LOOMBIRD

# Was es nicht alles gibt!

## Einsatz von Tools und Frameworks in .NET-Projekten

Oliver Szymanski | Michael Wiedeking

oliver.szymanski@loombird.com | michael.wiedeking@loombird.com

www.loombird.com

LOOMBIRD

## Problemstellung

- Effiziente Software-Entwicklung
- Zielgerichtete Software-Entwicklung
- Gewünschte Qualität bei minimalem Aufwand (das ist erfahrungsgemäß für jeden etwas anderes)
  - Minimaler Zeitaufwand
  - Minimale Kosten
  - Maximale Qualität

  Randbemerkung:

  - das ist erfahrungsgemäß für jeden etwas anderes
  - das bezieht sich auf jeweils andere Teile

LOOMBIRD

# Bestandteile der Software-Entwicklung

- Analyse
- Design
- Implementierung
- Test
- Deployment
- Betrieb
- Wartung

LOOMBIRD

## Prozess-Schritte

- Dokumentation
- Qualitätssicherung
- Konfigurationsmanagement
- Build-Management
- Fehler-Management

LOOMBIRD

## Die vorgestellten Tools ...

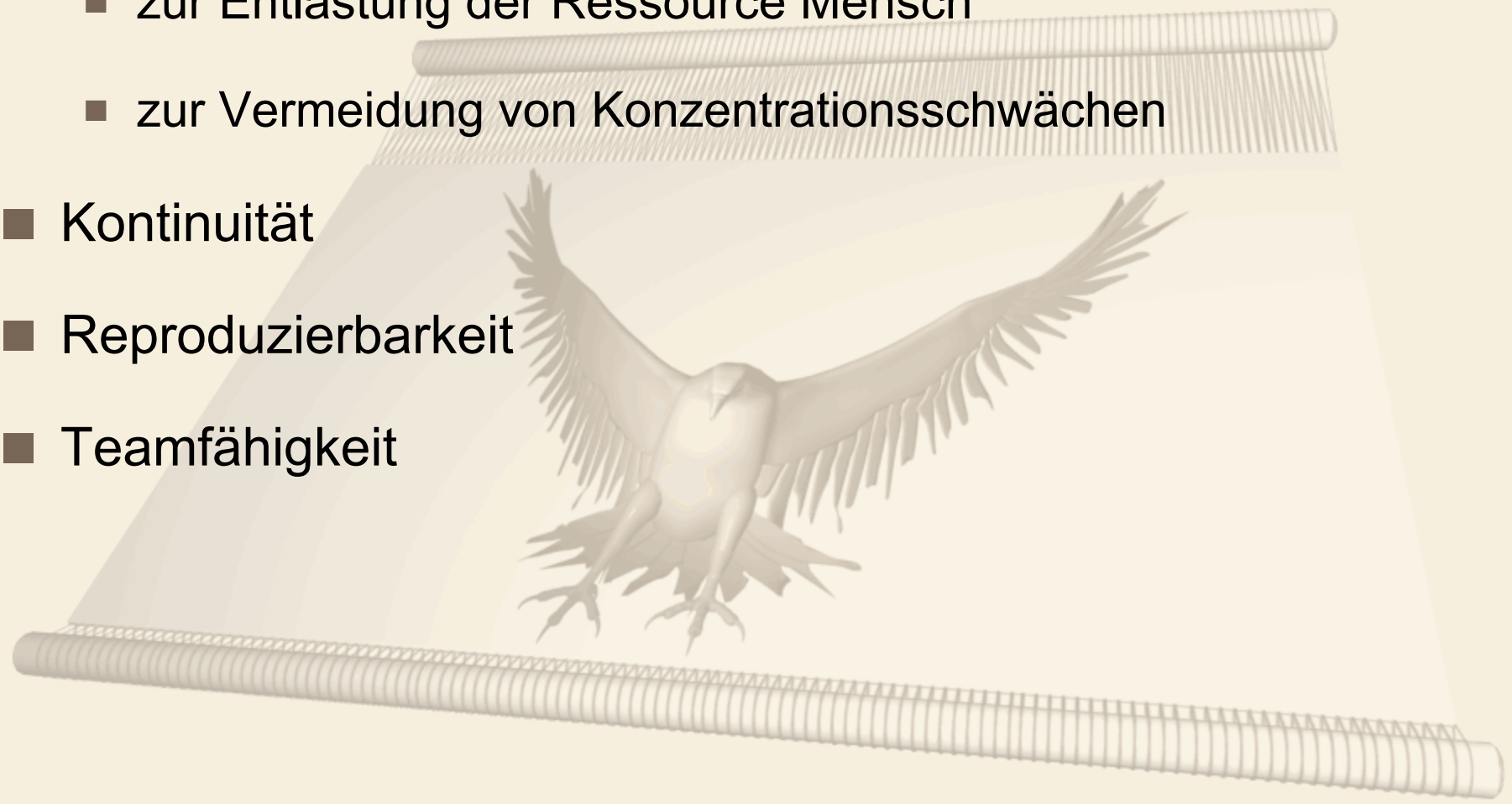
... konzentrieren sich auf

- Implementierung
- Test
- Deployment
- Betrieb
- Wartung

LOOMBIRD

# Erwartungen an Tools (1/2)

- Effizienz
  - zur Entlastung der Ressource Mensch
  - zur Vermeidung von Konzentrationsschwächen
- Kontinuität
- Reproduzierbarkeit
- Teamfähigkeit



LOOMBIRD

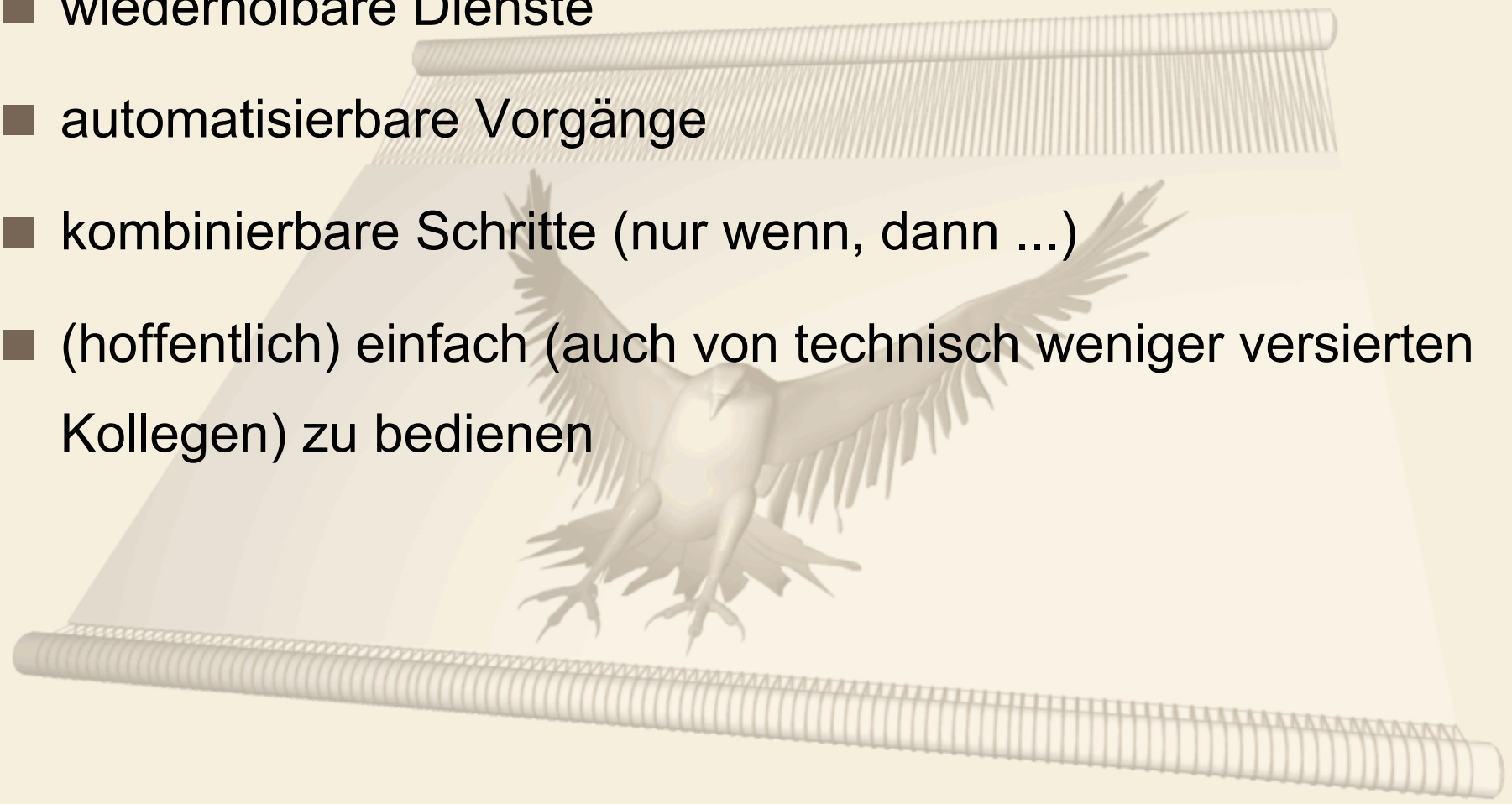
# Erwartungen an Tools (2/2)

- Transparenz
  - zur leichten Einarbeitung neuer Mitarbeiter
  - damit jeder alles versteht oder verstehen könnte
  - damit jeder alles machen kann oder machen könnte (eher technisch als fachlich)
- Rollentrennung
  - Betrieb findet im Rechenzentrum statt
  - Wartung wird von externer Firma durchgeführt

LOOMBIRD

## Die Lösung des Problems: Tools

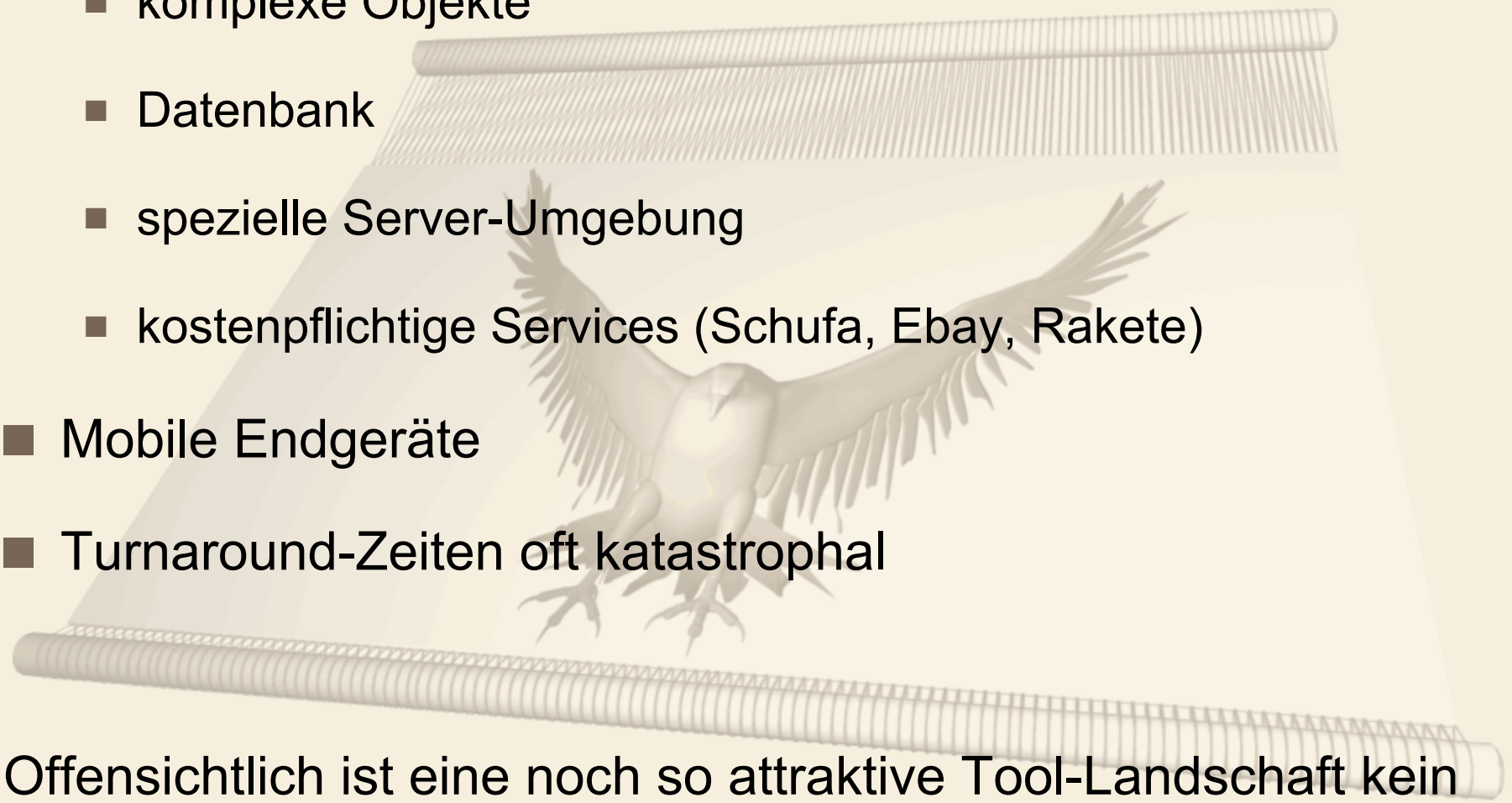
- spezifisch für ein konkretes Problem
- wiederholbare Dienste
- automatisierbare Vorgänge
- kombinierbare Schritte (nur wenn, dann ...)
- (hoffentlich) einfach (auch von technisch weniger versierten Kollegen) zu bedienen



LOOMBIRD

## Aber ...

- Unit-Tests oft schwierig wenn spezielle Umgebung nötig ist
  - komplexe Objekte
  - Datenbank
  - spezielle Server-Umgebung
  - kostenpflichtige Services (Schufa, Ebay, Rakete)
- Mobile Endgeräte
- Turnaround-Zeiten oft katastrophal



Offensichtlich ist eine noch so attraktive Tool-Landschaft kein Garant für eine verbesserte Entwicklungseffizienz

LOOMBIRD

# Lösung des Problems: Hilfsmittel

- Tools (wie oben)
- Patterns
- Konzepte
  - Wohldefinierter Prozess
  - Etabliertes Vorgehensmodell
- Loombird

LOOMBIRD

## Tool-Übersicht

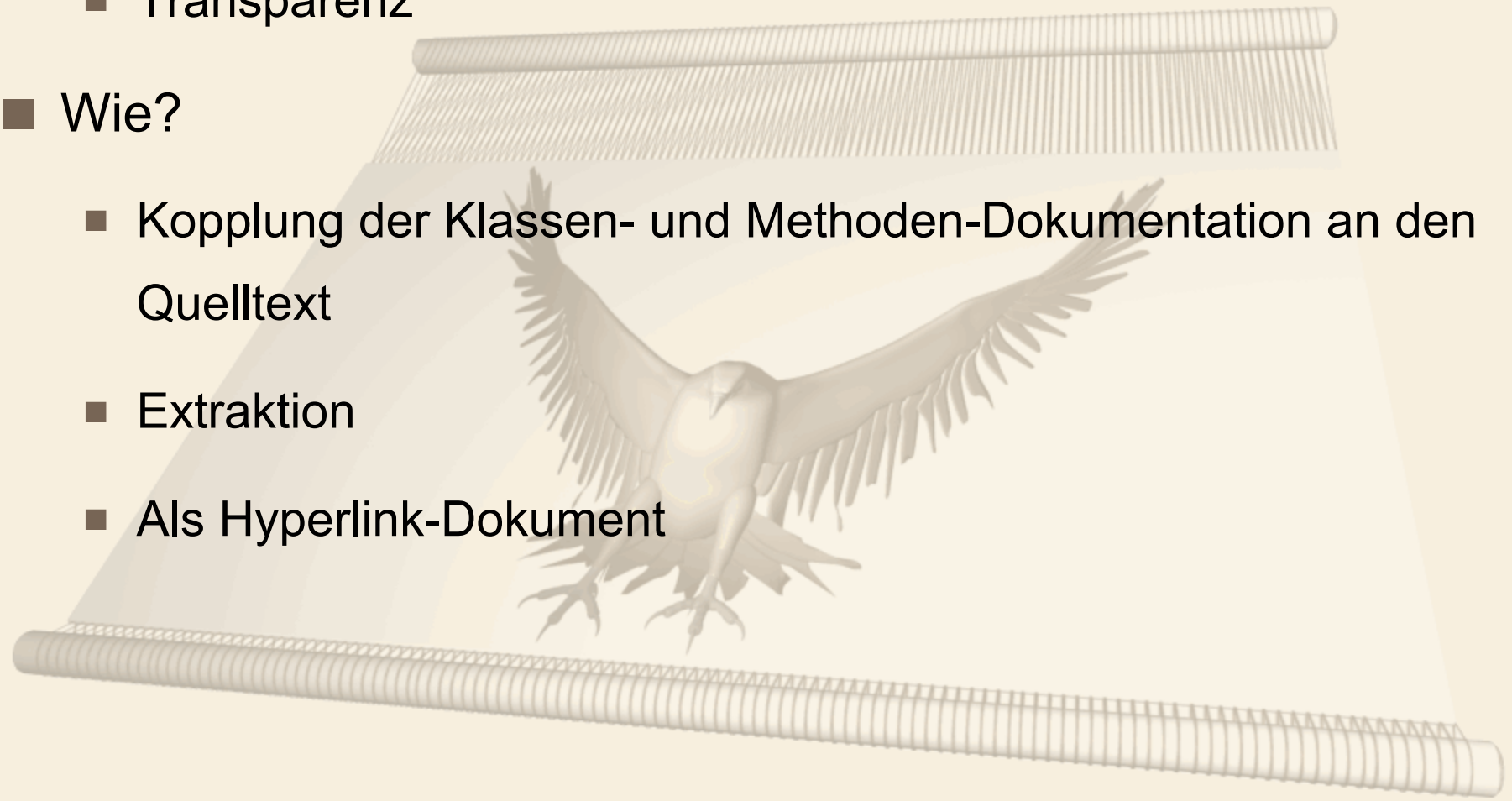
- NDoc
- NAnt
- NUnit
- Obfuscator
- ClickOnce
- Kleine Helferlein
- Log4Net
- NHibernate

- Loombird

LOOMBIRD

# Dokumentation

- Warum?
  - Transparenz
- Wie?
  - Kopplung der Klassen- und Methoden-Dokumentation an den Quelltext
  - Extraktion
  - Als Hyperlink-Dokument



LOOMBIRD

## NDoc

- NDoc erstellt aus Assemblies und XML-Dokumentationsdateien ein Class-Library-Dokumentationen
  - XML-Dokumentationsdateien erstellt der C#-Compiler oder AddOn-Tools wie VBCommenter
- NDoc unterstützt diverse Ausgabeformate wie
  - MSDN-Style HTML Help (*.chm),
  - VS.NET Help (HTML Help 2),
  - MSDN Online Style
- Weitere Formate sind durch AddOn-Documenter möglich
  - Es können eigene entwickelt werden

# NDoc: Erste Schritte

- Damit NDoc arbeiten kann, ist eine XML-Dokumentationsdatei des Assemblies erforderlich
  - dazu in Projekteinstellungen (bei C#) XML-Dokumentationsdatei aktivieren
  - Oder Parameter „/doc“ beim Kompilieren angeben
- Diese Dokumentationsdatei enthält die Code Kommentare, die man im Quellcode angegeben hat

LOOMBIRD

## Code Kommentare

- Code Kommentarzeilen werden mit „///“ eingeleitet und bestehen aus XML-Elementen
  - Beispiel:

```
public class MyClass() {

    /// <summary>
    ///
    /// </summary>
    /// <param name="s"></param>
    public MyClass( string s ) {
    }
}
```

- Innerhalb der Elemente wird die Beschreibung angegeben
- Das .NET-Framework definiert die Elemente
- VS .NET fügt Default-Elemente ein, wenn man „///“ eingibt

LOOMBIRD

## Kommentar-Tags

- <summary>Beschreibung</summary>
  - Beschreibung eines Typs
- <remark>Beschreibung</remark>
  - Zusätzliche Beschreibung eines Typs
- <returns>Beschreibung</returns>
  - Beschreibung einer Methodenrückgabe
- <seealso cref="Typ">Text</seealso>
  - Verweis auf einen Typ in See-Also-Section der Ausgabe
- <param name="name">Beschreibung</param>
  - Beschreibung eines Parameters

LOOMBIRD

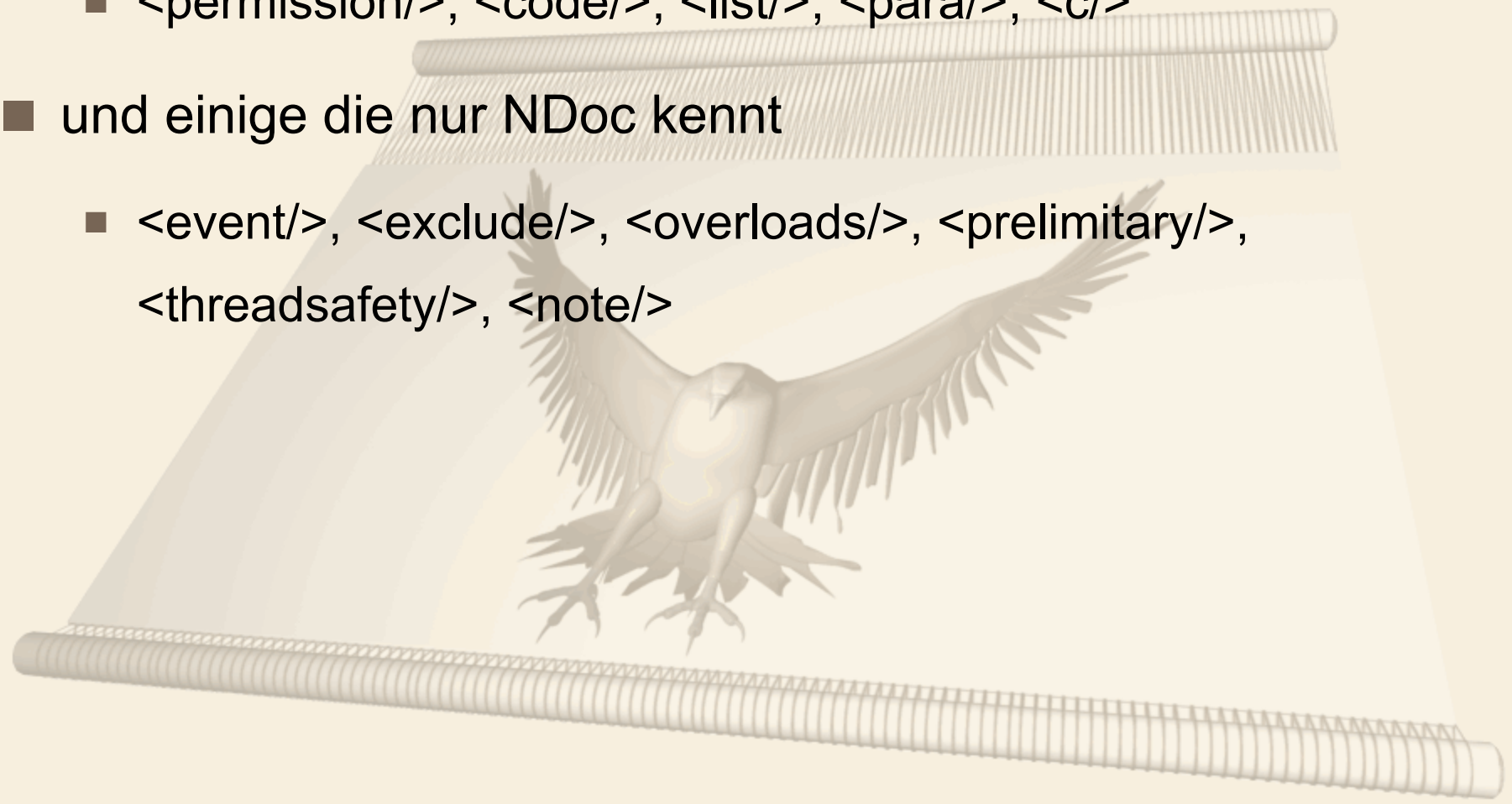
# Kommentar-Tags 2

- <exception cref="Typ">Beschreibung</exception>
  - Angabe welche Exception geworfen werden können
- <include file="Datei" path="xpath"/>
  - Inhalt eines XML-Elementes aus anderer Datei einbinden
- <example>Beschreibung</example>
  - Einfügen eines Beispieles
- <paramref name="name"/>
  - Referenz auf einen Parameter (z.B. in Summary)
- <see cref="Typ">Beschreibung</see>
  - Referenz auf einen Typen (z.B. in Summary)

LOOMBIRD

## Kommentar-Tags 3

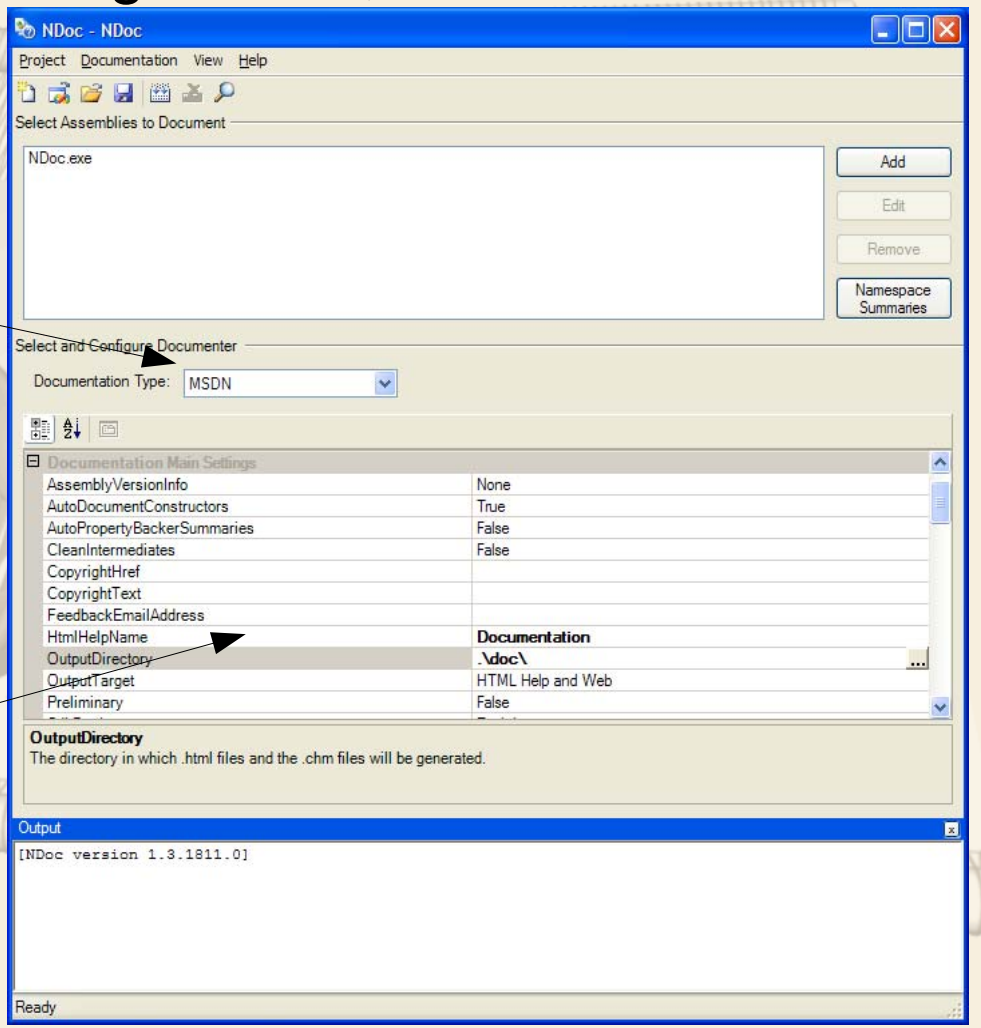
- Es gibt weitere:
  - <permission/>, <code/>, <list/>, <para/>, <c/>
- und einige die nur NDoc kennt
  - <event/>, <exclude/>, <overloads/>, <prelimitary/>, <threadsafety/>, <note/>



LOOMBIRD

# NDoc benutzen

- Hat man seinen Code (ausreichend ;-)) kommentiert und die XML-Dokumentationsdatei generiert, kann man NDoc aufrufen
- Documenter sind für die unterschiedlichen Ausgabeformate zuständig
- Die Documenter können konfiguriert werden



## NDoc im Buildprozess

- Mit NDoc Dokumentationen erstellen ist ein (je nach Projektgröße) zeitaufwendiger Vorgang
  - Deshalb nicht jedesmal neu generieren, sondern extra Build-Konfiguration für Dokumentation vorsehen
- Mit der NDoc GUI kann man NDoc Config-Dateien sichern und diese über ein Konsolenprogramm verwenden
- Das Konsolenprogramm eignet sich zur Integration im Buildprozess
- Man kann NDoc nach Kompilierung aufrufen
  - Post-Build Event in VS .NET oder SharpDevelop
  - Oder man verwendet NAnt

LOOMBIRD

## Aber ...

- Einbinden von Grafiken „schwierig“
- Uneinheitliche Ansatz
- Mäßige Unterstützung durch IDE

LOOMBIRD

# Build- und Deploy-Management

- Warum?
  - Zuverlässiges, identisches Zusammensetzen und Verteilen
  - Reproduzierbarkeit
- Wie?
  - Definition von Abläufen mit
    - Vorbedingungen
    - Aktion
    - Nachbedingungen
  - Bedingte Ausführung von Abläufen
  - Zeitlich gesteuerte Abläufe
  - Ereignisgesteuerte Abläufe

## NAnt

- Statt Shellkommando XML-basierte Konfigurationen zur Angabe von Tasks
- Nachteil: nicht so mächtige Kommandos wie find . -name exec rm {}
- Dafür Plattformunabhängigkeit
- Installation: entpacken und „\bin“-Verzeichnis in Path aufnehmen
- Starten: „nant“-Kommando
- Hilfe: Parameter „-help“

# NAnt Grundlagen

- NAnt benutzt XML-Konfigurationsdateien
- Eine Konfiguration besteht aus einem Projekt mit Targets die aus Tasks bestehen
  - z.B. das Target „test“ könnte aus dem Starten eines Servers, dem Deployen einer Anwendung und dem Starten des Client bestehen
- Auch kann für ein Projekt
  - eine Beschreibung
  - und Properties angegeben werden
    - im Buildfile oder per Kommandozeile oder global in der Datei nant.exe.config

LOOMBIRD

# Nant Beispielkonfiguration

```
<?xml version="1.0"?>
    <project name="Hello World" default="build" basedir=".">
        <description>The Hello World of build files.</description>
        <property name="debug" value="true" overwrite="false" />
        <target name="clean" description="remove all generated files">
            <delete file="HelloWorld.exe" failonerror="false" />
            <delete file="HelloWorld.pdb" failonerror="false" />
        </target>
        <target name="build" description="compiles the source code">
            <csc target="exe" output="HelloWorld.exe" debug="${debug}">
                <sources>
                    <includes name="HelloWorld.cs" />
                </sources>
            </csc>
        </target>
    </project>
```

LOOMBIRD

## NAnt Targets

- Targets können bedingt ausgeführt werden
- Targets können von anderen Targets abhängen
  - Wenn Target RUN von COMPILE abhängt und RUN ausgeführt wird, wird automatisch vorher COMPILE ausgeführt
- XML-Struktur

```
<target name="name" depends="t1,t2" if="expression"
  unless="expression" description="Beschreibung"/>
```

LOOMBIRD

## NAnt Tasks

- Einige der vordefinierten Tasks:
  - copy, delete, loadfile, mkdir, move
  - csc
  - cvs
  - exec
  - loadtasks
  - mail
  - unzip / zip
  - if / ifnot, foreach
  - nant, ndoc, nunit

## NAnt starten

- Beim Starten von NAnt wird die erste Datei mit Endung „build“ gesucht (oder mit Parameter „-buildfile:name“ angeben)
- NAnt führt als Parameter übergebene Targets aus
  - bei keiner Angabe in Buildfile definiertes Defaulttarget
  - z.B. „nant -buildfile:ndoc.build compile run“
- Parameter „-projecthelp“ listet Targets eines Buildfiles auf
- Eigenschaften werden mit dem Parameter „-D:name=wert“ angegeben

LOOMBIRD

## NAnt Listener und Logger

- Listener werden bei Zustandsänderungen im Buildprozess informiert: Start und Ende von Build, Target und Task
- Logger erweitern Listener: Zugang zu Output und Errorstream, LogLevel (-quit, -verbose, -debug)
  - NAnt.Core.DefaultLogger
    - Gibt Buildprozess-Informationen auf Console aus
  - NAnt.Core.MailLogger
    - Sendet Informationen über den Buildprozess an eMail-Listen
  - Nant.Core.XmlLogger
    - Formuliert Ausgabe als XML
- Angabe mit: „nant -listener:listenername -logger:loggername“

LOOMBIRD

## NAnt Expressions

- Mit Hilfe der Expressions können Ausdrücke ausgewertet und Funktionen in Attributwerten genutzt werden
- Innerhalb der Ausdrücke können die Properties angewendet werden (direkt mit dem Namen der Property)
- Es gibt eingebaute Funktionen und selbst definierte
- Expressions werden mit ${ eingeleitet und mit } beendet
- Operatoren: + - * / % > < == != and or usw.
- Beispiel:

```
${datetime::now() - file::get-last-write-time('out.dll')) >
   timespan::from-hours(1)}
```

LOOMBIRD

## NAnt Funktionen

- Assembly Funktionen: z.B. assembly::get-full-name(assembly)
- Conversion Funktionen: z.B. bool::to-string(value)
- Date/Time Funktionen: z.B. datetime::get-day(date)
- Directory Funktionen: z.B. directory::exists(path)
- Environment Funktionen: z.B. environment::get-variable(name)
- File Funktionen: z.B. file::get-length(file)
- Math Funktionen: z.B. math::abs(value)
- NAnt Funktionen: z.B. property::exists(name)
- Operating System, Path, String, Unix/Cygwin, Version Funktionen

## NAnt eigene Funktionen

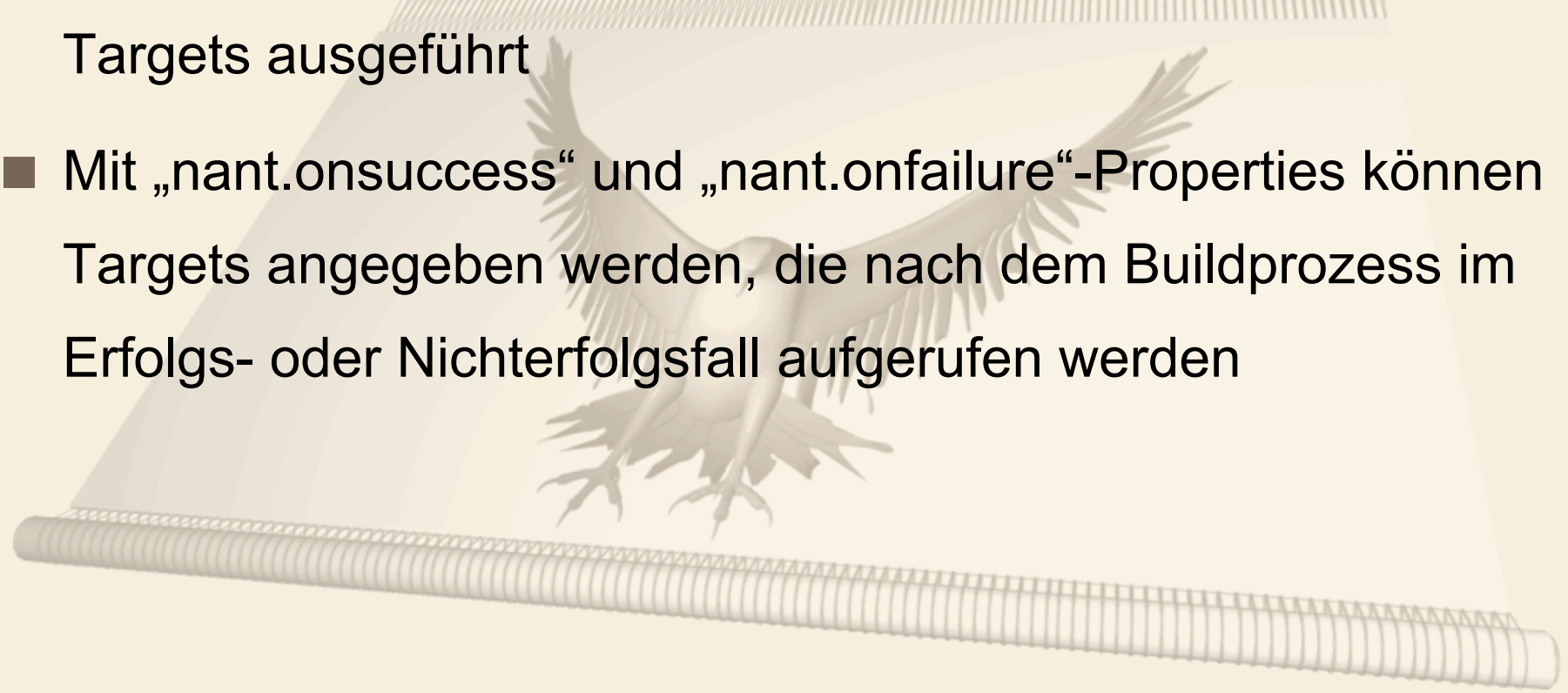
- Assembly mit Funktionen entweder in „nant\bin“-Verzeichnis ablegen oder mit Task „loadtasks“ im Buildfile laden oder mit Task „script“ neue Funktionen angeben

```
[FunctionSet("hello", "Hello")]
public class HelloFunctions : FunctionSetBase {

    public HelloFunctions(Project project,
                          PropertyDictionary properties)
                        : base(project, properties) {
    }

    [Function("hello-world")]
    public static string HelloWorldfunc() {
        return "Hello World!!";
    }
}
```

LOOMBIRD

## Änderungen zu Ant

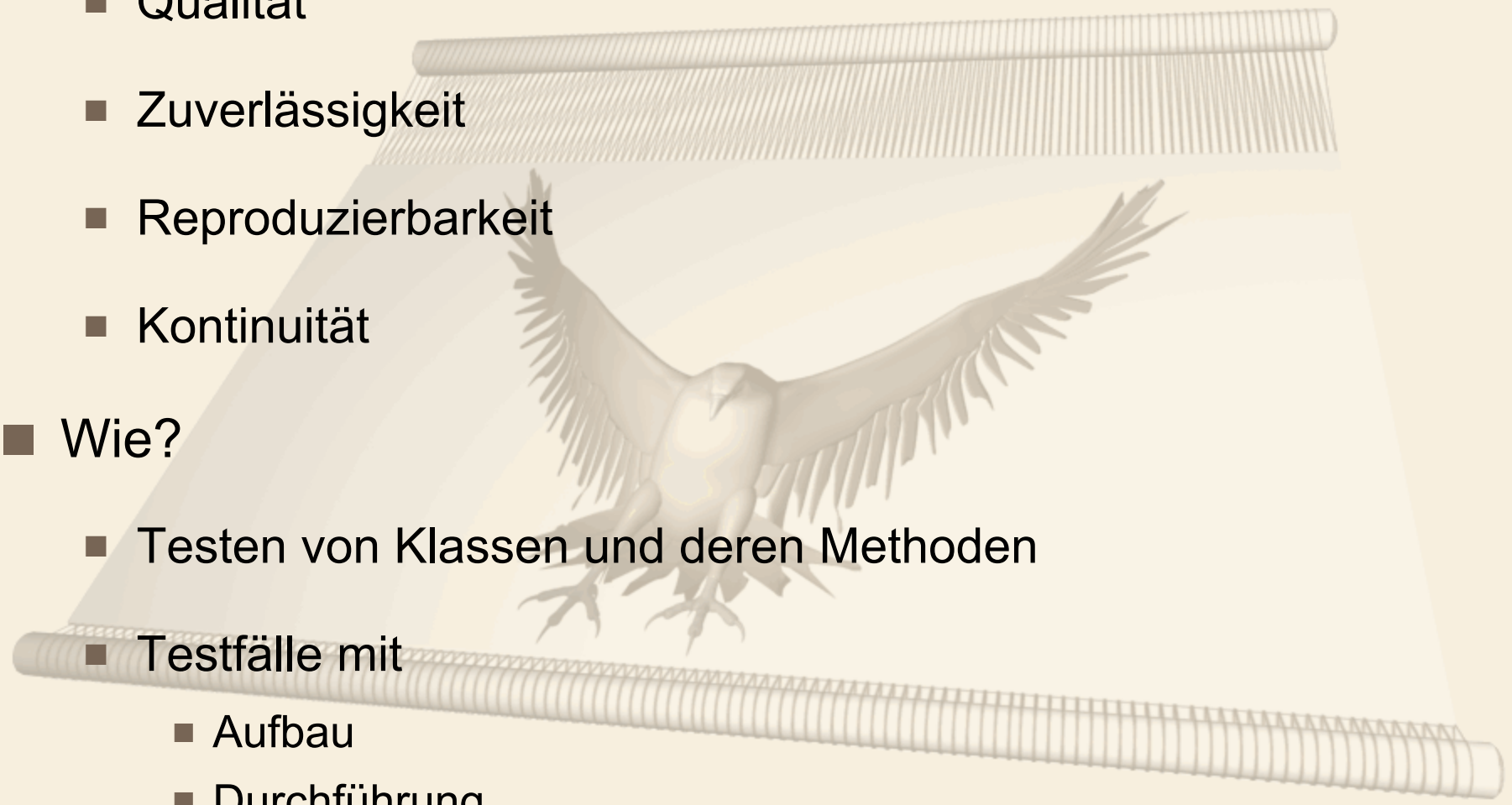
- NAnt sucht erste Datei mit Endung „build“, Ant nimmt build.xml
- In NAnt ist es möglich Tasks ohne Targets aufzunehmen, diese werden dann in angegebener Reihenfolge vor den Targets ausgeführt
- Mit „nant.onsuccess“ und „nant.onfailure“-Properties können Targets angegeben werden, die nach dem Buildprozess im Erfolgs- oder Nichterfolgsfall aufgerufen werden



LOOMBIRD

## Unit-Tests

- Warum?
  - Qualität
  - Zuverlässigkeit
  - Reproduzierbarkeit
  - Kontinuität
- Wie?
  - Testen von Klassen und deren Methoden
  - Testfälle mit
    - Aufbau
    - Durchführung
    - Abbau



LOOMBIRD

## NUnit

- NUnit ist ein UnitTest-Tool für das .NET-Framework
- Es ist vergleichbar mit JUnit, setzt aber statt Vererbung und Namenskonventionen auf Attribute
- Hauptattribute sind dabei
  - TestFixture: Klassen die Tests definieren
  - Test: Eine einzelne Testmethode
  - TestFixtureSetUp, TestFixtureTearDown:
    Methoden bevor und nachdem Tests einer TestFixture ausgeführt werden
  - SetUp, TearDown:
    Methoden bevor und nachdem jeweils ein Test ausgeführt wurde

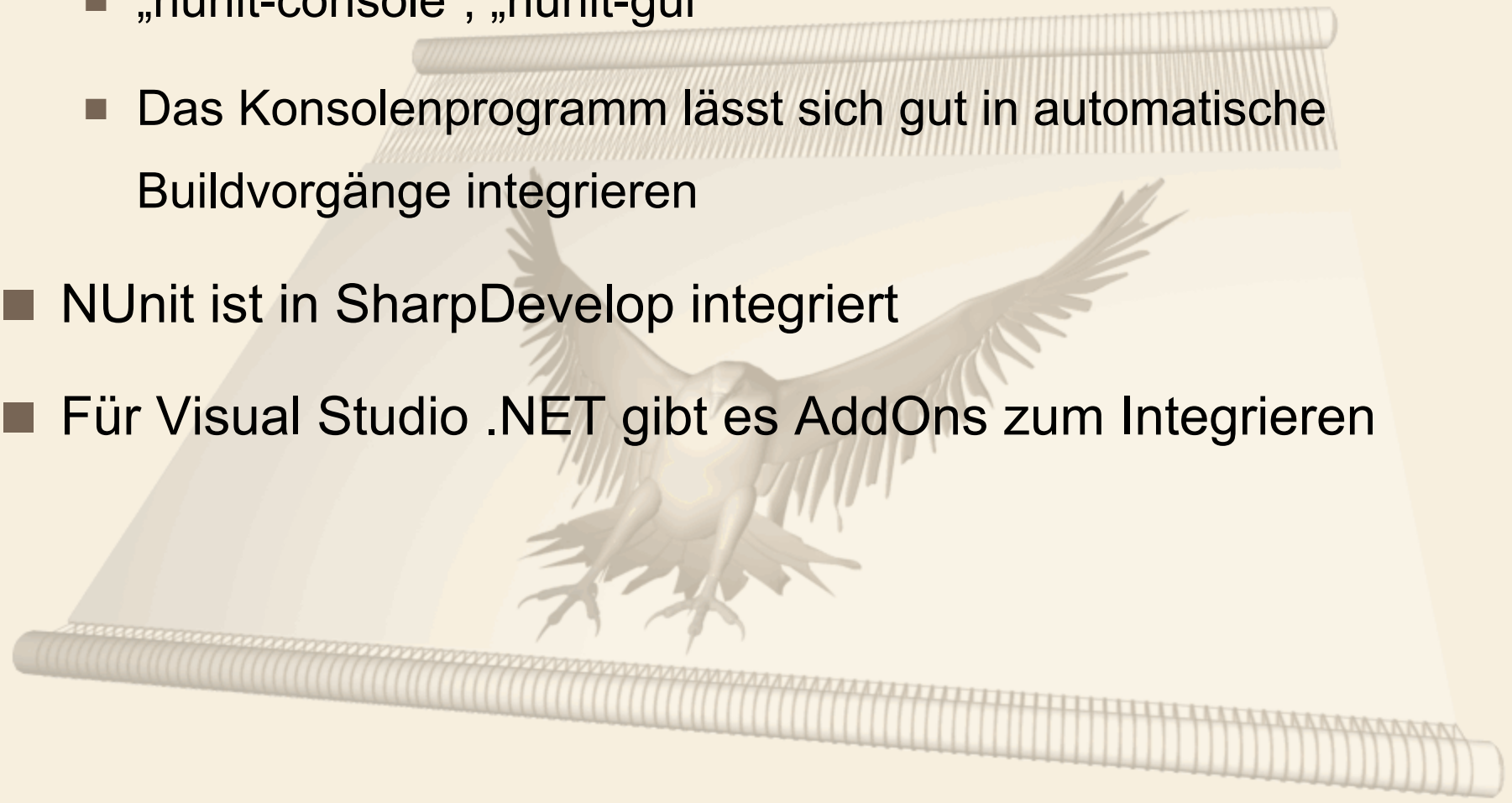
LOOMBIRD

## NUnit und Attribute

- Weitere zu nutzende Attribute:
  - ExpectedException: zur Angabe, dass innerhalb einer Testmethode eine bestimmte Exception ausgelöst werden sollte
  - Category: um Test zusätzlich zu ihrer Hierarchie zu gruppieren und nur bestimmte Test auszuführen
  - Explicit: gibt an, dass eine TestFixture oder ein Test nur durchgeführt wird, wenn er vom Benutzer explizit ausgewählt wurde
  - Suite: zur Bündelung von Test verwendbar, von der Benutzung wird allerdings abgeraten
  - Ignore: Einen Test beim Ablauf nicht ausführen (evtl. wenn er noch nicht implementiert wurde)

## Tests starten

- NUnit liefert eine Konsole- und eine GUI-Applikation
  - „nunit-console“, „nunit-gui“
  - Das Konsolenprogramm lässt sich gut in automatische Buildvorgänge integrieren
- NUnit ist in SharpDevelop integriert
- Für Visual Studio .NET gibt es AddOns zum Integrieren



## Aber ...

- Nachträgliche Implementierung in den meisten Fällen zum Scheitern verurteilt
- Erfordert diszipliniertes und kontinuierliches Vorgehen
- Wenn bei Fremdressourcen (z.B. Datenbank) die nötige Abstraktion eingehalten wird, so ist ein sinnvolles Testen praktisch nicht möglich
- Ein ’new C()’ verhindert das globale Austauschen von Klassen bzw. Objekten

LOOMBIRD

# Software-Verteilung

- Warum?
  - Kontinuität
  - Nur aktuelle Versionen im Einsatz
- Wie?
  - Client kann steuern
    - was geladen werden muss
    - wann geladen werden muss
  - Hollywood-Prinzip

    (rufen Sie nicht an, wir rufen Sie an)

LOOMBIRD

## ClickOnce

- Bereitstellungstechnologie für .NET-Windows Forms Clients (über HTTP)
- Ab Visual Studio 2005
- Anwendungen einfach installieren und aktualisieren
- Windows Forms-Clients statt Webanwendungen:
  - Effizientere Möglichkeiten der Benutzeroberfläche
  - Skripting bei HTML-Seiten entfällt
  - Funktionalität auf Client realisierbar
  - Drucken leichter beeinflussbar

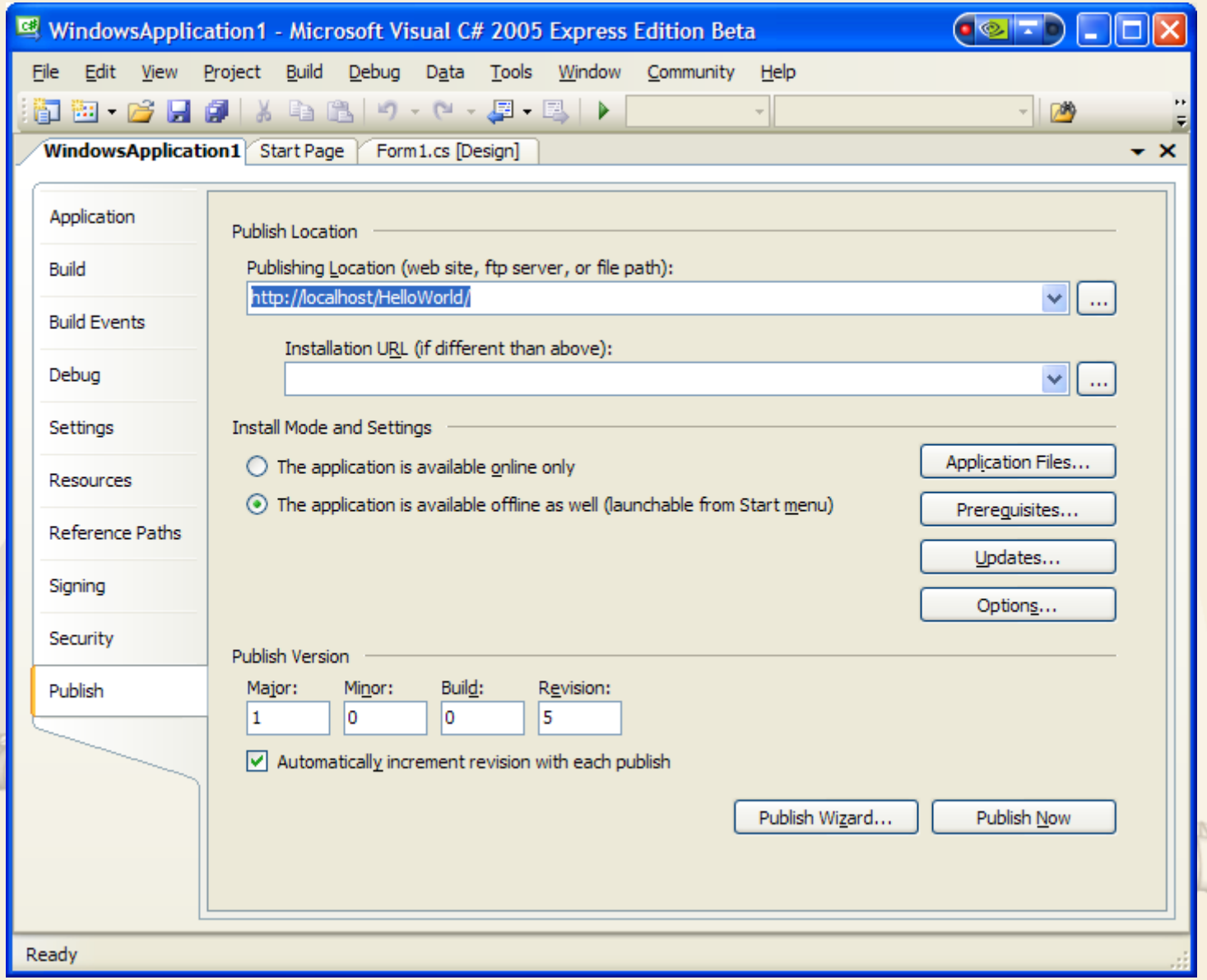
LOOMBIRD

## Click Once

- Transaktionsorientierte Aktualisierungen
- Anwendung kann selbst feststellen ob sie Online/Offline ausgeführt wird
- Automatische oder vom Programm selbst gesteuerte (per Code) Aktualisierung möglich
- Per ausführbarer Win32-"Bootstrapper"-Datei ist es sogar möglich beim ersten „Click“ das .NET-Framework zu installieren (sowie weitere erforderliche Komponenten)
- Anwendungen können im Startmenü verlinkt werden
- Load-On-Demand bei Assemblies/Ressourcen

LOOMBIRD

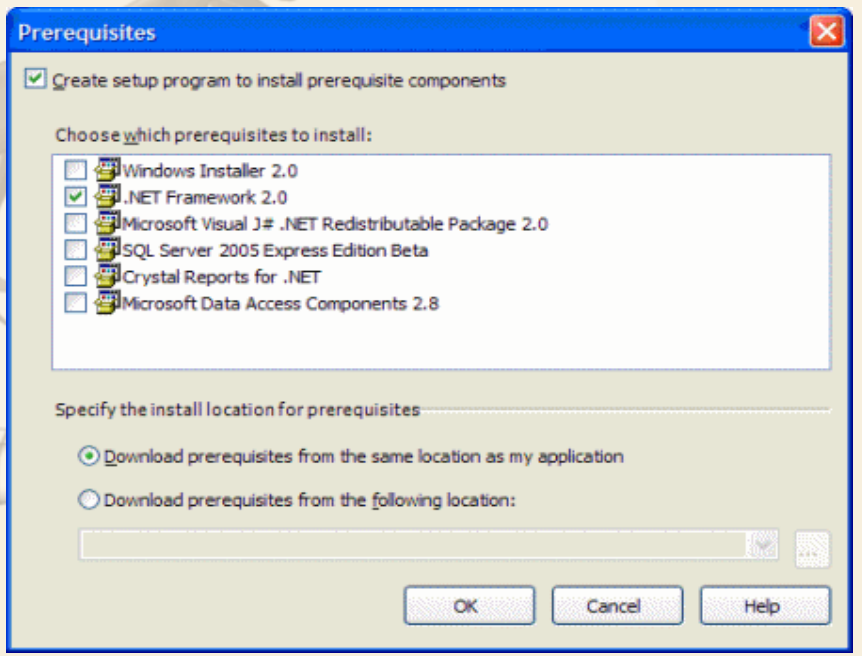
# Click Once



LOOMBIRD

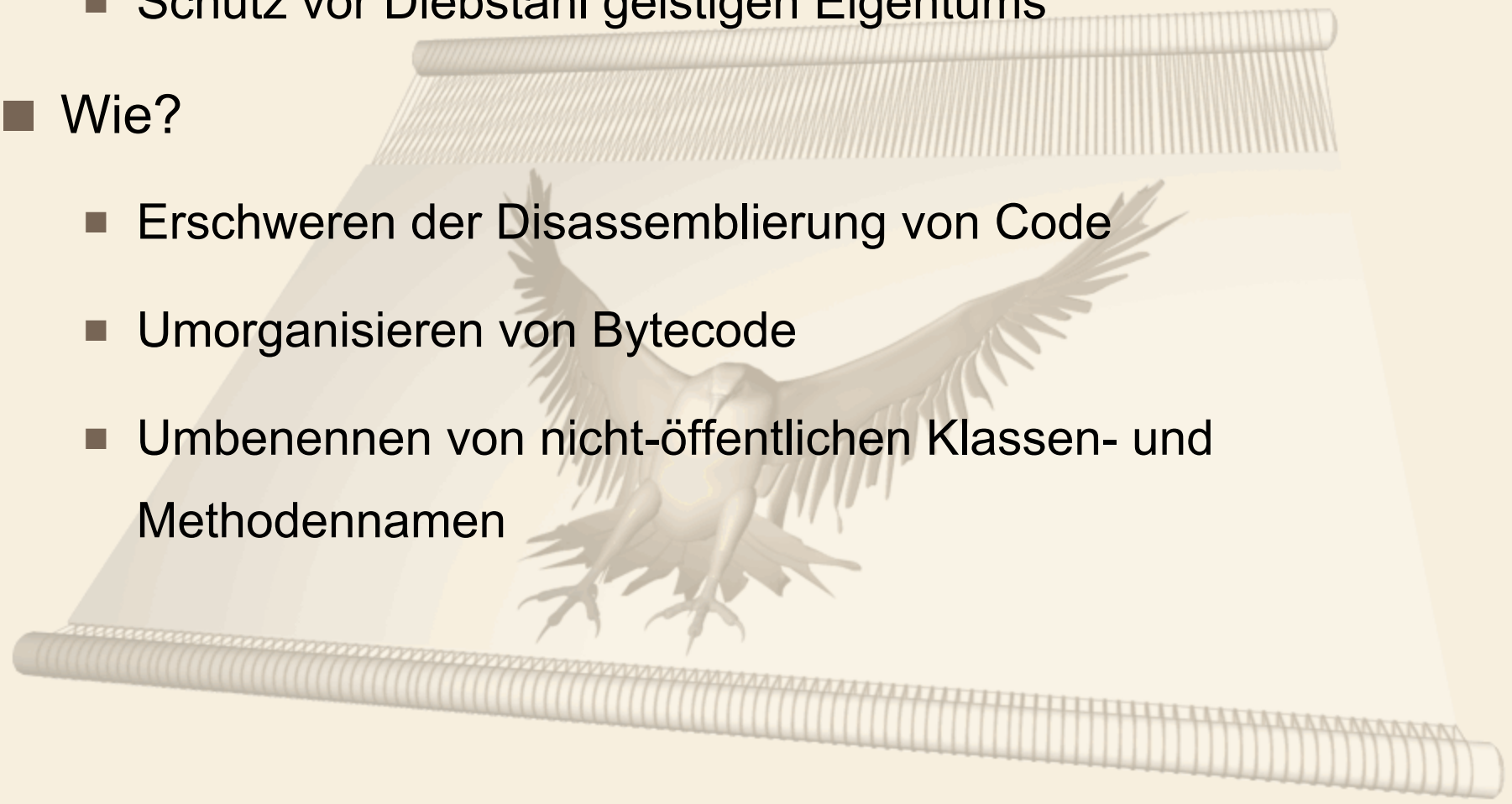
# Click Once

- Click Once wird in Visual Studio 2005 über Publish als Projekteigenschaften konfiguriert
- Man kann angeben, wo die Anwendung hinterlegt ist (Webserver oder URL), welche Dateien dazugehören, wie und wann aktualisiert wird und welche Komponenten auf dem Client erforderlich sind
- Auch eine Website für das Publish kann als Datei angegebenwerden



# Obfuscating

- Warum?
  - Schutz vor Diebstahl geistigen Eigentums
- Wie?
  - Erschweren der Disassemblierung von Code
  - Umorganisieren von Bytecode
  - Umbenennen von nicht-öffentlichen Klassen- und Methodennamen



LOOMBIRD

## Obfuscator

- Eine Vielzahl an Obfuscators sind für .NET erhältlich
  - Reine Obfuscators
    - Demeanor
    - Dotfuscator (Mapping File/Utility für StackTraces, auch .NET CF)
    - Salamander .NET Obfuscator (auch .NET CF, Managed und Unmanaged Code)
  - Tool-Sammlungen
    - XenoCode (Code Protection, Deployment, Optimization)
    - Thininstall (Statt Obfuscator hier Verschlüsselung von Assembly, Install / Start von CD, Verlinkung von benötigten DLLs/Assemblies, eigene Trials)

LOOMBIRD

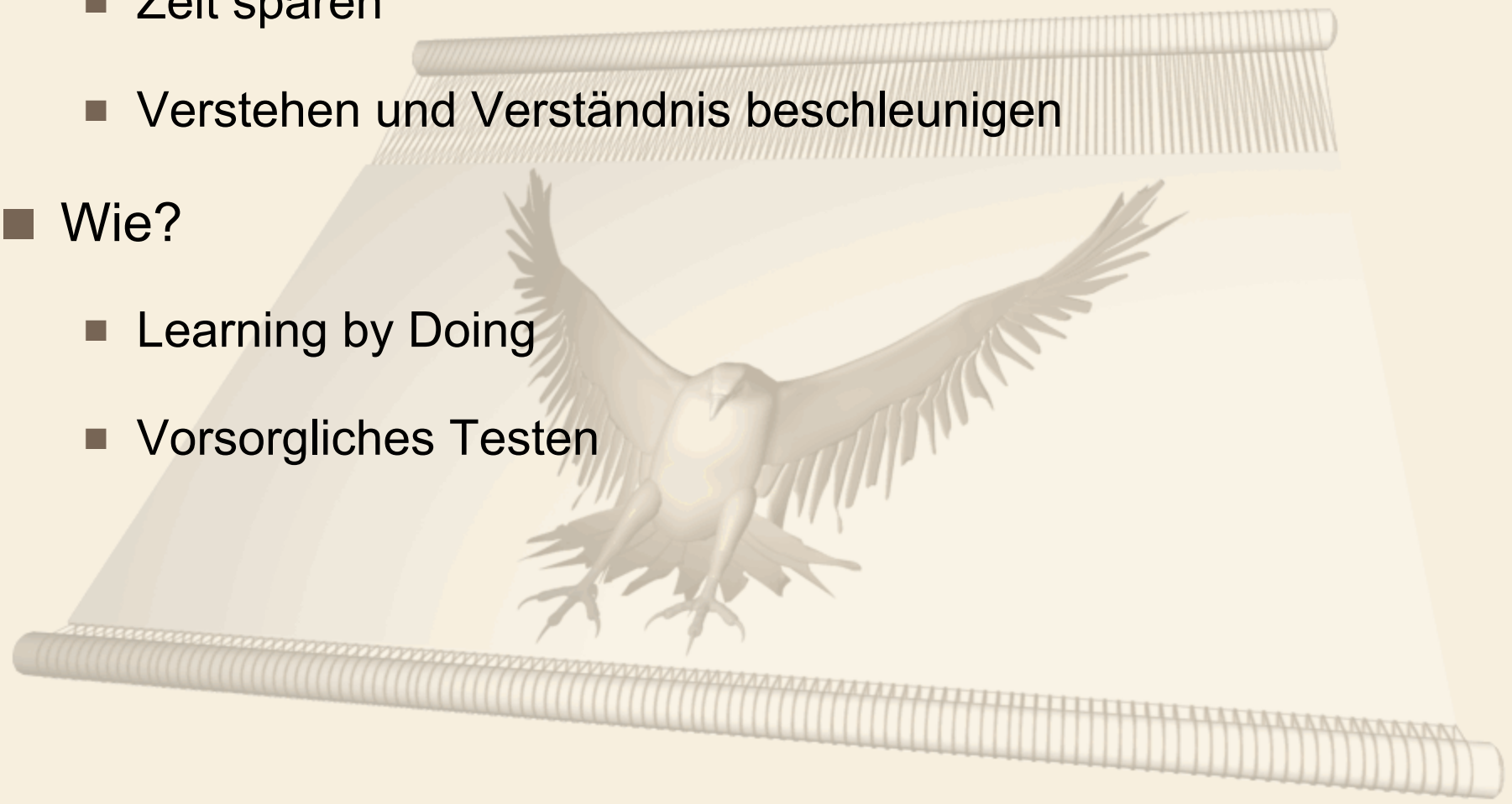
# Obfuscator Grundlagen

- Features
  - Symbolic Names und Metadata
  - String Protection
  - Control Flow
  - Decompiler Protection
  - Optimization (Entfernung von unnötigem Code/Metadaten)
- Probleme
  - StackTrace Integrität (Zeilennummern, Methodennamen, etc.)
  - Inkrementelle Änderungen (Update beim Client)
  - Reflection (wg. Umbenennung von Methoden)

LOOMBIRD

## Kleine Helferlein

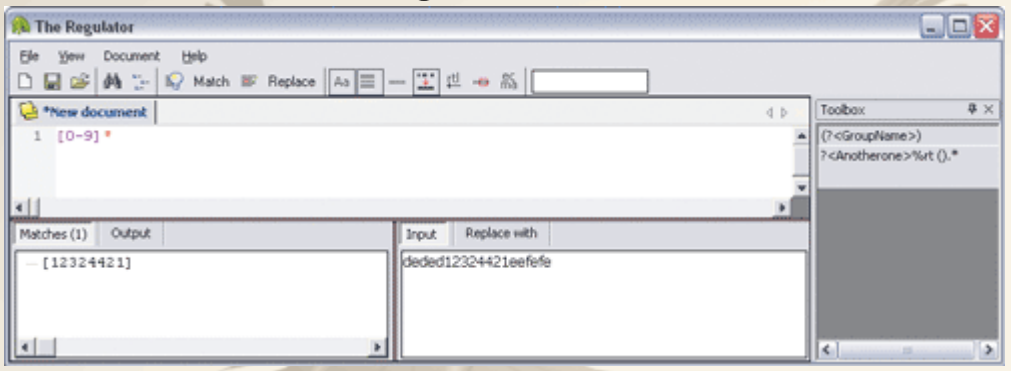
- Warum?
  - Zeit sparen
  - Verstehen und Verständnis beschleunigen
- Wie?
  - Learning by Doing
  - Vorsorgliches Testen



LOOMBIRD

# Regulator für Regular Expressions

- Tool für die Eingabe und Tests von Regulären Ausdrücken
- Reguläre Ausdrücke geben Muster für Zeichenketten an
  - Bietet sich an um Benutzereingaben zu verifizieren oder
  - für Suchen und Ersetzungen in Zeichenketten

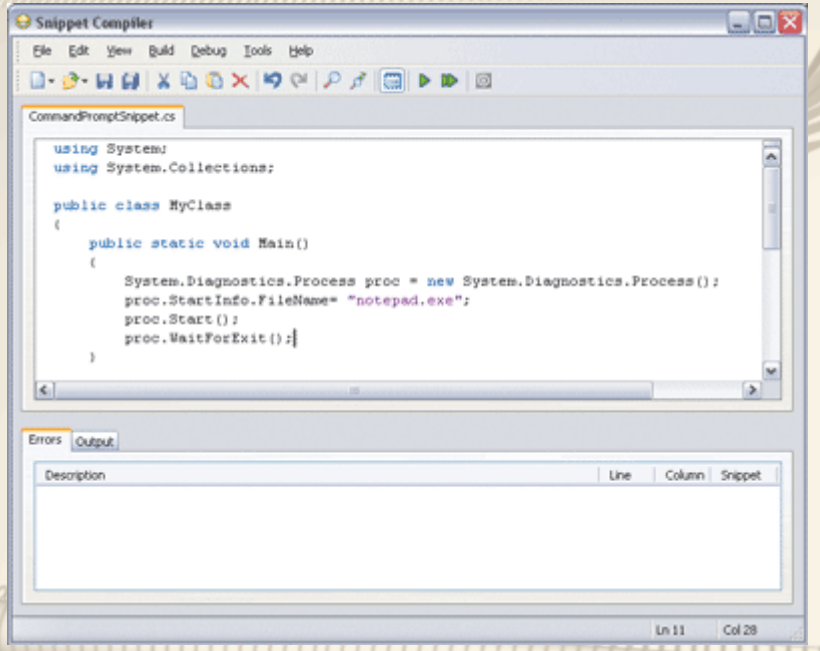



- In Regulator kann man Eingaben, auf denen die Regulären Ausdrücke getestet werden, angeben
- Es gibt eine Anbindung an regexlib.com für vordefinierte Reguläre Ausdrücke (z.B. für Telefonnummern)

LOOMBIRD

# Snippet Compiler

- Code Snippet dient zum Testen kurzer Codeausschnitte, für die sich kein eigenes Projekt lohnt
- Gut um Beispiele zu Demonstrieren, etc.



LOOMBIRD

## Code Smith

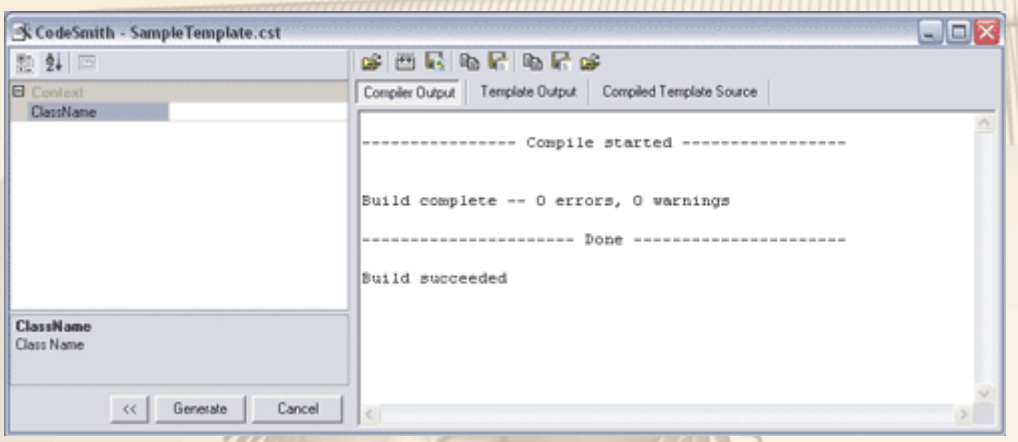
- Mit Code Smith kann man Code Templates einfach definieren
- Template Definitionen per Datei und aus Datenbanken

```
<%@ CodeTemplate Language="C#" TargetLanguage="C#"
  Description="Singleton" %>
<%@ Property Name="ClassName" Type="String" Category="Context"
   Description="Class Name" %>

public sealed class <%= ClassName %>
{
    private static volatile <%= ClassName %> _instance;
    private <%= ClassName %>() {}
    private static readonly object _syncRoot = new object();

    public static <%= ClassName %> Value
    { get {
...
```

LOOMBIRD

# Code Smith

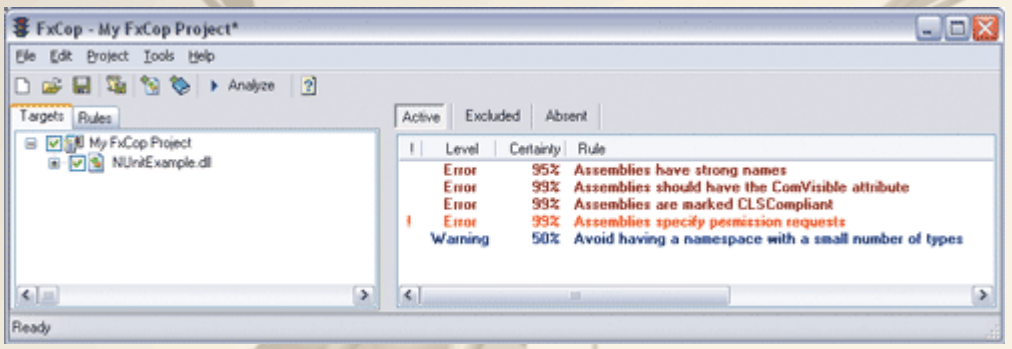
- Property werden definiert und als Werte in den Templates ersetzt, Syntax dabei ist wie in ASP.NET



LOOMBIRD

# FxCop

- FxCop ermöglicht es .NET-Code anhand von Regeln zu prüfen
- Es gibt vordefinierte Regeln von Microsoft und man kann eigene Regeln definieren
  - z.B. Vorhandensein eines Default-Konstruktors (kein Parameter)

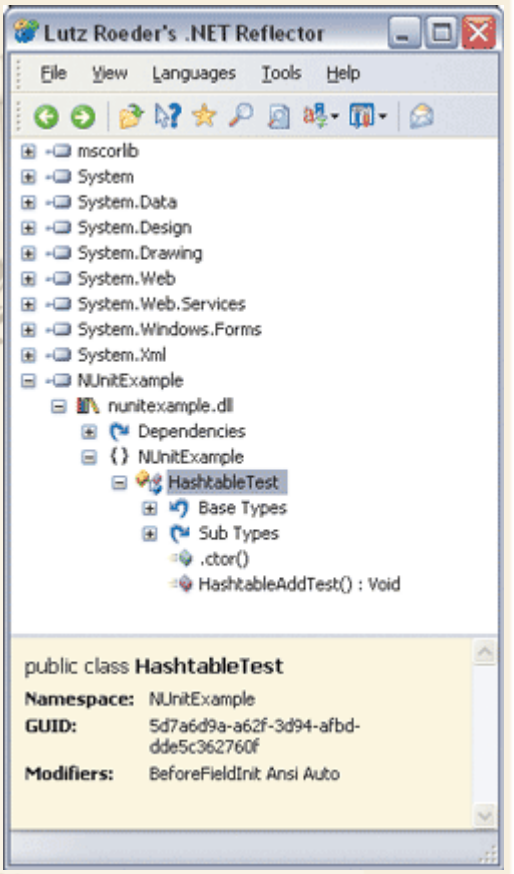



- In FxCop legt man Projekt an und fügt zu prüfende Assemblies hinzu
- FxCop analysiert dann Assemblies und listet Error/Warnings auf die man detailliert anschauen kann

LOOMBIRD

# Reflector

- Reflector ist ein Class Browser und Decompiler für .NET-Assemblies
- Reflector nutzt dazu die .NET-Reflection-API
- Reflector zeigt den Microsoft Intermediate Language (MSIL) Code und nach decompilieren Quellcode
- Nützlich um Nachzusehen, wie Microsoft selbst Aufgaben im Framework gelöst hat



LOOMBIRD

## Weitere Tools

- Mehr Tools z.B. für C# siehe
  - http://msdn.microsoft.com/vcsharp/team/tools/default.aspx
- SharpZipLib zum Umgang mit Zip-Dateien
  - http://www.icsharpcode.net
  - Auch für Zip/Unzip Ant-Task benötigt
- TestRunner for NUnit
  - AddOn für VS .NET
- OpenLicense als Lizenzmanager
  - http://openlicense.tigris.org

LOOMBIRD

## Log4Net

- Logging leichtgemacht
- Vorteile zu Console.Writeline: konfigurierbar was zur Laufzeit wann geloggt werden soll
- Unterschiedliche Ziele, Formate und Inhalte sowie Loglevel
- Basiskomponenten: Logger, Logmanager, Appender, Layouts
- Logger sind Entitäten mit Namen die hierarchisch aufgebaut werden können
  - Logger ist Kind eines anderen, wenn der Vatername als Präfix gefolgt von „.“ im Kindnamen vorkommt
  - „foo.bar.child“ ist somit Kind von „foo.bar“

LOOMBIRD

## Logger

- Logger sind Entitäten mit Namen die hierarchisch aufgebaut werden können
  - Logger ist Kind eines anderen, wenn der Vatername als Präfix gefolgt von „.“ im Kindnamen vorkommt
  - „foo.bar.child“ ist somit Kind von „foo.bar“
- Root Logger: hat keinen Namen und ist Basis für alle anderen
- Logger haben Loglevel (wird geerbt falls nicht angegeben):
  - ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF
  - Root Logger hat Level DEBUG
  - Ein Logger loggt nur Ausgaben, wenn deren Level >= dem Level des Loggers ist

LOOMBIRD

## Logger-Methoden

- bool IsDebugEnabled { get; }
- void Debug(object message);
- void Debug(object message, Exception t);
- void DebugFormat(string format, params object[] args);
  - Log message string using the System.String.Format syntax
- void DebugFormat(IFormatProvider provider, string format, params object[] args);
  - Log message string using the System.String.Format syntax
- Hier für DEBUG, für alle anderen Level ebenso

LOOMBIRD

## LogManager

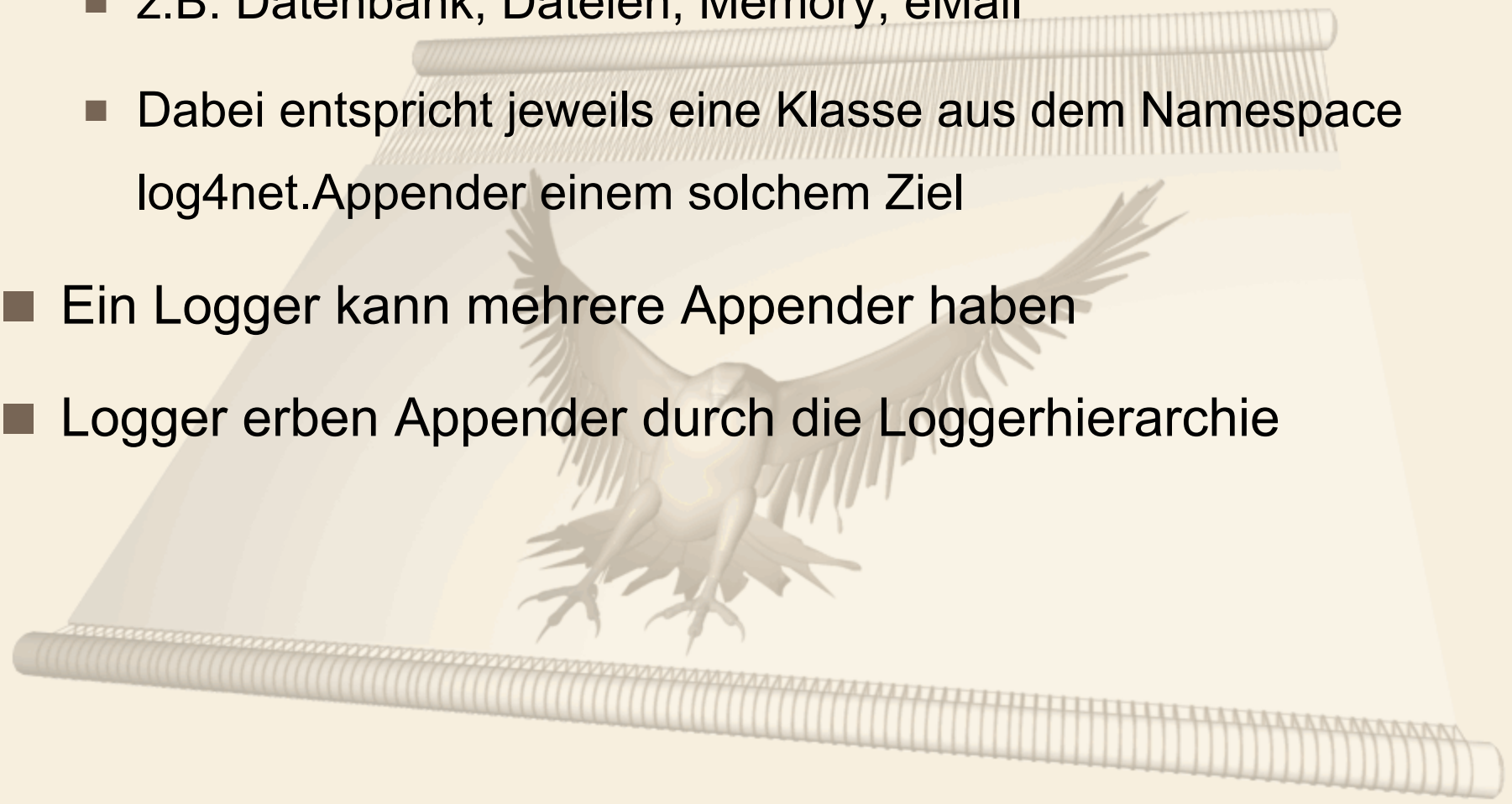
- Logger erhält man über statische Methoden der Klasse LogManager
- Jeder Aufruf einer der Methoden liefert bei gleichen Parametern dasselbe Objekt

```
public static ILog GetLogger(string name);
public static ILog GetLogger(Type type);
```

- Name entspricht dabei dem Namen des Loggers
- Bei Type wird als Name der fully qualified Klassenname genutzt

## Appender und Filter

- Appender stellen die unterschiedlichen Ausgabeziele dar
  - z.B. Datenbank, Dateien, Memory, eMail
  - Dabei entspricht jeweils eine Klasse aus dem Namespace log4net.Appender einem solchem Ziel
- Ein Logger kann mehrere Appender haben
- Logger erben Appender durch die Loggerhierarchie



## Filter

- Filter können Appender nur bestimmte Ausgaben schreiben lassen
- Filter stammen aus dem Namespace log4net.Filter
  - DenyAllFilter, LevelMatchFilter, LevelRangeFilter, LoggerMatchFilter, PropertyFilter, StringMatchFilter

## Layouts

- Layouts können Appender hinzugefügt werden um die Ausgabe zu gestalten
  - ExceptionLayout - Renders the exception text from logging event
  - PatternLayout - Formats the logging event according to a flexible set of formatting flags
  - RawTimeStampLayout - Extracts timestamp from logging event
  - RawUtcTimeStampLayout - Extracts the timestamp from the logging event in Universal Time
  - SimpleLayout - Formats simply: [level] – [message]
  - XmlLayout Formats the logging event as an XML element
  - XmlLayoutSchemaLog4j

## Object Renderer

- Will man Objekte bestimmter Klassen ausgeben und diese sollen speziell formatiert werden, kann man ObjectRender dazu anmelden
- Diese ObjectRender kann man selbst erzeugen (Implementierung des Interface log4net.ObjectRenderer.IObjectRenderer)
- ObjectRender unterstützen auch Klassenhierarchien
  - z.B. wird ein ObjectRender für die Klasse Frucht auch für die abgeleiteten Klassen Apfel und Banane genutzt, solange nicht ein speziell Renderer für diese angemeldet wird

LOOMBIRD

## Konfiguration

- log4net kann programmatisch oder per Konfigurationsdatei konfiguriert werden

```
<log4net>
    <appender name="A1" type="log4net.Appender.ConsoleAppender">

        <layout type="log4net.Layout.PatternLayout">
            <conversionPattern value=
"%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />
        </layout>
    </appender>

    <!-- Set root logger level to DEBUG and its only appender to A1 -->
    <root>
        <level value="DEBUG" />
        <appender-ref ref="A1" />
    </root>
</log4net>
```

LOOMBIRD

## Logging Bsp.

```
using log4net;
using log4net.Config;

public class MyApp
{
    private static readonly ILog log = LogManager.GetLogger
                                                (typeof(MyApp));

    static void Main(string[] args)
    {
        XmlConfigurator.Configure(new System.IO.FileInfo(args[0]));

        log.Info("Entering application.");
        Bar bar = new Bar();
        bar.DoIt();
        log.Info("Exiting application.");
    }
}
```

LOOMBIRD

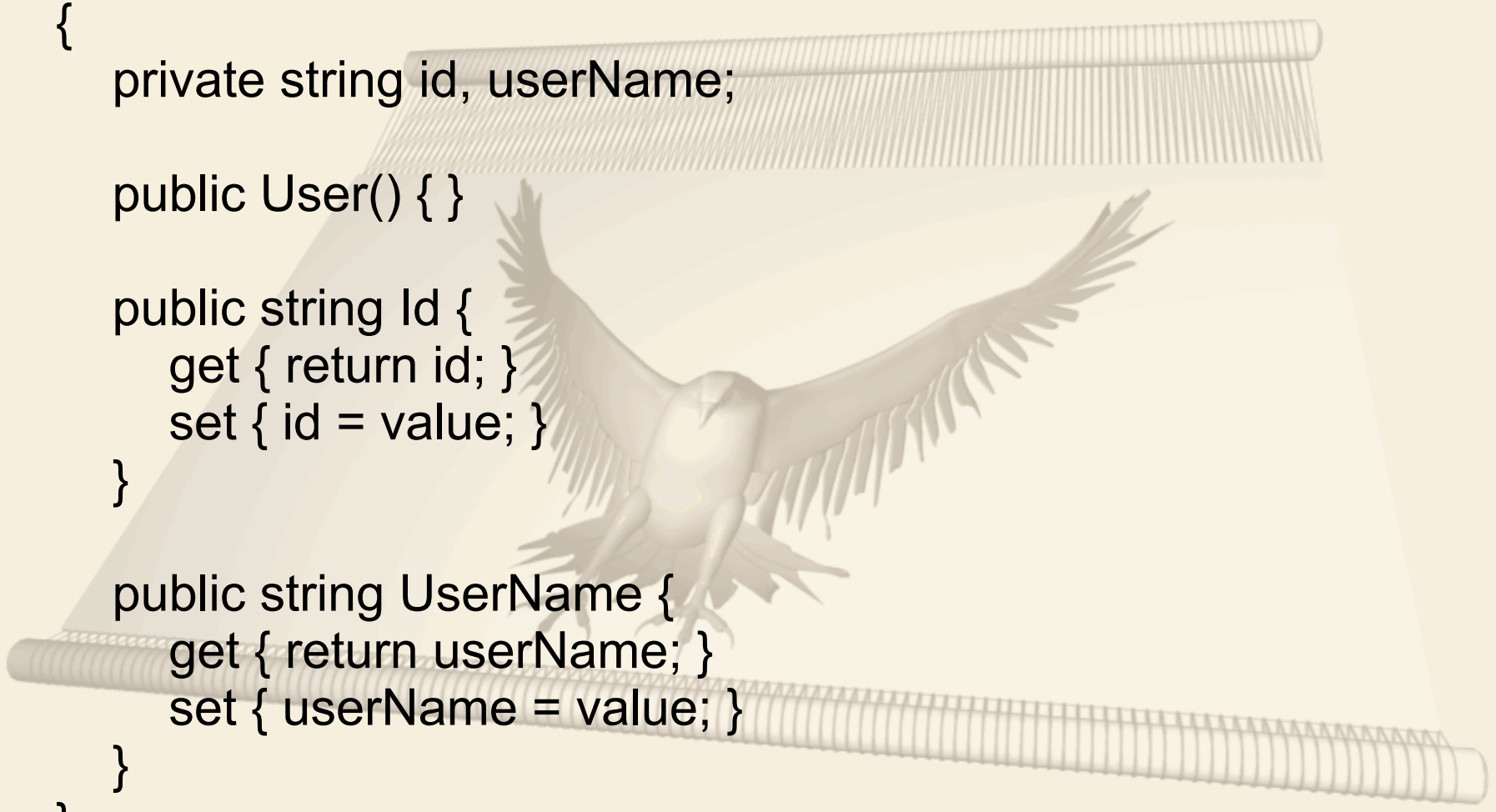
## NHibernate

- Objektpersistenz Bibliothek für relationale Datenbanken
- NHibernate legt Objekte in Datenbanken ab und holt sie wieder heraus
  - Selbst wird kein SQL dazu geschrieben
- Mapping von Objekt und Datenbanktabellen wird konfiguriert
- NHibernate ist verantwortlich, dass die Objekte persistent gehalten werden
- Erspart viel Fleißarbeit

LOOMBIRD

## Beispiel

```
namespace Nhibernate.Examples.QuickStart {
    public class User
    {
        private string id, userName;

        public User() { }

        public string Id {
            get { return id; }
            set { id = value; }
        }

        public string UserName {
            get { return userName; }
            set { userName = value; }
        }
    }
}
```



LOOMBIRD

## Beispiel Mapping

- Mappen der Klasse NHibernate.Examples.QuickStart.User aus dem Assembly Nhibernate.Examples in die Datenbank
- Dazu im simpelsten Fall Datei User.hbm.xml (allg. Klassenname.hbm.xml) in dasselbe Verzeichnis wie die Klasse

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
    <class name="NHibernate.Examples.QuickStart.User,
                        NHibernate.Examples" table="users">
        <id name="Id" column="LogonId" type="String" length="20">
            <generator class="assigned" />
        </id>
        <property name="UserName" column="Name"
                        type="String" length="40"/>
    </class>
</hibernate-mapping>
```

LOOMBIRD

# Datenbankkonfiguration

- Konfiguration der DB in Application-Config von .NET

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section
      name="nhibernate"
      type="System.Configuration.NameValueSectionHandler,
            System, Version=1.0.5000.0, Culture=neutral,
            PublicKeyToken=b77a5c561934e089"
    />
  </configSections>

  <nhibernate>
    <add key="hibernate.connection.provider"
      value="NHibernate.Connection.DriverConnectionProvider"
    />
    <add key="hibernate.dialect"
      value="NHibernate.Dialect.MsSql2000Dialect"
    />
    <add key="hibernate.connection.driver_class"
      value="NHibernate.Driver.SqlClientDriver"
    />
    <add key="hibernate.connection.connection_string"
      value="Server=localhost;initial catalog=nhibernate;Integrated Security=SSPI"
    />
  </nhibernate>
</configuration>
```

LOOMBIRD

## Codebeispiel

```
Configuration cfg = new Configuration();
// läd die Mapping Informationen
cfg.AddAssembly("NHibernate.Examples");

ISessionFactory factory = cfg.BuildSessionFactory();
ISession session = factory.OpenSession();
ITransaction transaction = session.BeginTransaction();

User newUser = new User();
newUser.Id = "joe_cool";
newUser.UserName = "Joseph Cool";

session.Save(newUser);

transaction.Commit();
session.Close();
```

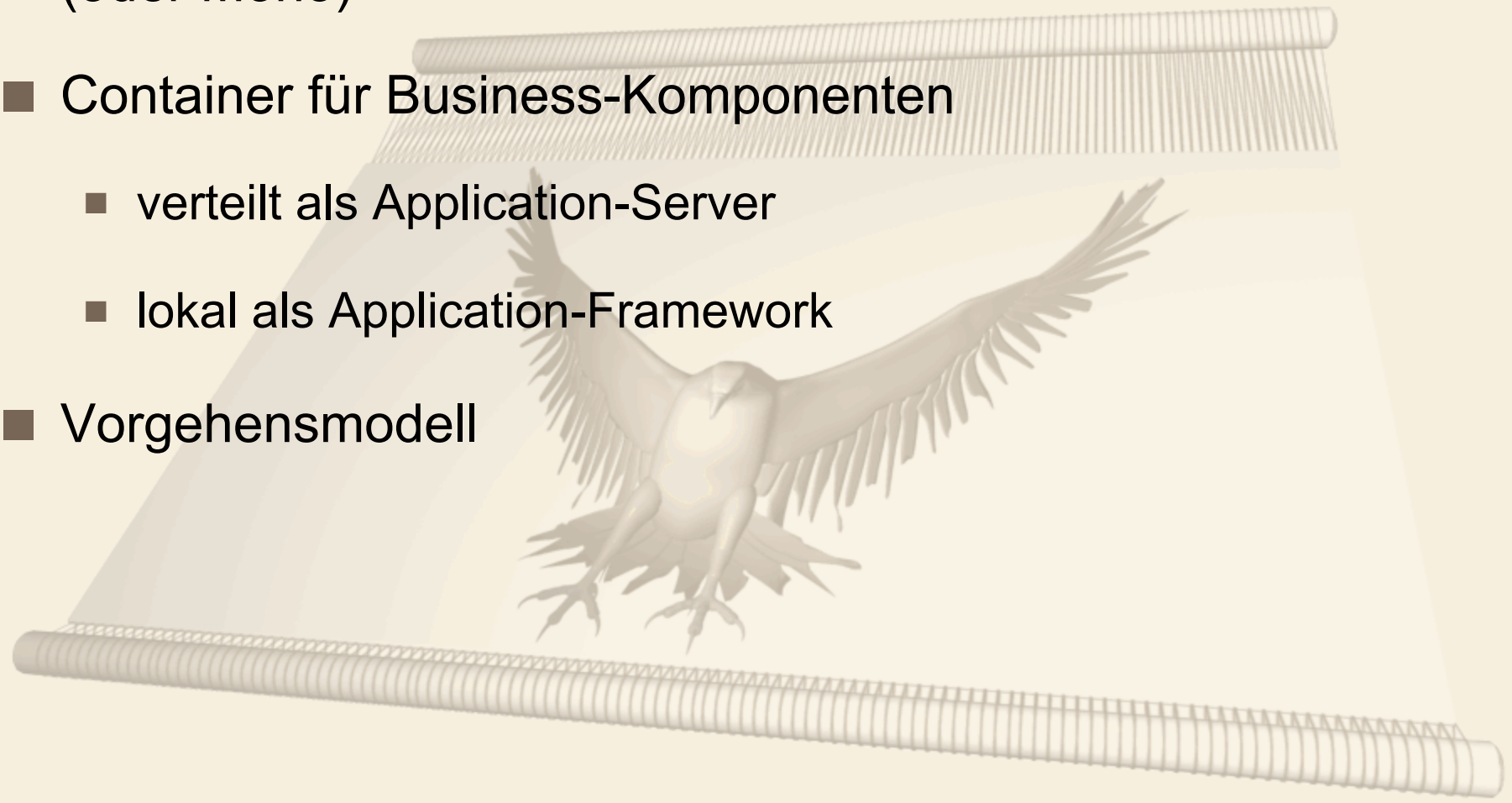
LOOMBIRD

# Loombird

LOOMBIRD

## Loombird

- Komponenten-Technologie für .NET (oder Mono)
- Container für Business-Komponenten
  - verteilt als Application-Server
  - lokal als Application-Framework
- Vorgehensmodell



LOOMBIRD

# Warum App Server oder Business Container?

- Was ist eine Business Komponente?
  - Eine Sammlung von Business-Funktionen und -Services
  - Etwas, das sich für Technik nicht interessiert
  - Eine geeignete Umgebung braucht
    - Um die technischen Details zu lösen
- Was ist ein Business Container?
  - Die Umgebung für eine Business Komponente
  - Verantwortung für
    - Lebenszyklus, Threading, Pooling, Resourcenmanagement, Persistenz

LOOMBIRD

# Warum App Server oder Business Container?

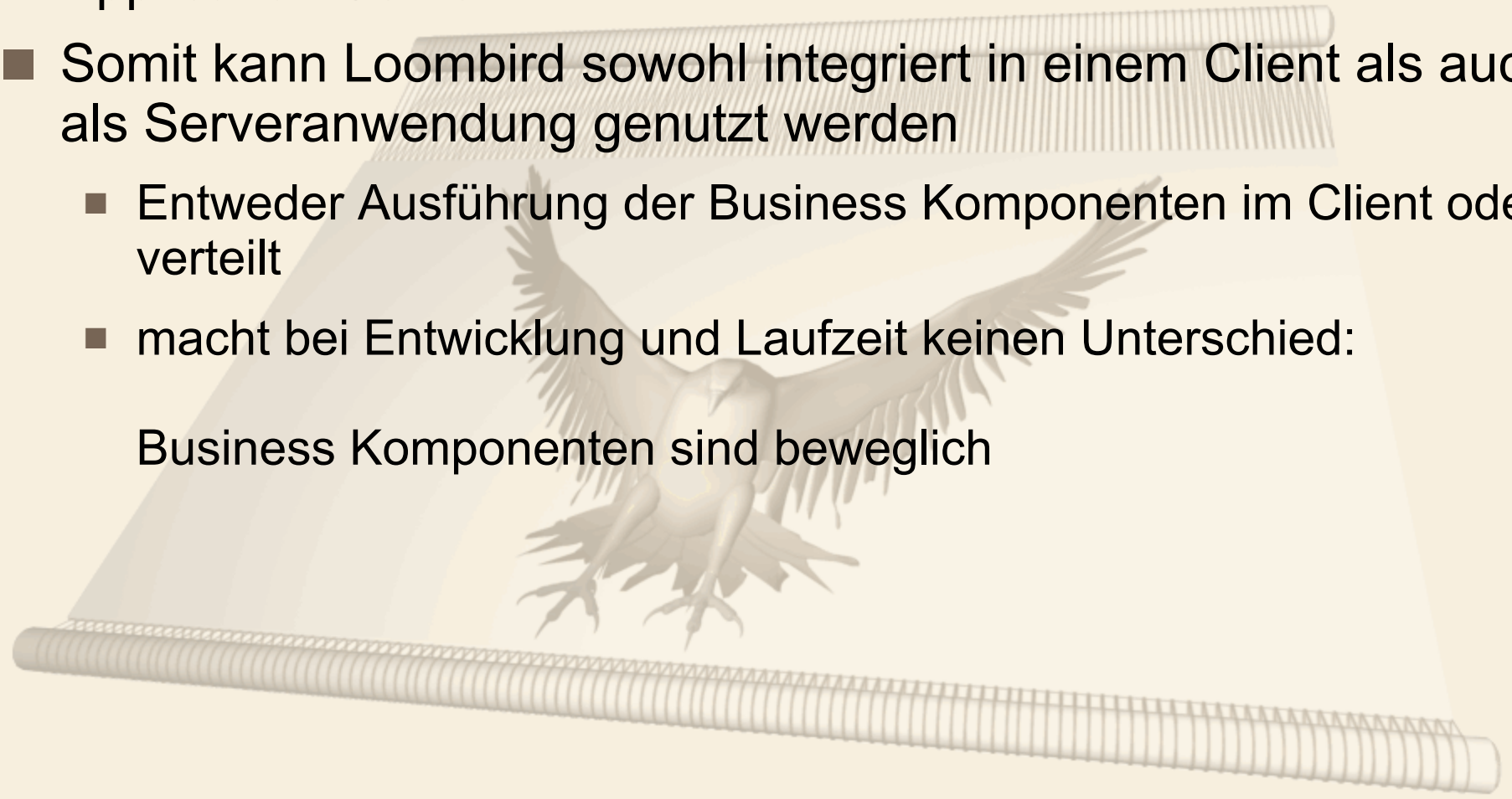
- Was ist ein Application Server?
  - Eine Serveranwendung in der ein Business Container läuft
  - Unterstützt Aufrufe von Business Komponenten übers Netzwerk
- Warum eines davon nutzen?
  - Zentralisierbarkeit bei Application Servern
  - Trennung von Verantwortlichkeiten
  - Zielorientierte Strukturierung
    - Service
    - Subsystem
    - Deployment
  - Trennung von Business und Technologie

LOOMBIRD

# Was ist der Loombird Business Container?

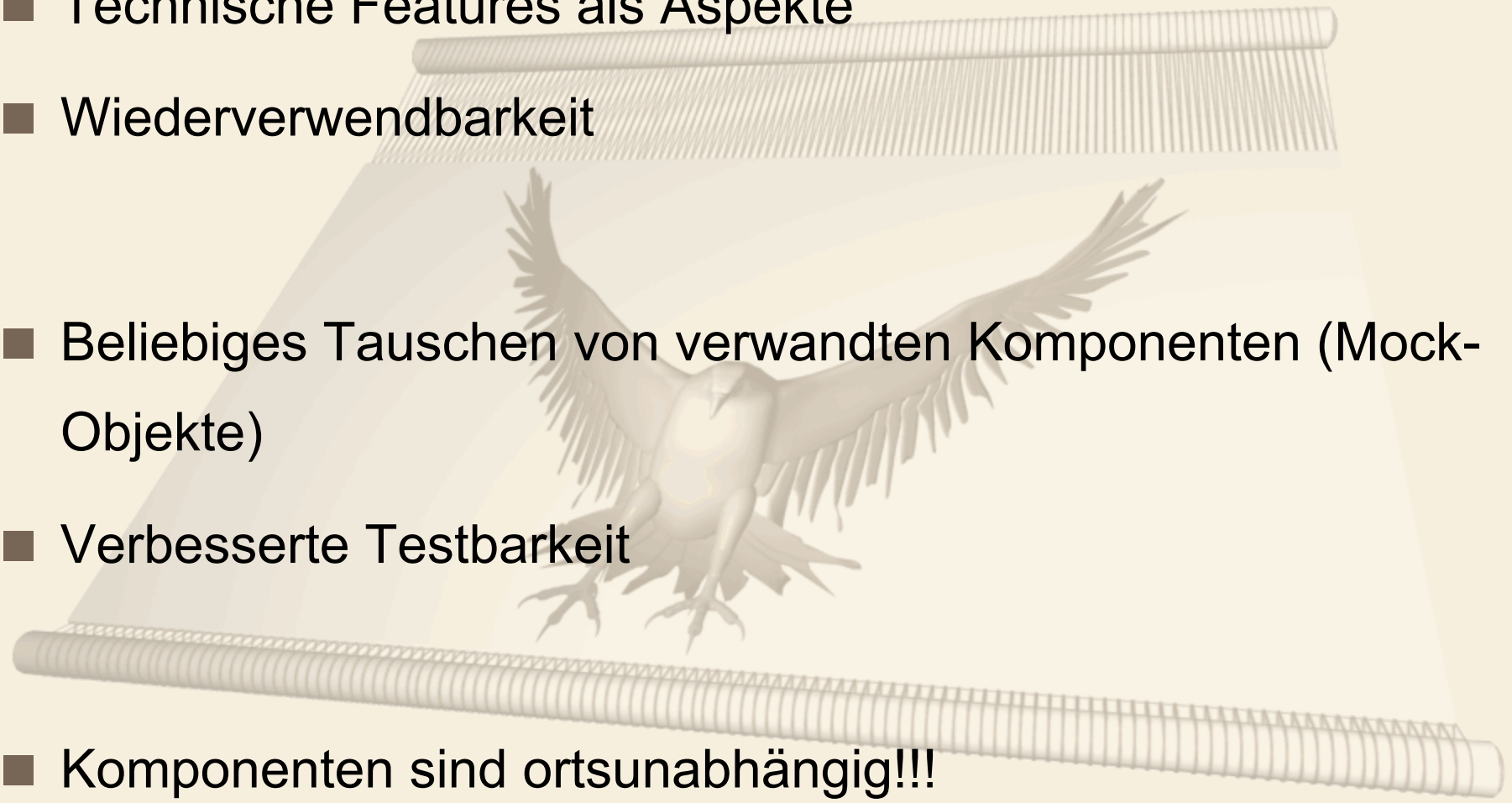
- Loombird ist sowohl reiner Business Container als auch Application Server
- Somit kann Loombird sowohl integriert in einem Client als auch als Serveranwendung genutzt werden
  - Entweder Ausführung der Business Komponenten im Client oder verteilt
  - macht bei Entwicklung und Laufzeit keinen Unterschied:

    Business Komponenten sind beweglich



LOOMBIRD

## Loombird – Vorteile



- Trennung von Business und Technologie
- Technische Features als Aspekte
- Wiederverwendbarkeit
- Beliebiges Tauschen von verwandten Komponenten (Mock-Objekte)
- Verbesserte Testbarkeit
- Komponenten sind ortsunabhängig!!!

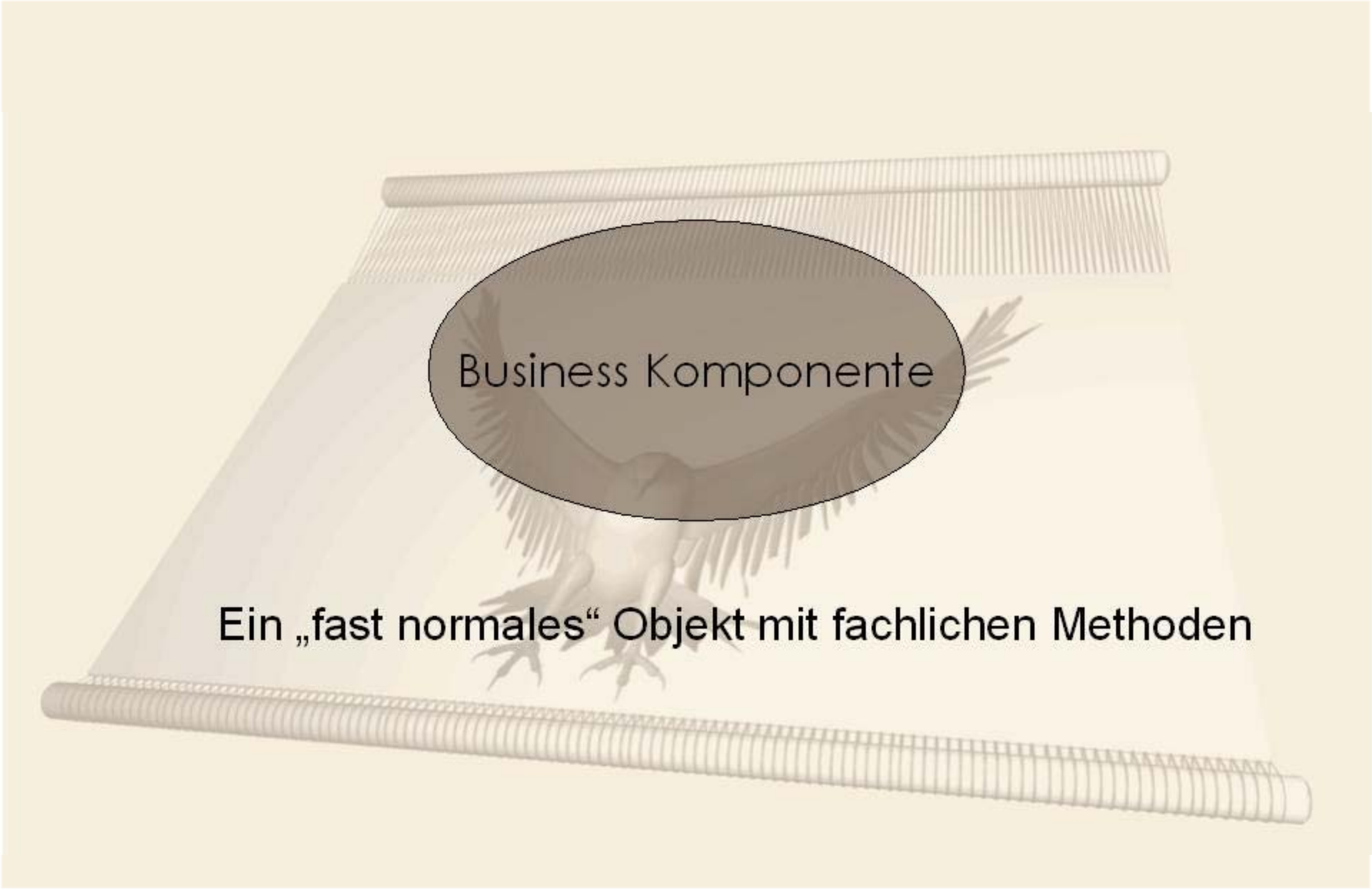
LOOMBIRD

## Loombird – Spezialitäten

- Interceptoren (Aspekte)
- Wohldefinierte Unterschiede zwischen
  - standardisierten Schnittstellen (Context)
  - benutzerdefinierten Erweiterungen (Property)
- Erhöhte Haltbarkeit von Komponenten
  - Implizite Parameter
  - Komponenten bleiben trotz erweiterten Anforderungen unverändert nutzbar

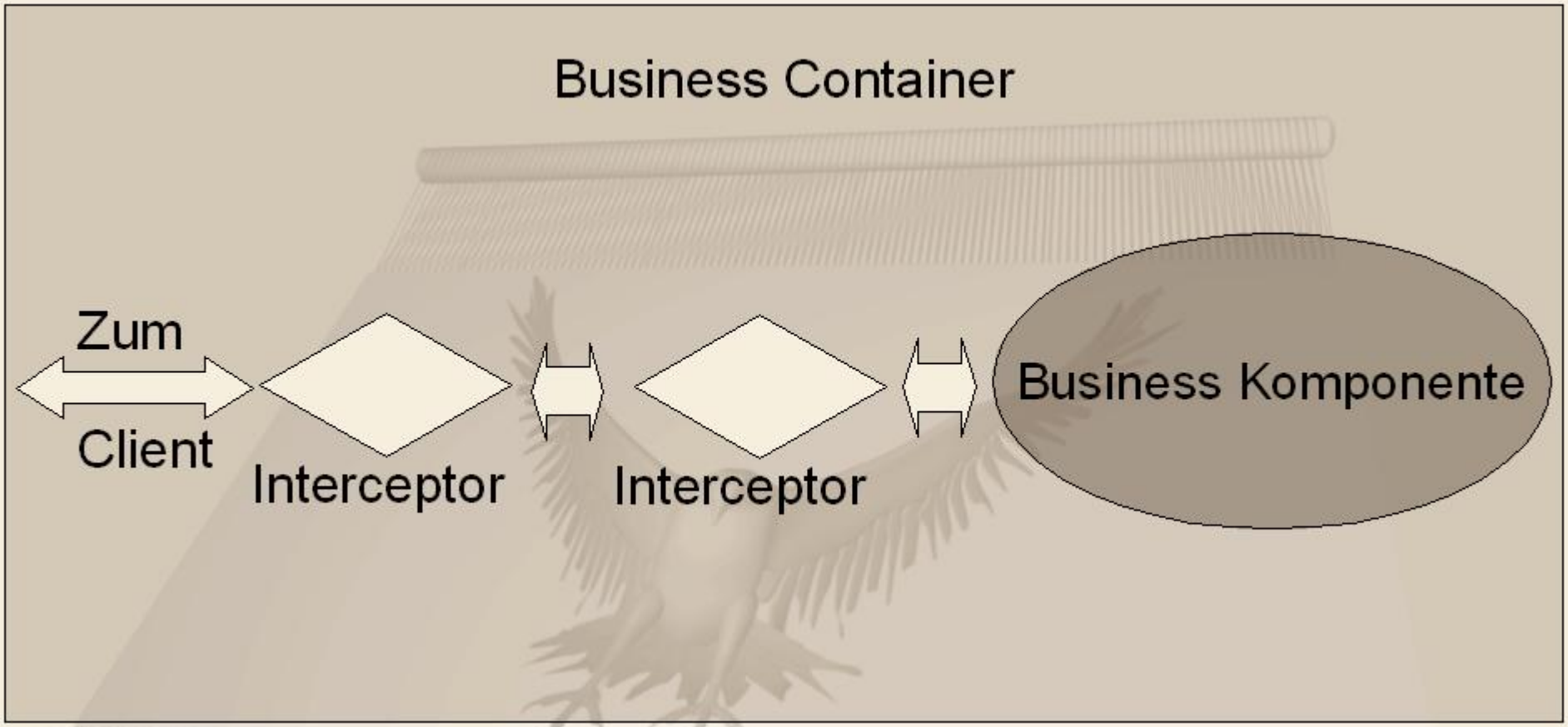
LOOMBIRD

# Business Komponente



LOOMBIRD

# Interceptoren

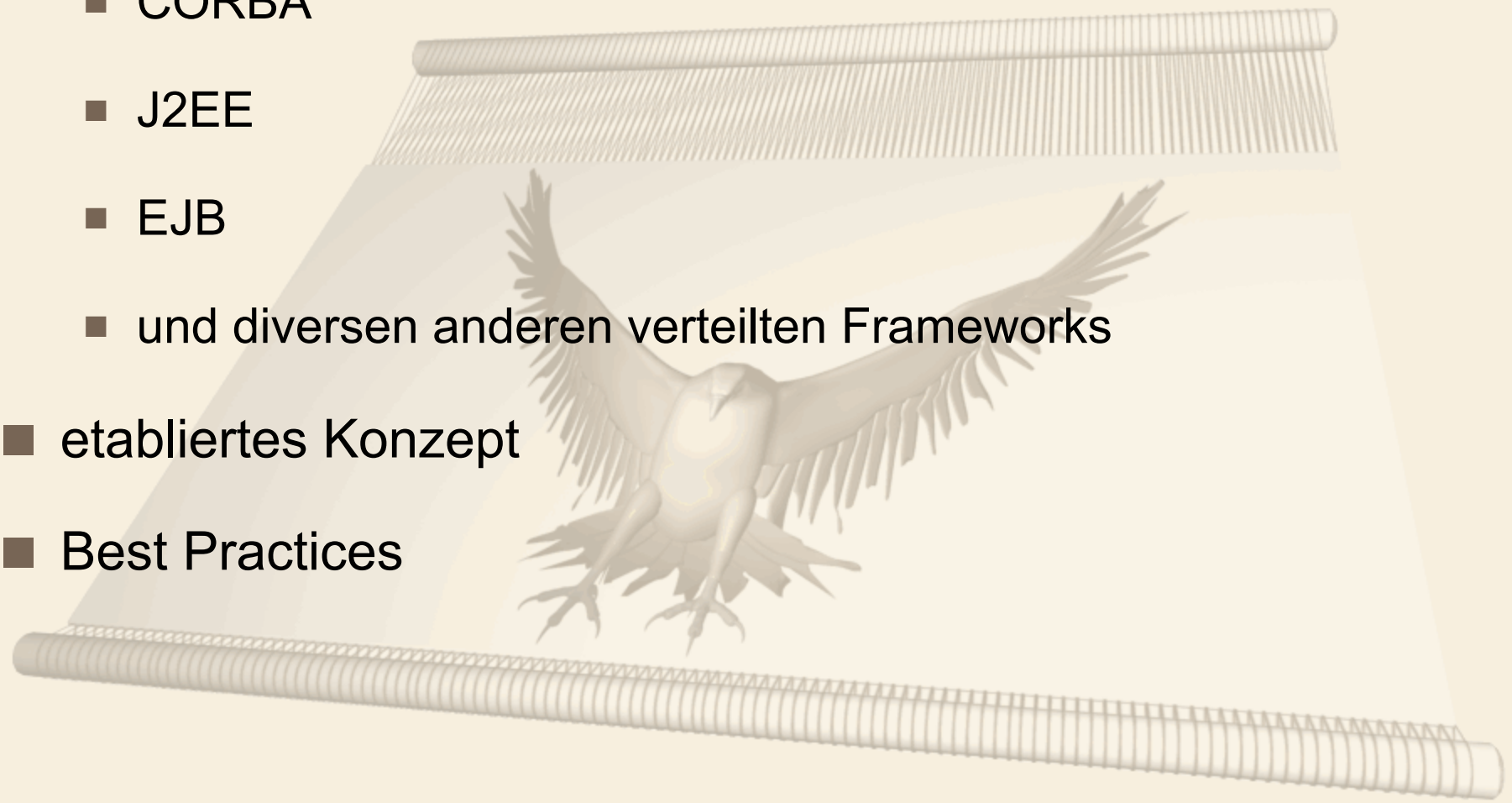



Die Business Komponente kann mit beliebigen Interceptoren um Funktionalität erweitert werden:
z.B. Security und Transaktionen

LOOMBIRD

# Warum Loombird (1/3)

- Ergebnis der Arbeit mit
  - CORBA
  - J2EE
  - EJB
  - und diversen anderen verteilten Frameworks
- etabliertes Konzept
- Best Practices



LOOMBIRD

# Warum Loombird (2/3)

- Bestechende Einfachheit
- Komplexität wird – wie gewohnt – über Datenbanken abgewickelt
  - Transaktionen
  - Clustering
  - Skalierung
  - Ausfallsicherheit
  - Datensicherung

LOOMBIRD

## Warum Loombird (2/3)

- Zeitersparnis
  - bei der Entwicklung
  - beim Deployment
  - beim Testen
- Lokationstransparenz
  - Mixed-Mode
    (ohne Änderung als Client- oder Server-Applikation verfügbar)
- Einheitliches Vorgehensmodell
- Volle Prozessintegration

LOOMBIRD

# Vielen Dank!

oliver.szymanski@loombird.com

michael.wiedeking@loombird.com

www.loombird.com