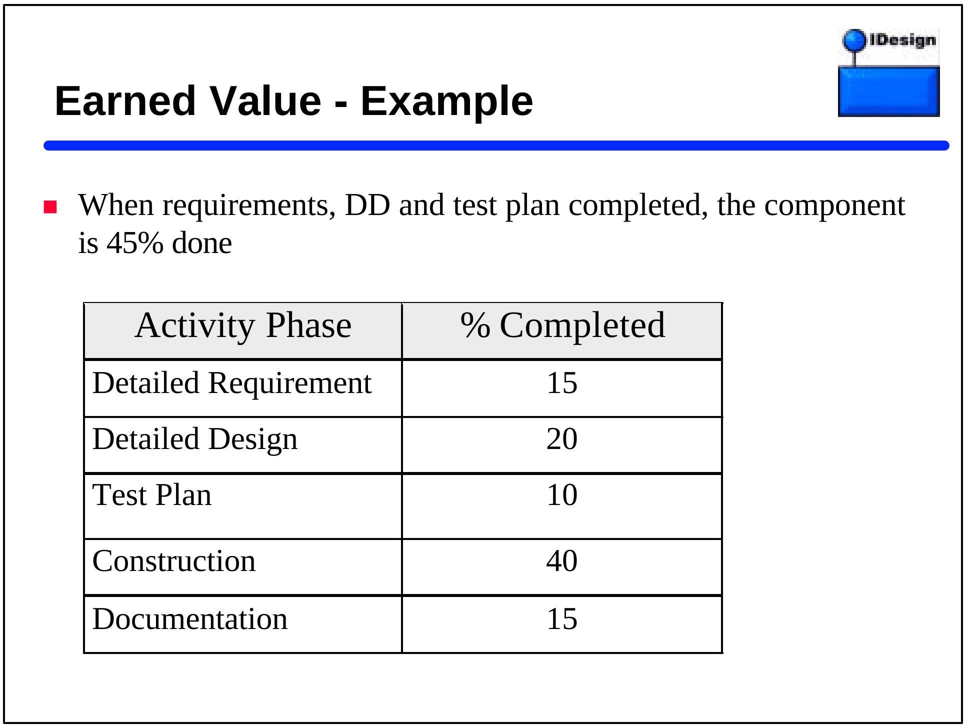
Earned Value Tracking

- Assign value of work item for the completion of component
- Compare earned value (sum of all accomplished activities across components) against effort spent
- Can predict completion date and costs



Earned Value - Example

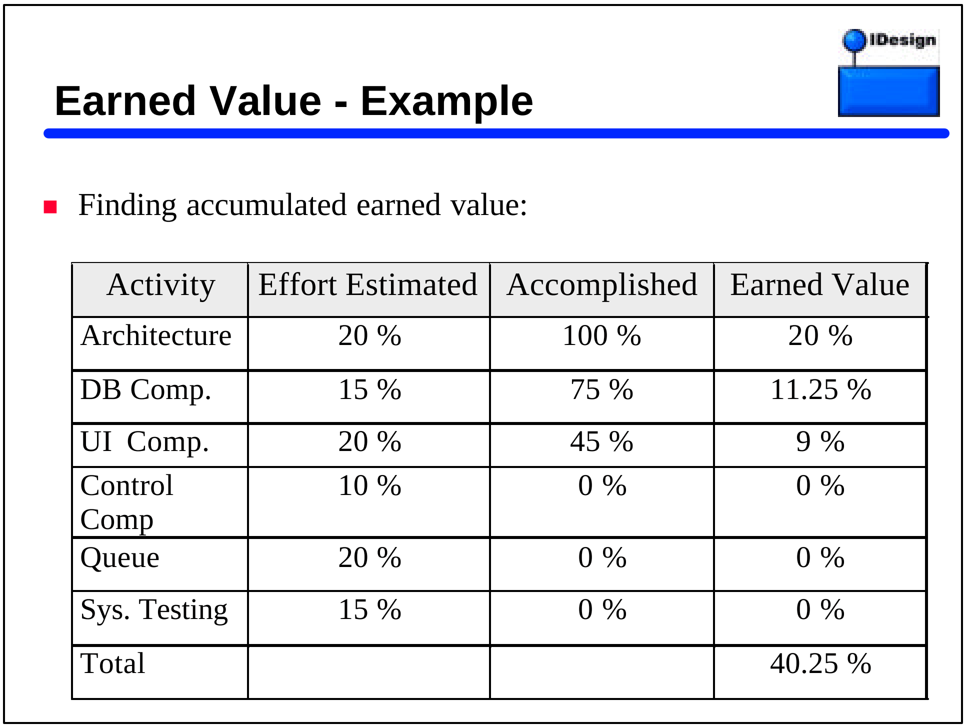
Activity	Effort Estimated	Earned Value
Architecture	40 days	20 %
DB Comp.	30 days	15 %
UI Comp.	40 days	20 %
Control comp.	20 days	10%
Queue Comp.	40 days	20 %
System Testing	30 days	15 %
Total	200 days	100 %



Earned Value - Example

- When requirements, DD and test plan completed, the component is 45% done

Activity Phase	% Completed
Detailed Requirement	15
Detailed Design	20
Test Plan	10
Construction	40
Documentation	15

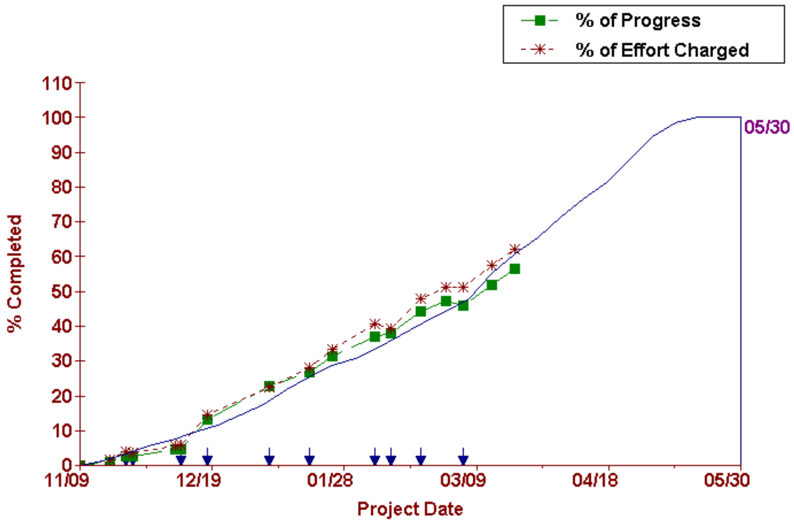


Earned Value - Example

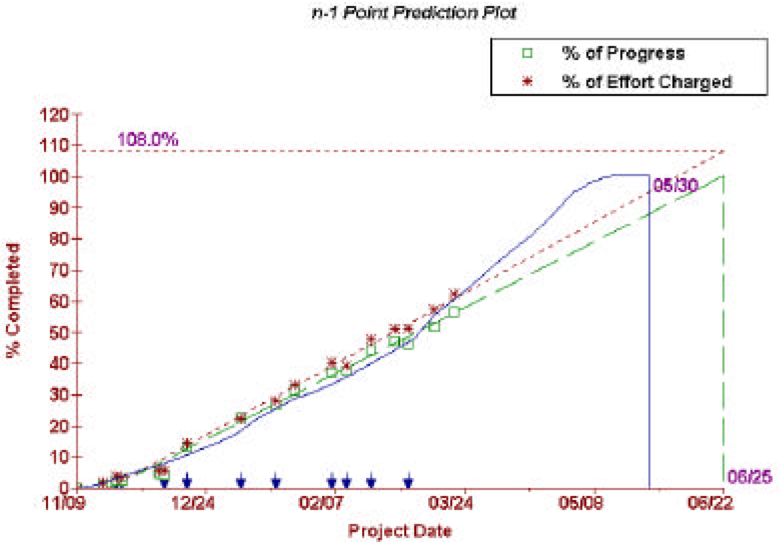
- Finding accumulated earned value:

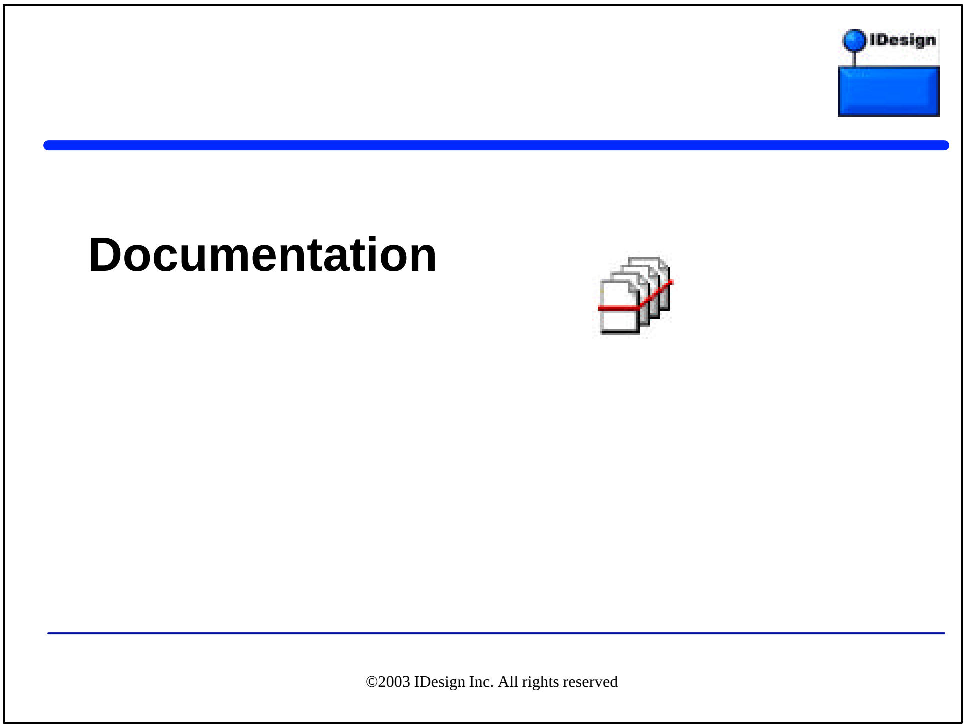
Activity	Effort Estimated	Accomplished	Earned Value
Architecture	20 %	100 %	20 %
DB Comp.	15 %	75 %	11.25 %
UI Comp.	20 %	45 %	9 %
Control Comp	10 %	0 %	0 %
Queue	20 %	0 %	0 %
Sys. Testing	15 %	0 %	0 %
Total			40.25 %

Earned Value - Example

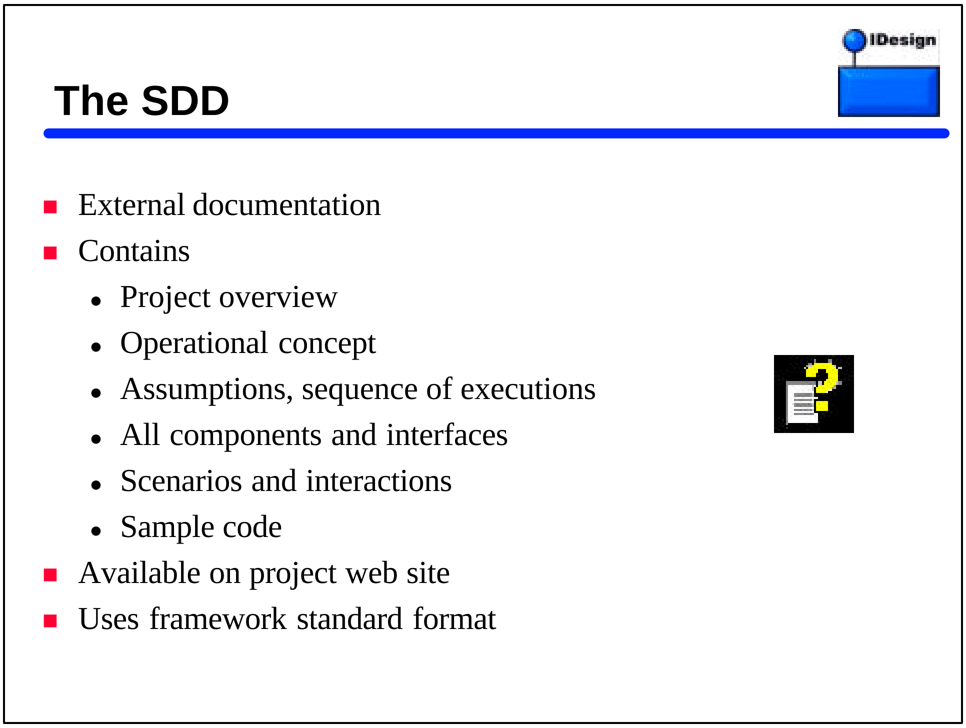


Earned Value - Example



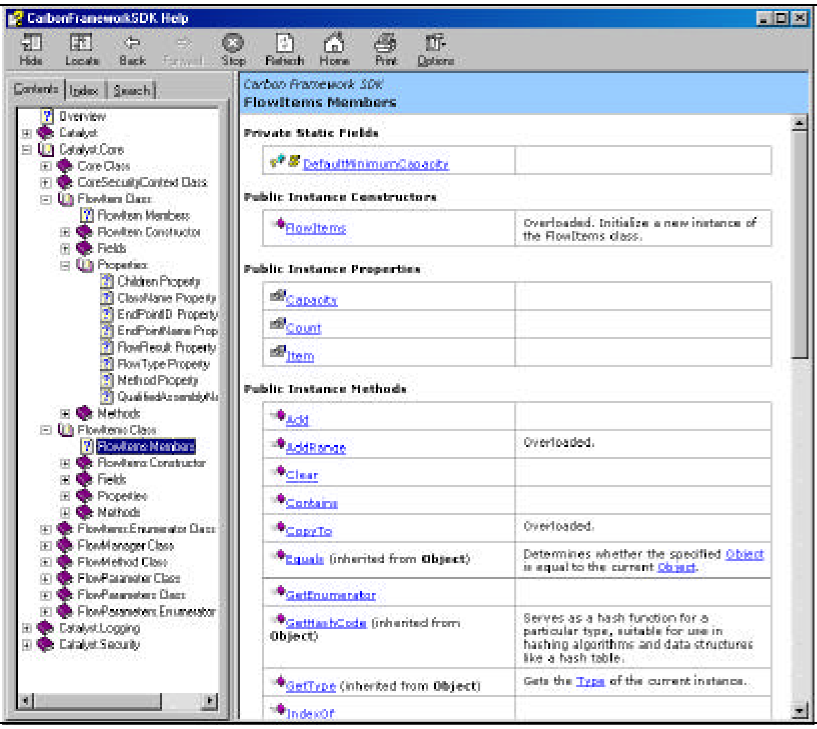


Documentation



The SDD

- External documentation
- Contains
  - Project overview
  - Operational concept
  - Assumptions, sequence of executions
  - All components and interfaces
  - Scenarios and interactions
  - Sample code
- Available on project web site
- Uses framework standard format



Carbon Framework SDK

FlowItems Members

Private Static Fields

DefaultMinimumCapacity	

Public Instance Constructors

FlowItems	Overloaded. Initialize a new instance of the FlowItems class.

Public Instance Properties

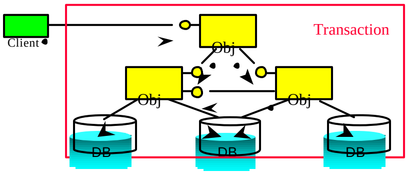
Capacity	
Count	
Item	

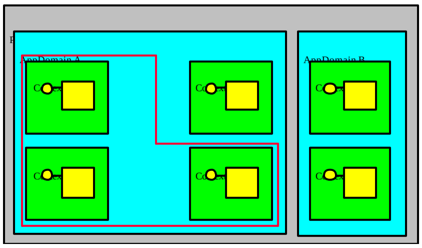
Public Instance Methods

Add	
AddRange	Overloaded.
Clear	
Contains	
CopyTo	Overloaded.
Equals (inherited from **Object**)	Determines whether the specified Object is equal to the current Object.
GetEnumerator	
GetHashCode (inherited from **Object**)	Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from **Object**)	Gets the Type of the current instance.
IndexOf	

The SDD

- SDD should contain context maps
  - Concurrency
  - Security
  - Transaction
- Context maps are some of the most important items you will design and document

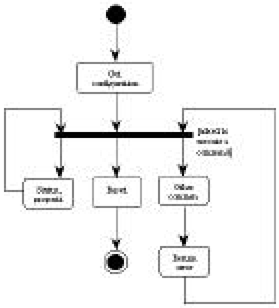




Requirement Management and Traceability

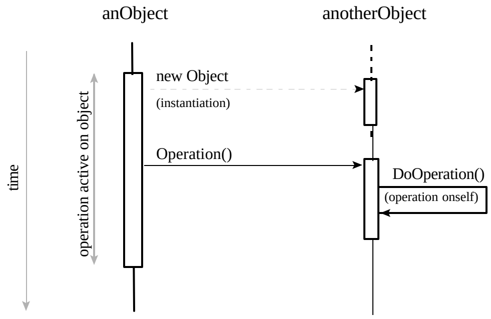
Requirement Management and Traceability

- Base Software Requirement Spec (SRS) on use cases
- Describe use cases graphically in UML activities diagrams
  - The required dynamic behavior of the system



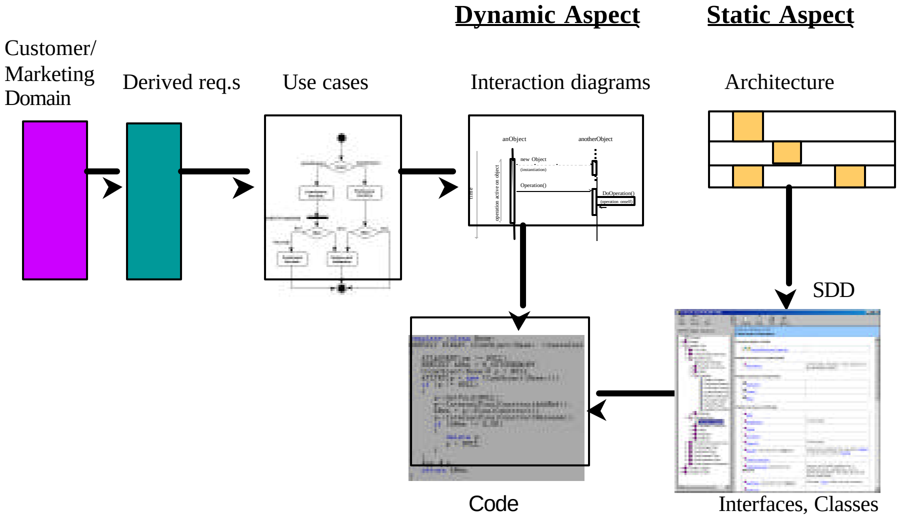
Requirement Management and Traceability

- After breaking the product to components, factor interfaces using UML interaction diagrams:



- Interaction diagram per use case/activity diagram

Requirement Management and Traceability



Dynamic Aspect

Static Aspect

Customer/ Marketing Domain

Derived req.s

Use cases

Interaction diagrams

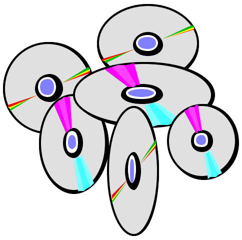
Architecture

SDD

Code

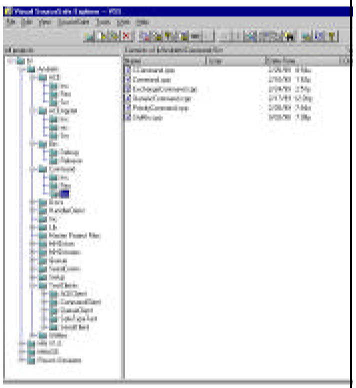
Interfaces, Classes

Configuration Management and Source Control



Source Control

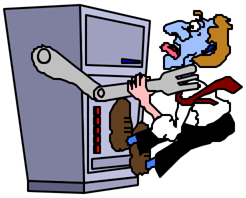
- Each assembly kept separate
  - Standard folders structure
  - History, branches
  - Component documents
  - Client and test software
- Daily builds (automated) and daily smoke test
- Integrated with VS.NET
- Connected to project web site
  - Some component are available separately



Build Environment

- Customize it !
- One container solution, grouping many sub projects
- Automate activities:
  - COM export and registration
  - Installing in the GAC
  - Setup projects
- One click to build, set up, deploy, test

SQC



SQC

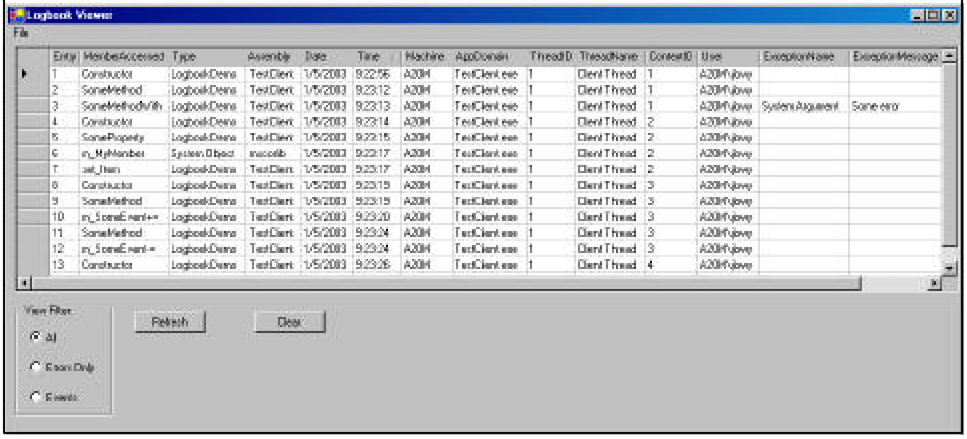
- Test plan per component and for the system
  - Based on the component SRS and use cases
- EVERY component has its own testing environment
  - Invoke all methods, call backs and error handlers (white box)
  - Fall back to isolate problems
  - Testing performed by the developers
- SQC performs
  - System level testing only
  - Daily builds and smoke test (automated)

0 Tolerance to Defects

- At any given time, a component has zero bugs <Period>
  - Components must be rock solid
  - Added inherit complexity in component-based application
- Defensive programming
  - Assert every assumption
- Component is stand alone
- Log files
  - Interleaved entries
- Average lifespan of a bug is a few minutes

Logbook

- Logbook/flight recorder
    - Verbosity levels



Entry	MemberAccessed	Type	Assembly	Date	Time	Machine	AppDomain	ThreadID	ThreadName	ContextID	User	ExceptionName	ExceptionMessage
1	Constructor	LogbookDemo	TestClient	1/5/2003	9:22:56	A20M	TestClient.exe	1	Client Thread	1	A20M\juval		
2	SomeMethod	LogbookDemo	TestClient	1/5/2003	9:23:12	A20M	TestClient.exe	1	Client Thread	1	A20M\juval		
3	SomeMethodWith	LogbookDemo	TestClient	1/5/2003	9:23:13	A20M	TestClient.exe	1	Client Thread	1	A20M\juval	System.Argument	Some erro
4	Constructor	LogbookDemo	TestClient	1/5/2003	9:23:14	A20M	TestClient.exe	1	Client Thread	2	A20M\juval		
5	SomeProperty	LogbookDemo	TestClient	1/5/2003	9:23:15	A20M	TestClient.exe	1	Client Thread	2	A20M\juval		
6	m_MyMember	System.Object	mscorlib	1/5/2003	9:23:17	A20M	TestClient.exe	1	Client Thread	2	A20M\juval		
7	set_Item	LogbookDemo	TestClient	1/5/2003	9:23:17	A20M	TestClient.exe	1	Client Thread	2	A20M\juval		
8	Constructor	LogbookDemo	TestClient	1/5/2003	9:23:19	A20M	TestClient.exe	1	Client Thread	3	A20M\juval		
9	SomeMethod	LogbookDemo	TestClient	1/5/2003	9:23:19	A20M	TestClient.exe	1	Client Thread	3	A20M\juval		
10	m_SomeEvent+=	LogbookDemo	TestClient	1/5/2003	9:23:20	A20M	TestClient.exe	1	Client Thread	3	A20M\juval		
11	SomeMethod	LogbookDemo	TestClient	1/5/2003	9:23:24	A20M	TestClient.exe	1	Client Thread	3	A20M\juval		
12	m_SomeEvent-=	LogbookDemo	TestClient	1/5/2003	9:23:24	A20M	TestClient.exe	1	Client Thread	3	A20M\juval		
13	Constructor	LogbookDemo	TestClient	1/5/2003	9:23:26	A20M	TestClient.exe	1	Client Thread	4	A20M\juval		

Other Issues

- Simulation and Emulation
- Training
- Peer Reviews
- Metrics
- Visibility to Management

Simulation and Emulation

- Every component has both simulator and emulator
  - Emulation returns “success” on every call, easy to develop
  - Simulator is as real as the component
    - Manages state
- Both are useful
  - Development - give your client the interfaces early
  - Debugging
  - Demo
  - Not changing the DB, access the HW, etc.
  - Trigger rare conditions, errors
  - Automate smoke tests
- Should be able to switch modes programmatically

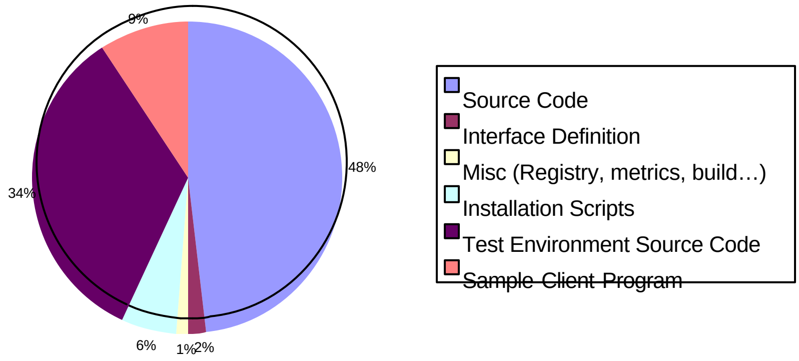
Training

- Internal training sessions
- External courses
- Staff mentoring
- Each SW engineers should have:
  - A pile of books to read (7”)
  - Articles
- Emphasis on deductive knowledge sharing

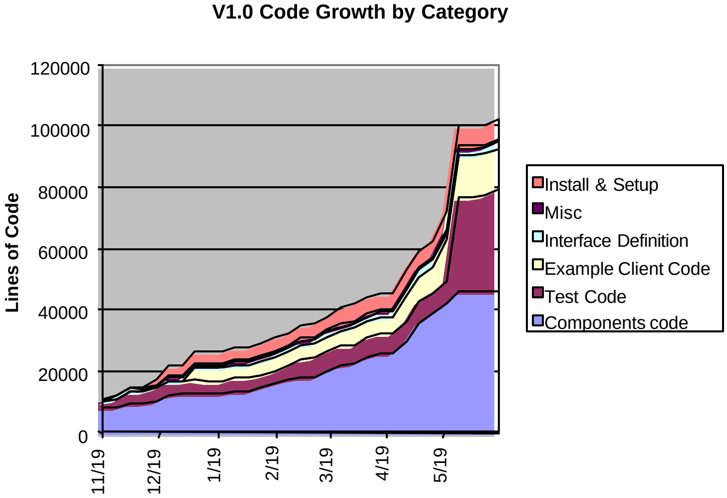
Peer Reviews

- What to review
  - Component requirement reviews
  - Component test plan reviews
  - Component design reviews
  - Code reviews (ALL the core code is reviewed)
- Techniques
  - Formal review
  - Walk through
  - Buddy programming
- High degree of mutual involvement
  - Strict coding standards are a necessity
  - Team spirit and vision for producing highest quality work

Metrics - Code Categories

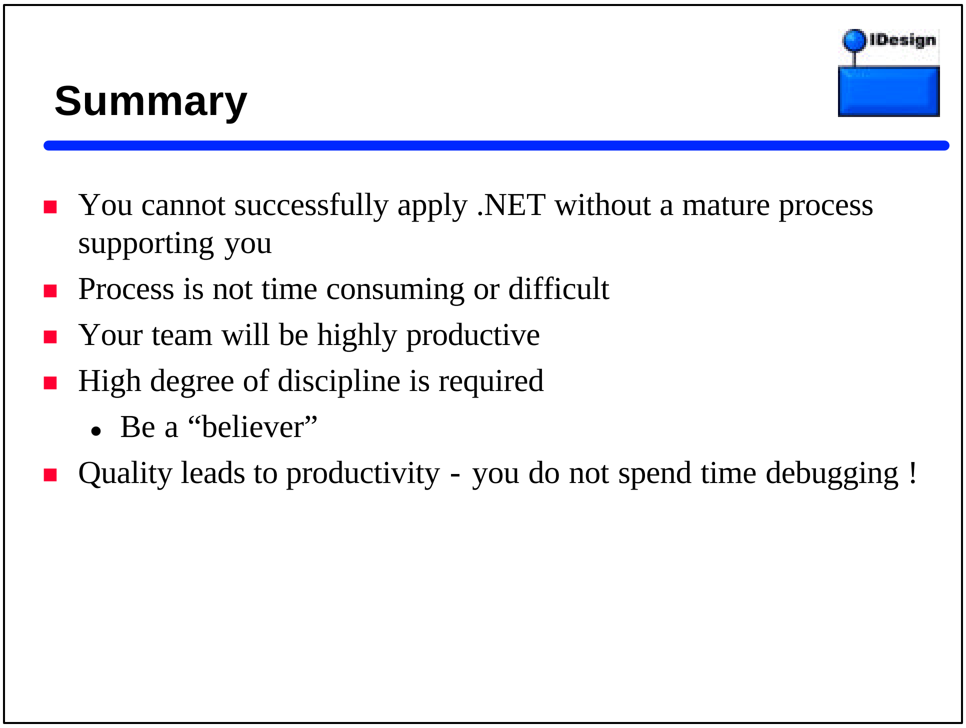


Metrics - Code Growth



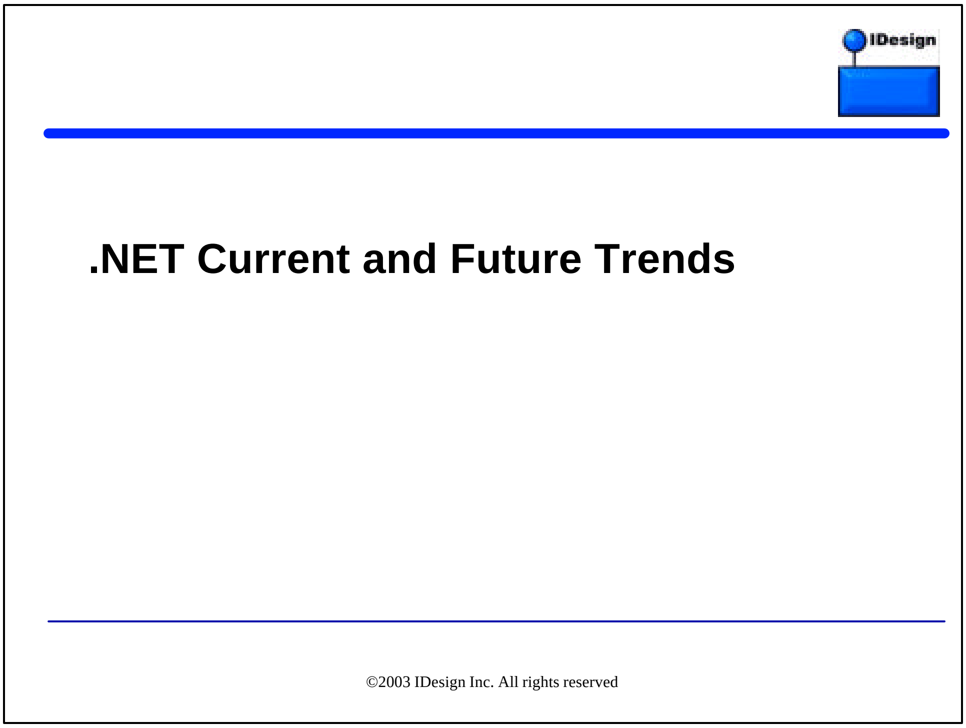
Management Visibility

- Risk management
- Frequent “push” status reports
  - Components integration points are your mile stones
  - Earned value charts
- Frequent demos (component testers)



Summary

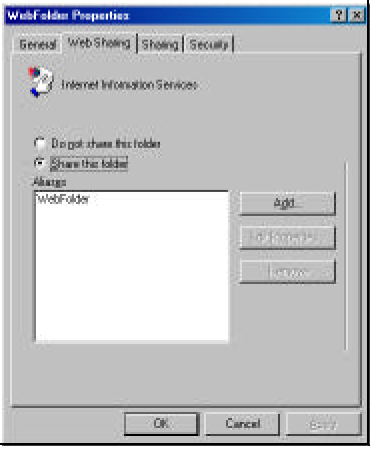
- You cannot successfully apply .NET without a mature process supporting you
- Process is not time consuming or difficult
- Your team will be highly productive
- High degree of discipline is required
  - Be a “believer”
- Quality leads to productivity - you do not spend time debugging !



.NET Current and Future Trends

Streaming Windows Forms

- Apply web share on a folder
- Send the link!

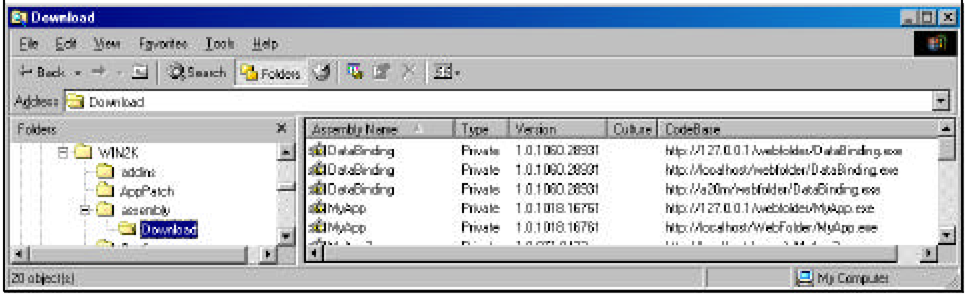


○ Do not share this folder
● Share this folder

Aliases
WebFolder

Streaming Windows Forms

- At client machine, application placed in Download Assembly Cache
  - .NET remembers origin
  - Assembly executes with appropriate security policy



Address: Download

Assembly Name	Type	Version	Culture	CodeBase
DataBinding	Private	1.0.1060.28931		http://127.0.0.1/webfolder/DataBinding.exe
DataBinding	Private	1.0.1060.28931		http://localhost/webfolder/DataBinding.exe
DataBinding	Private	1.0.1060.28931		http://a20m/webfolder/DataBinding.exe
MyApp	Private	1.0.1018.16761		http://127.0.0.1/webfolder/MyApp.exe
MyApp	Private	1.0.1018.16761		http://localhost/WebFolder/MyApp.exe

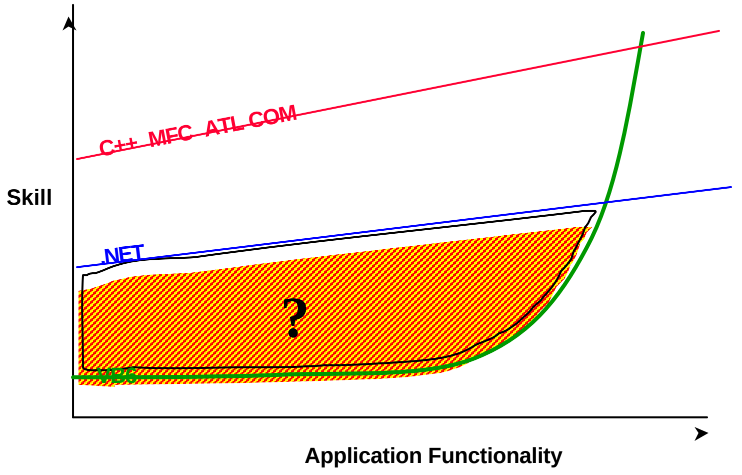
20 object(s)

Streaming Windows Forms

- Death of the browser as front-end
- Can still be a web application
  - Use web services to connect to remote server

Feature	Windows Forms	ASP.NET
Rich UI	+	-
Ease of development	+	-
Easy deployment	+	+
Web access to server	+	+
Multi platform support	-	-

C# and VB.NET Future Trends



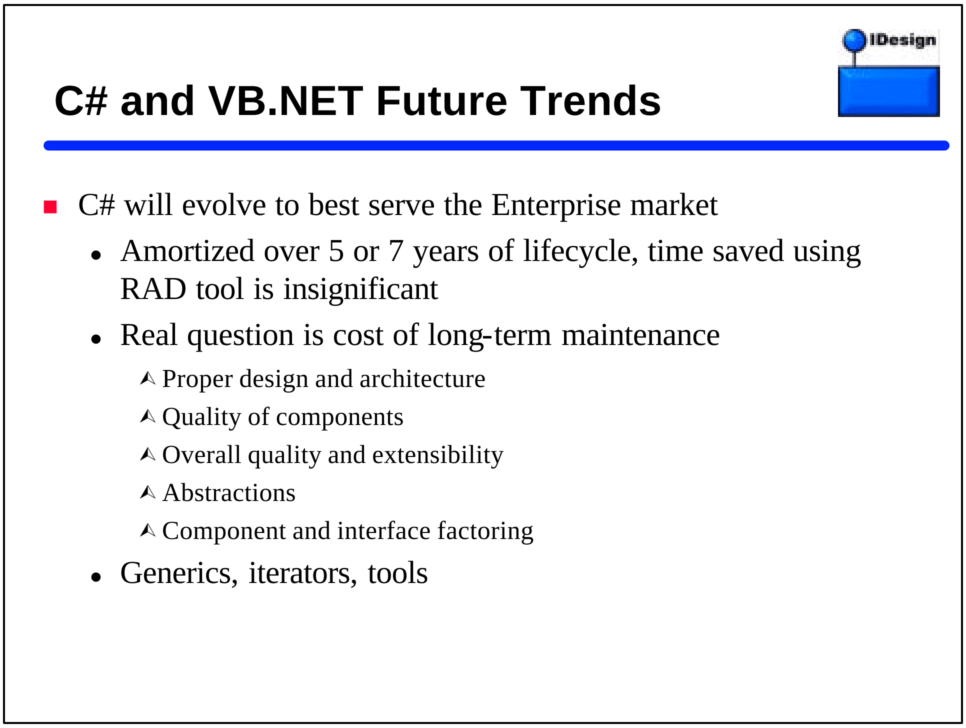
C# and VB.NET Future Trends

- First to market often at expense of long-term maintenance
  - Requires different skills of developers and managers
- VB.NET will evolve to cover gap in skill/functionality curve
  - Advanced wizards
  - Classes
  - Changes to the framework
  - Task automation tools
  - Fastest edit-test-continue cycle
  - Productivity
    - No need for unsafe code, generics, etc

Opinion

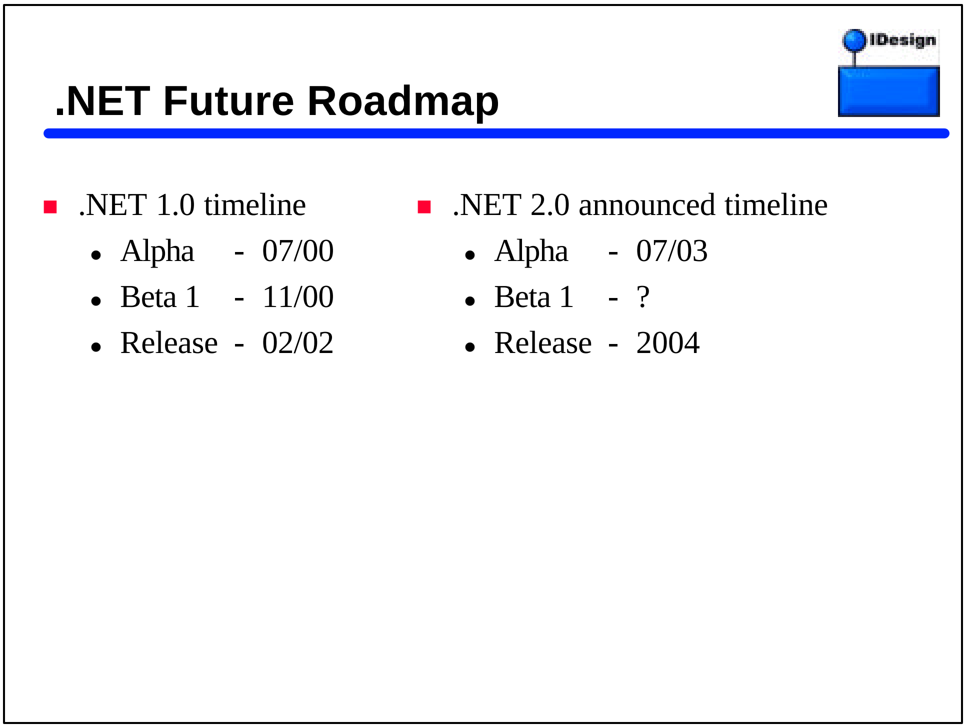
C# and VB.NET Future Trends

- First to market often at expense of long-term maintenance
  - Requires different skills of developers and managers
- VB.NET will evolve to cover gap in skill/functionality curve
  - Advanced wizards
  - Classes
  - Changes to the framework
  - Task automation tools
  - Fastest edit-test-continue cycle
  - Productivity
    - No need for unsafe code, generics, etc



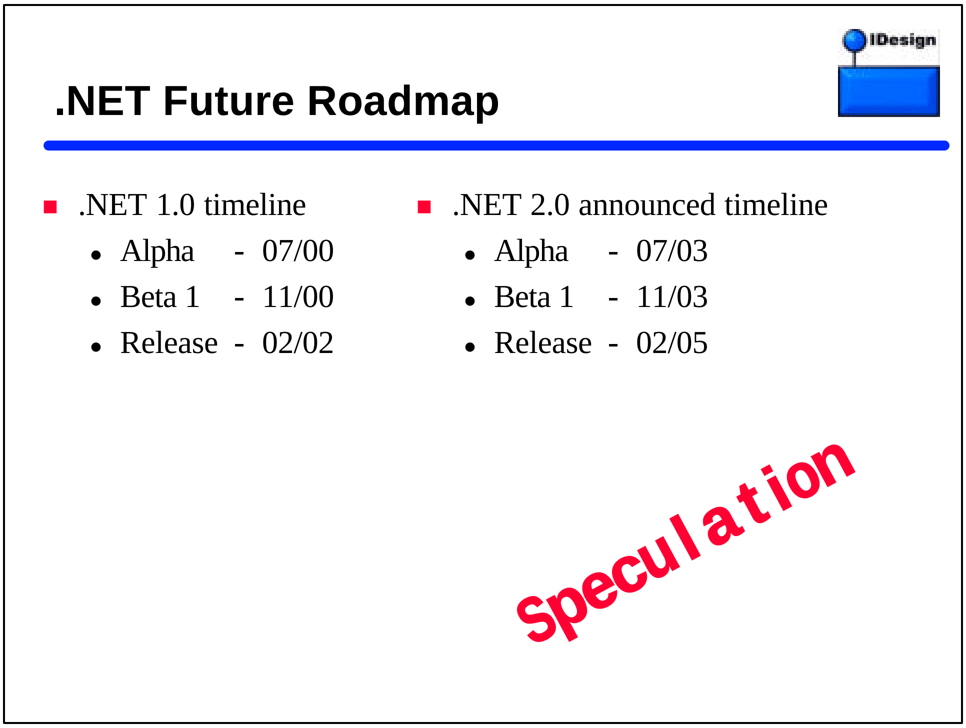
C# and VB.NET Future Trends

- C# will evolve to best serve the Enterprise market
  - Amortized over 5 or 7 years of lifecycle, time saved using RAD tool is insignificant
  - Real question is cost of long-term maintenance
    - Proper design and architecture
    - Quality of components
    - Overall quality and extensibility
    - Abstractions
    - Component and interface factoring
  - Generics, iterators, tools



.NET Future Roadmap

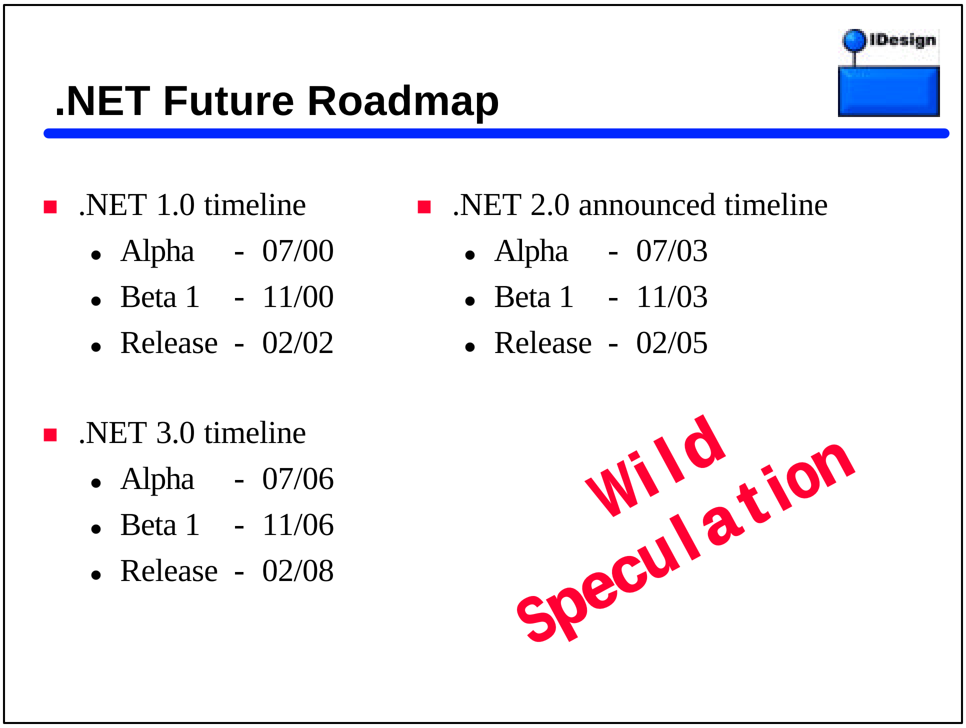
- .NET 1.0 timeline
  - Alpha - 07/00
  - Beta 1 - 11/00
  - Release - 02/02
- .NET 2.0 announced timeline
  - Alpha - 07/03
  - Beta 1 - ?
  - Release - 2004



.NET Future Roadmap

- .NET 1.0 timeline
  - Alpha - 07/00
  - Beta 1 - 11/00
  - Release - 02/02
- .NET 2.0 announced timeline
  - Alpha - 07/03
  - Beta 1 - 11/03
  - Release - 02/05

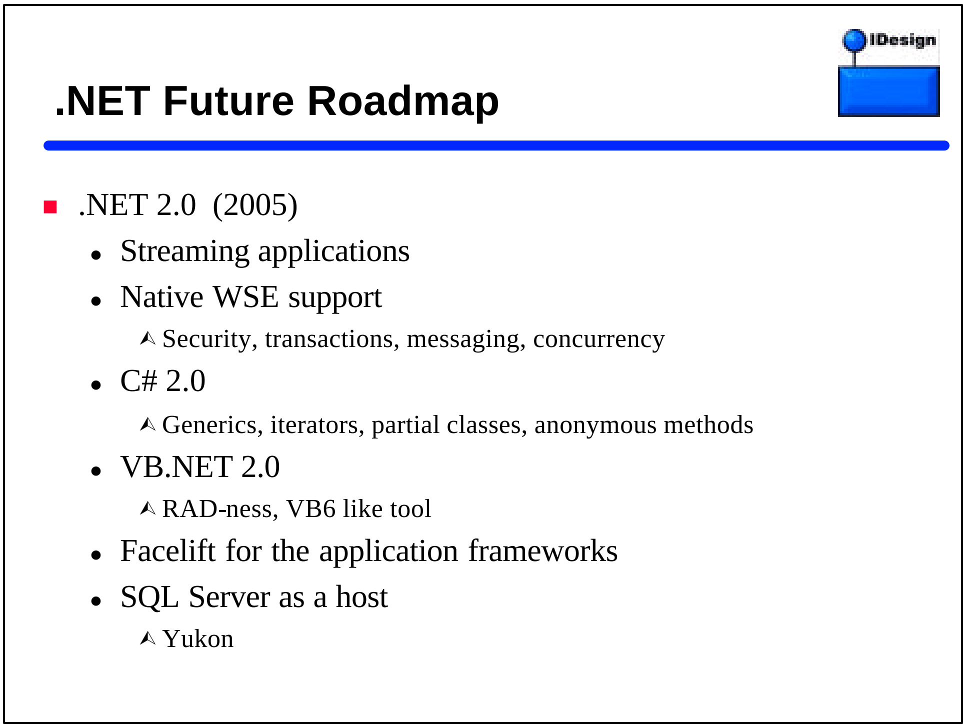
Speculation



.NET Future Roadmap

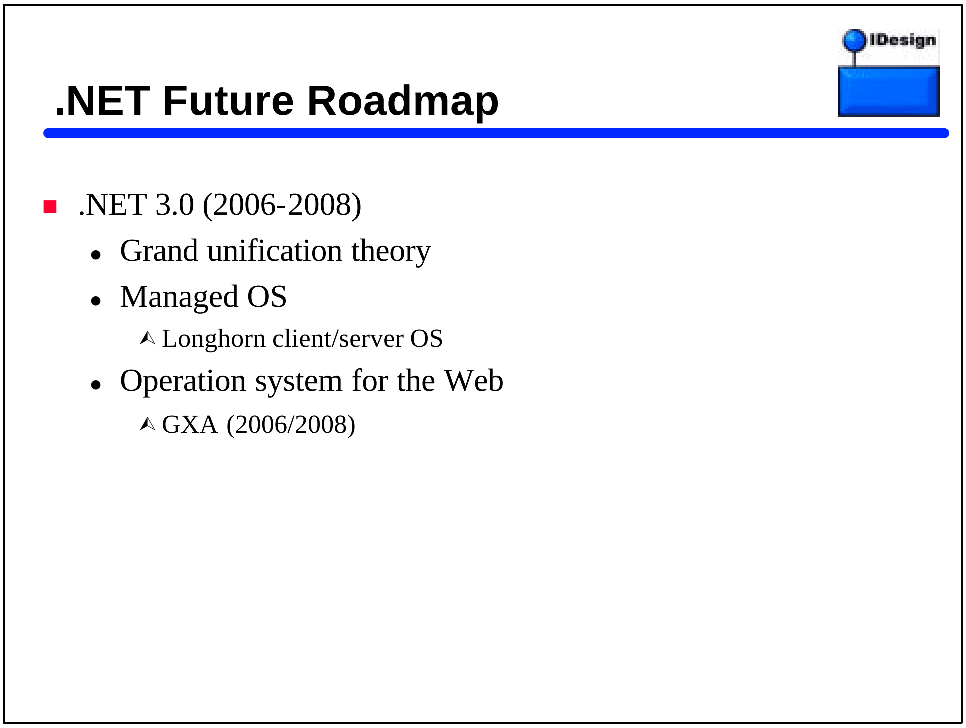
- .NET 1.0 timeline
  - Alpha - 07/00
  - Beta 1 - 11/00
  - Release - 02/02
- .NET 2.0 announced timeline
  - Alpha - 07/03
  - Beta 1 - 11/03
  - Release - 02/05
- .NET 3.0 timeline
  - Alpha - 07/06
  - Beta 1 - 11/06
  - Release - 02/08

Wild Speculation



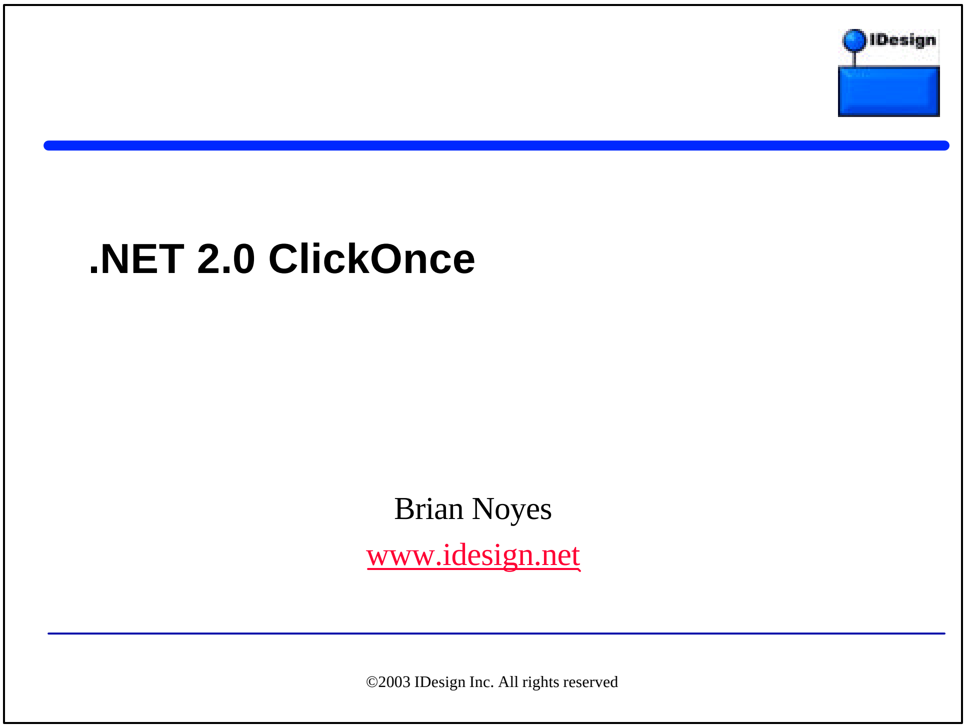
.NET Future Roadmap

- .NET 2.0 (2005)
  - Streaming applications
  - Native WSE support
    - Security, transactions, messaging, concurrency
  - C# 2.0
    - Generics, iterators, partial classes, anonymous methods
  - VB.NET 2.0
    - RAD-ness, VB6 like tool
  - Facelift for the application frameworks
  - SQL Server as a host
    - Yukon



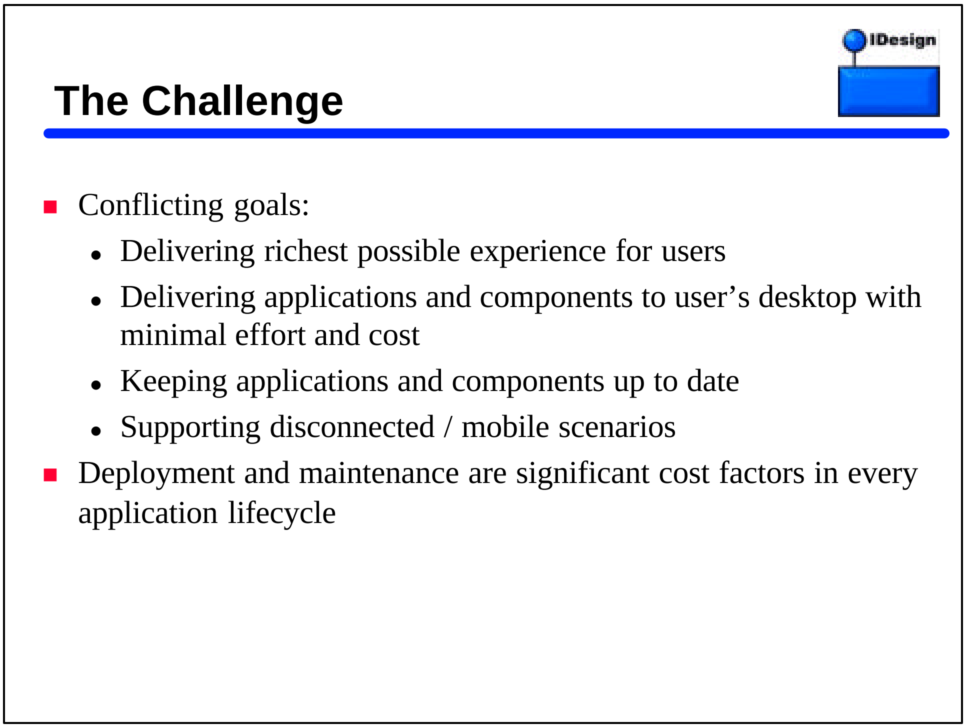
.NET Future Roadmap

- .NET 3.0 (2006-2008)
  - Grand unification theory
  - Managed OS
    - Longhorn client/server OS
  - Operation system for the Web
    - GXA (2006/2008)



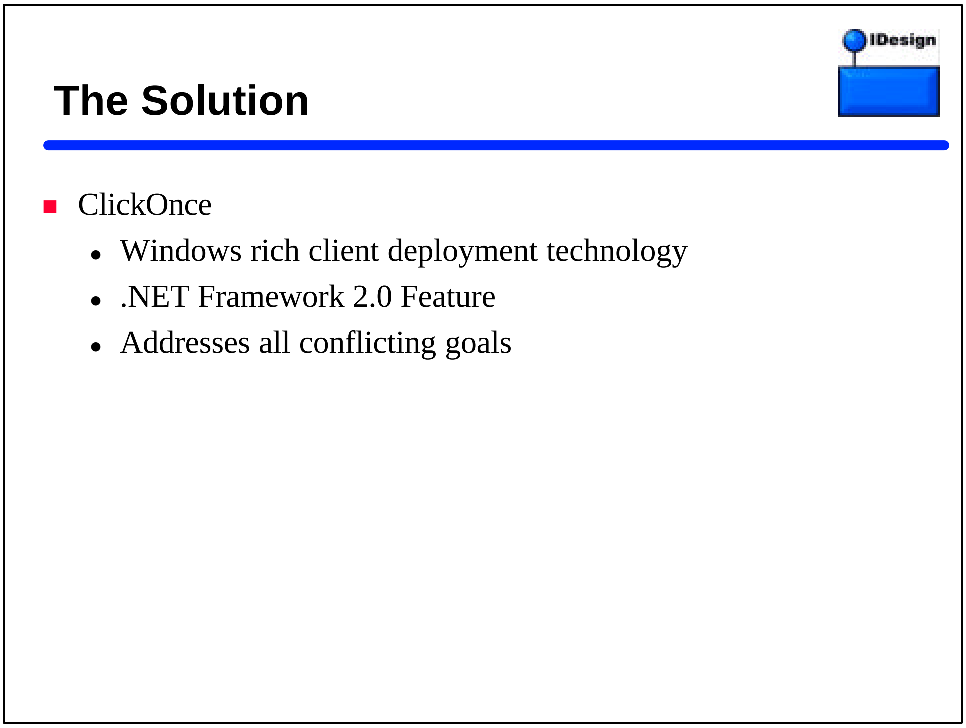
.NET 2.0 ClickOnce

Brian Noyes
www.idesign.net



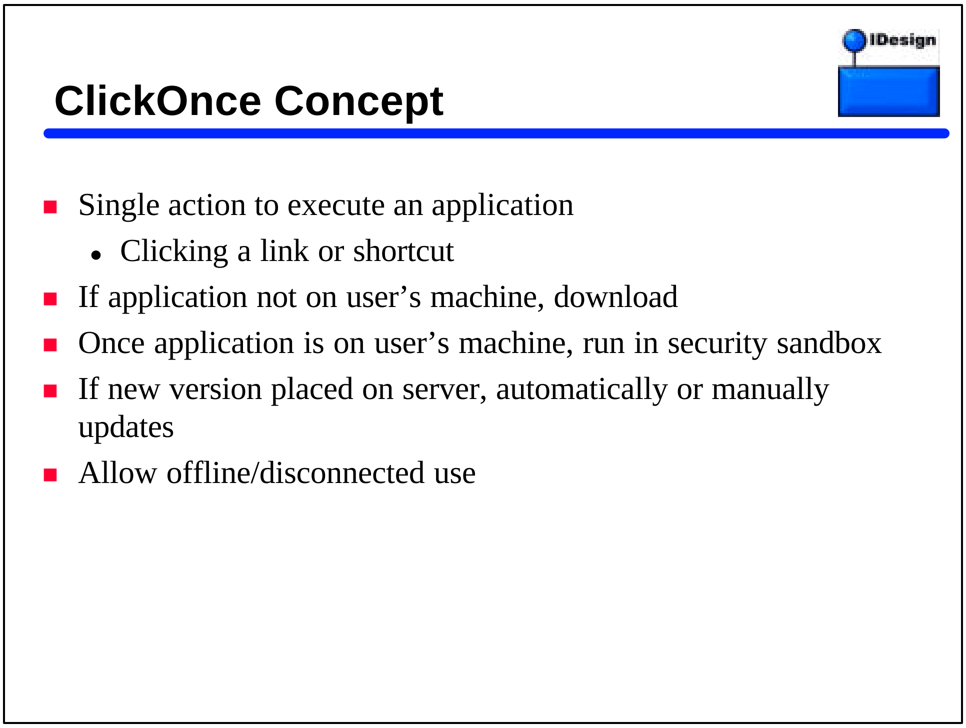
The Challenge

- Conflicting goals:
  - Delivering richest possible experience for users
  - Delivering applications and components to user’s desktop with minimal effort and cost
  - Keeping applications and components up to date
  - Supporting disconnected / mobile scenarios
- Deployment and maintenance are significant cost factors in every application lifecycle



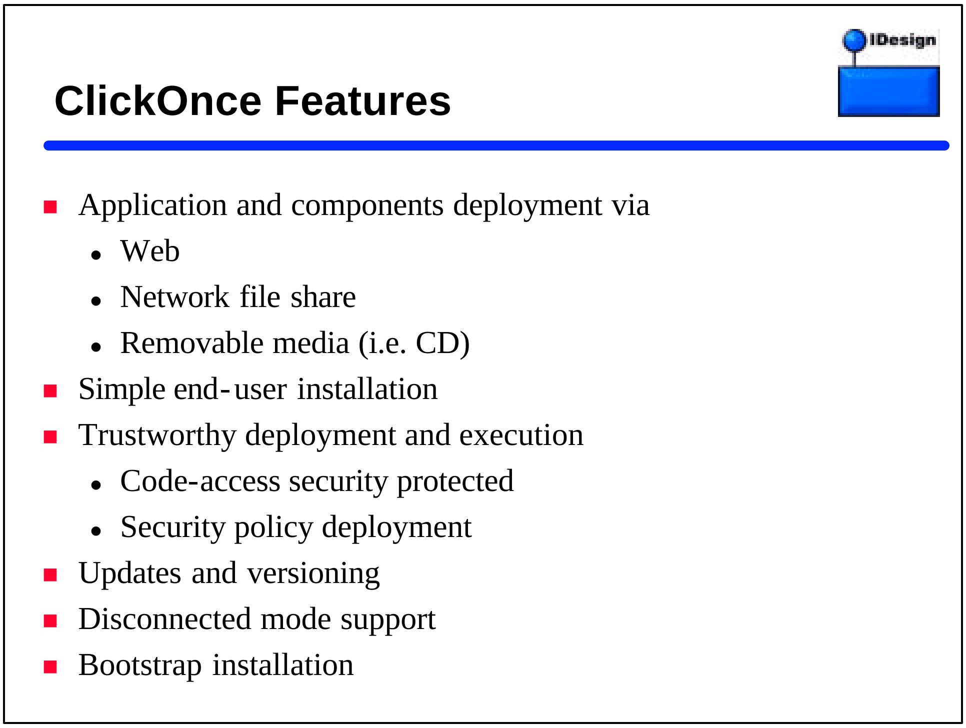
The Solution

- ClickOnce
  - Windows rich client deployment technology
  - .NET Framework 2.0 Feature
  - Addresses all conflicting goals



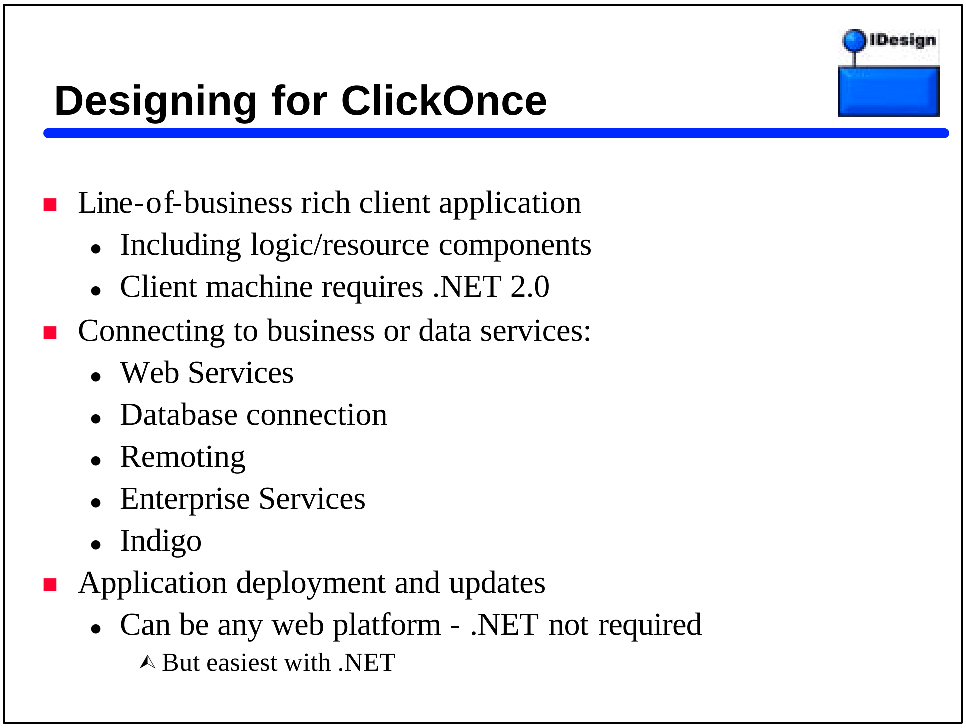
ClickOnce Concept

- Single action to execute an application
  - Clicking a link or shortcut
- If application not on user’s machine, download
- Once application is on user’s machine, run in security sandbox
- If new version placed on server, automatically or manually updates
- Allow offline/disconnected use



ClickOnce Features

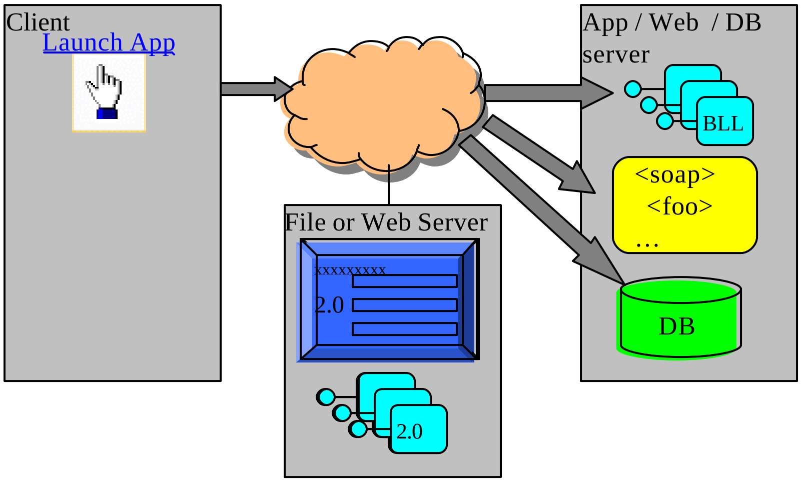
- Application and components deployment via
  - Web
  - Network file share
  - Removable media (i.e. CD)
- Simple end-user installation
- Trustworthy deployment and execution
  - Code-access security protected
  - Security policy deployment
- Updates and versioning
- Disconnected mode support
- Bootstrap installation



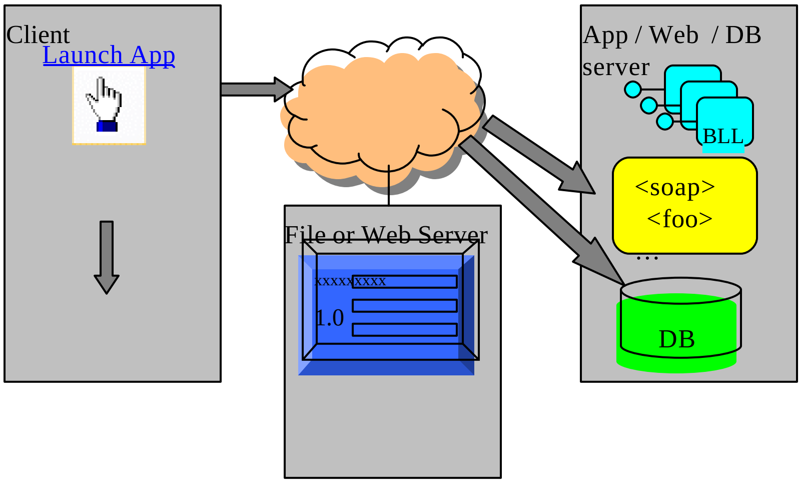
Designing for ClickOnce

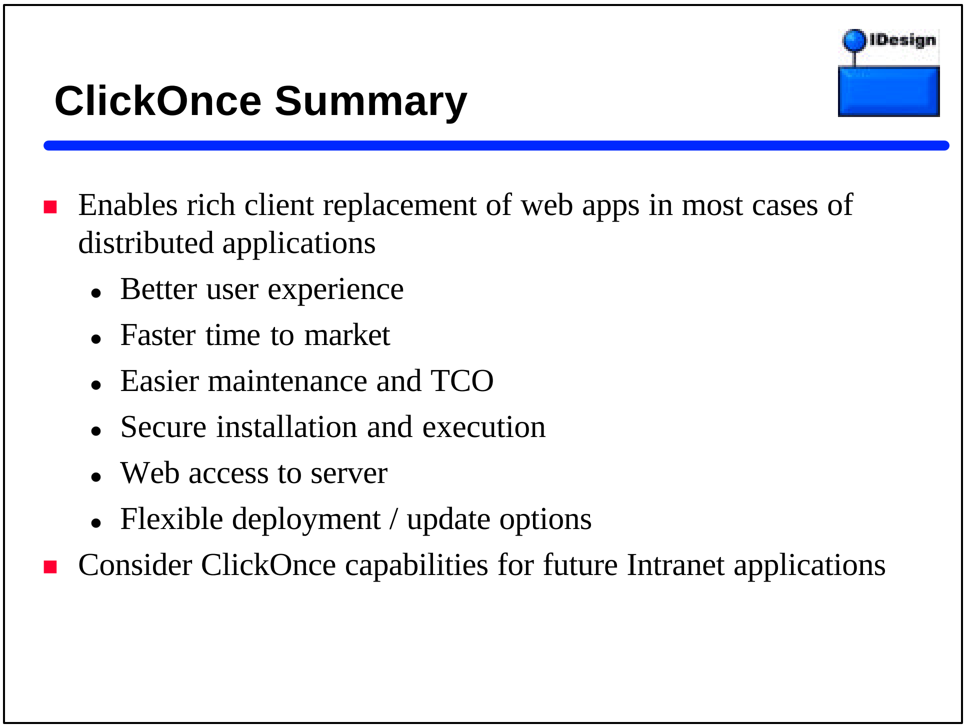
- Line-of-business rich client application
  - Including logic/resource components
  - Client machine requires .NET 2.0
- Connecting to business or data services:
  - Web Services
  - Database connection
  - Remoting
  - Enterprise Services
  - Indigo
- Application deployment and updates
  - Can be any web platform - .NET not required
    - But easiest with .NET

ClickOnce in Action



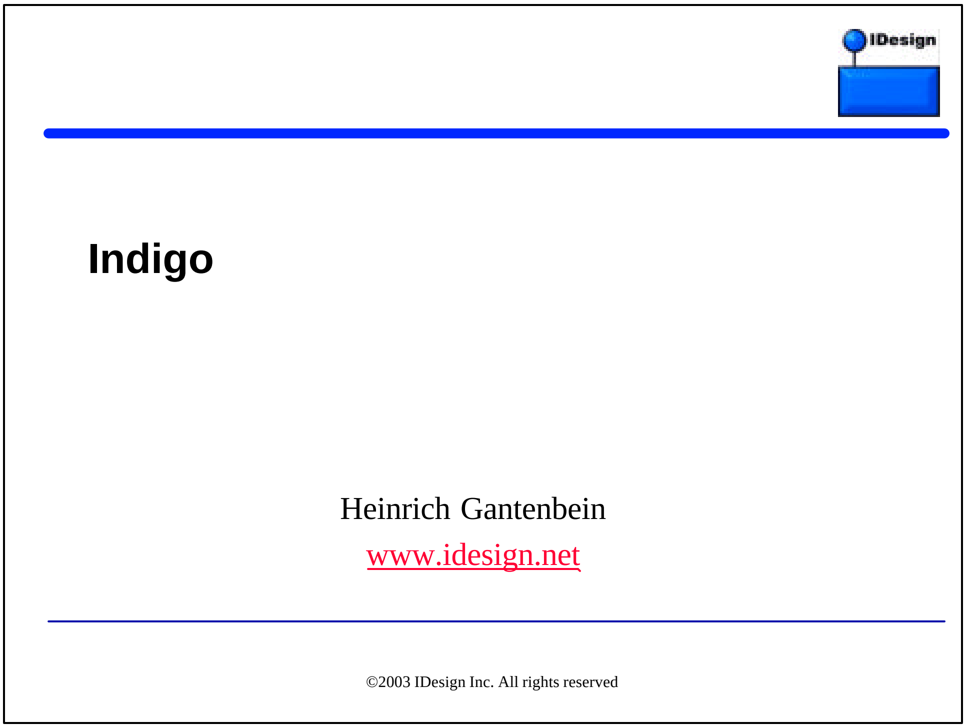
Scaling Out with ClickOnce





ClickOnce Summary

- Enables rich client replacement of web apps in most cases of distributed applications
  - Better user experience
  - Faster time to market
  - Easier maintenance and TCO
  - Secure installation and execution
  - Web access to server
  - Flexible deployment / update options
- Consider ClickOnce capabilities for future Intranet applications



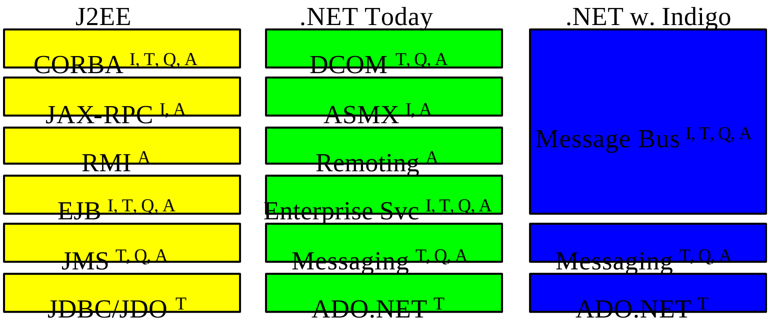
Indigo

Heinrich Gantenbein
www.idesign.net

Indigo Goals

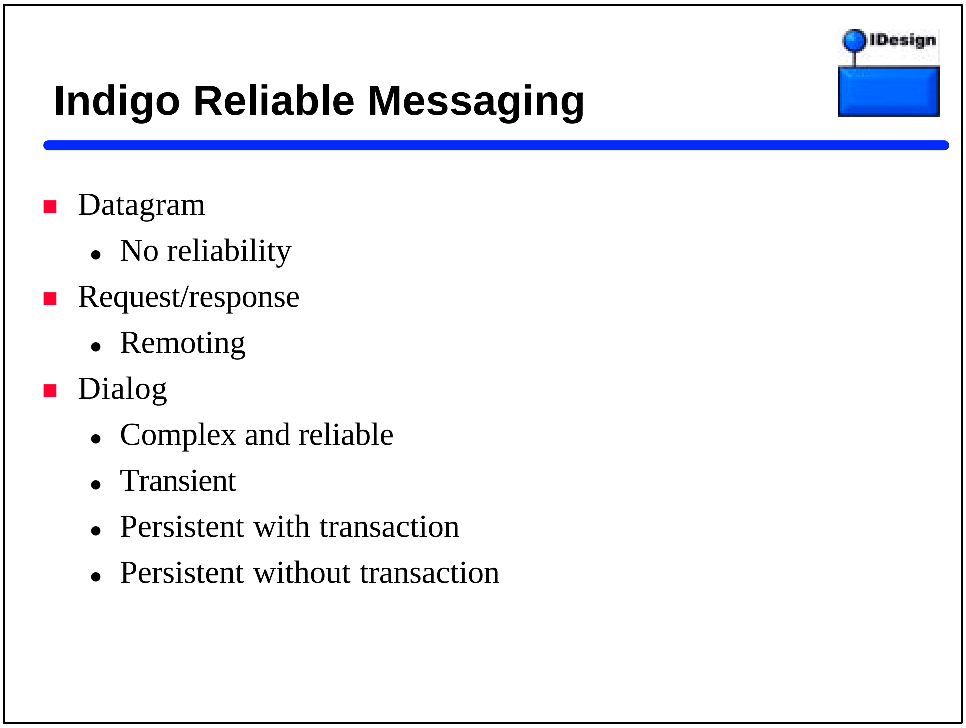
- Unify remoting component technologies
  - .NET Remoting
  - Enterprise services
  - Web services (including WSE)
  - Messaging (MSMQ, WS-Conversation)
- Provide for all components (including local)
  - Efficient, easy atomic transactions
  - Unified serialization architecture
- Compliance with WS-Standards and GXA

Distributed Component Stack



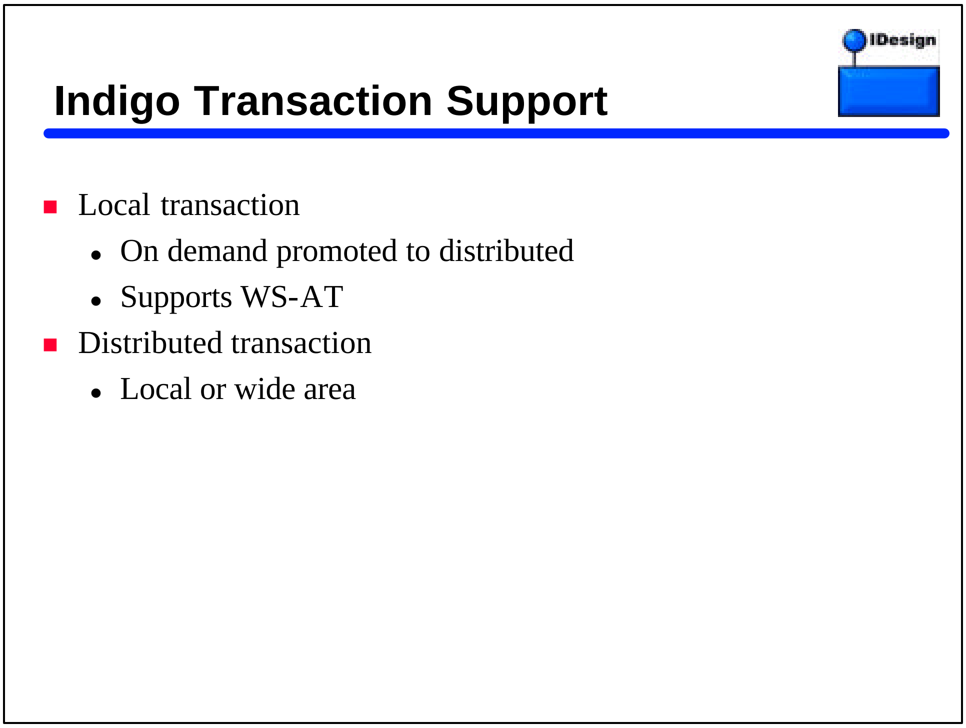
J2EE	.NET Today	.NET w. Indigo
CORBA $^{I, T, Q, A}$	DCOM $^{T, Q, A}$	Message Bus $^{I, T, Q, A}$
JAX-RPC $^{I, A}$	ASMX $^{I, A}$	
RMI ^{A}	Remoting ^{A}	
EJB $^{I, T, Q, A}$	Enterprise Svc $^{I, T, Q, A}$	
JMS $^{T, Q, A}$	Messaging $^{T, Q, A}$	Messaging $^{T, Q, A}$
JDBC/JDO ^{T}	ADO.NET ^{T}	ADO.NET ^{T}

I= interop, T=transaction, Q=queuing, A=async



Indigo Reliable Messaging

- Datagram
  - No reliability
- Request/response
  - Remoting
- Dialog
  - Complex and reliable
  - Transient
  - Persistent with transaction
  - Persistent without transaction



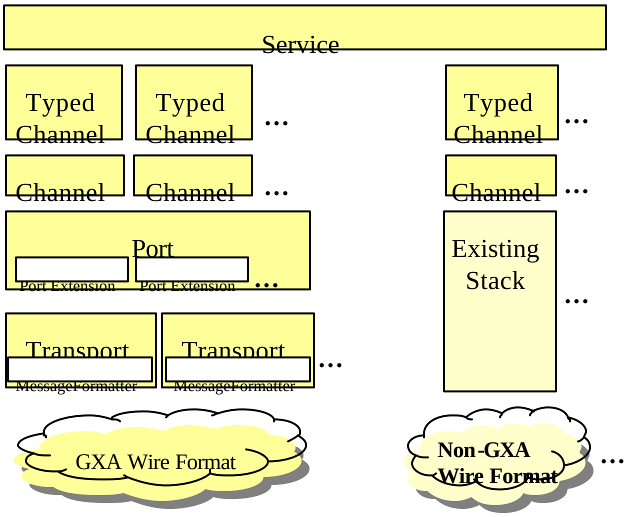
Indigo Transaction Support

- Local transaction
  - On demand promoted to distributed
  - Supports WS-AT
- Distributed transaction
  - Local or wide area

Indigo Security

- Security
- Policy
- Trust
- Secure conversation
- Privacy
- Federation
- Authorization

Indigo Programming



Service

Typed Channel

Typed Channel

…

Typed Channel

…

Channel

Channel

…

Channel

…

Port

Port Extension

Port Extension

…

Existing Stack

…

Transport

MessageFormatter

Transport

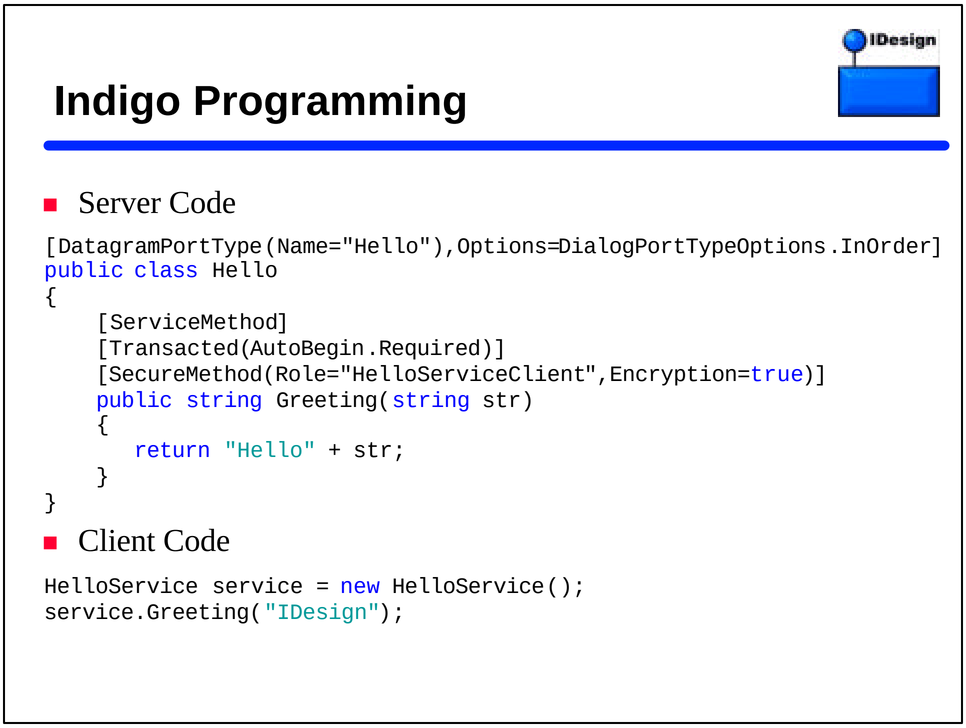
MessageFormatter

…

GXA Wire Format

Non-GXA Wire Format

…



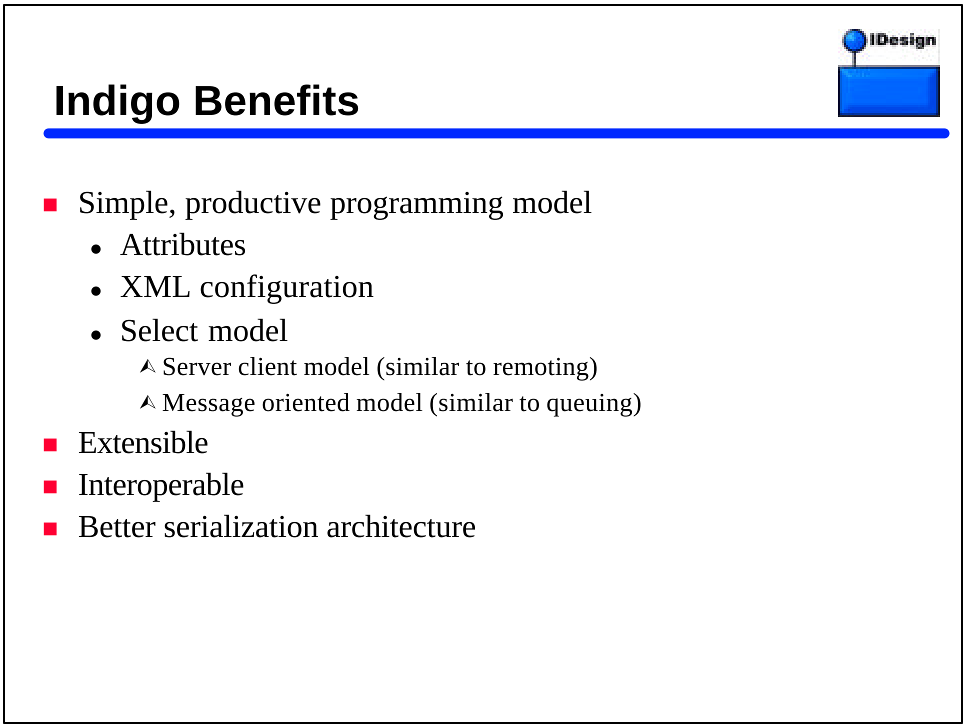
Indigo Programming

- Server Code

```
[DatagramPortType(Name="Hello"),Options=DialogPortTypeOptions.InOrder]
public class Hello
{
    [ServiceMethod]
    [Transacted(AutoBegin.Required)]
    [SecureMethod(Role="HelloServiceClient",Encryption=true)]
    public string Greeting(string str)
    {
       return "Hello" + str;
    }
}
```

- Client Code

```
HelloService service = new HelloService();
service.Greeting("IDesign");
```



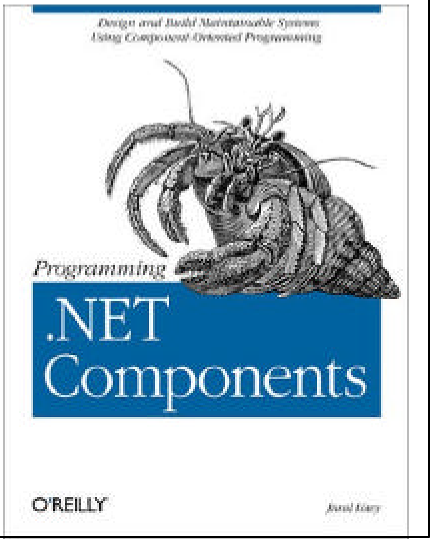
Indigo Benefits

- Simple, productive programming model
  - Attributes
  - XML configuration
  - Select model
    - Server client model (similar to remoting)
    - Message oriented model (similar to queuing)
- Extensible
- Interoperable
- Better serialization architecture

Q&A

Resources

- Programming .NET components
  - By Juval Lowy, O'Reilly 2003
- www.idesign.net
  - Code library
  - Coding standard
- .NET Master Class
  - 3-4 annually
  - Upcoming events on www.idesign.net


Design and Build Maintainable Systems
Using Component-Oriented Programming
Programming
.NET Components
O'REILLY
Juval Lowy