Knockout – MVVM für Javascript und HTML5

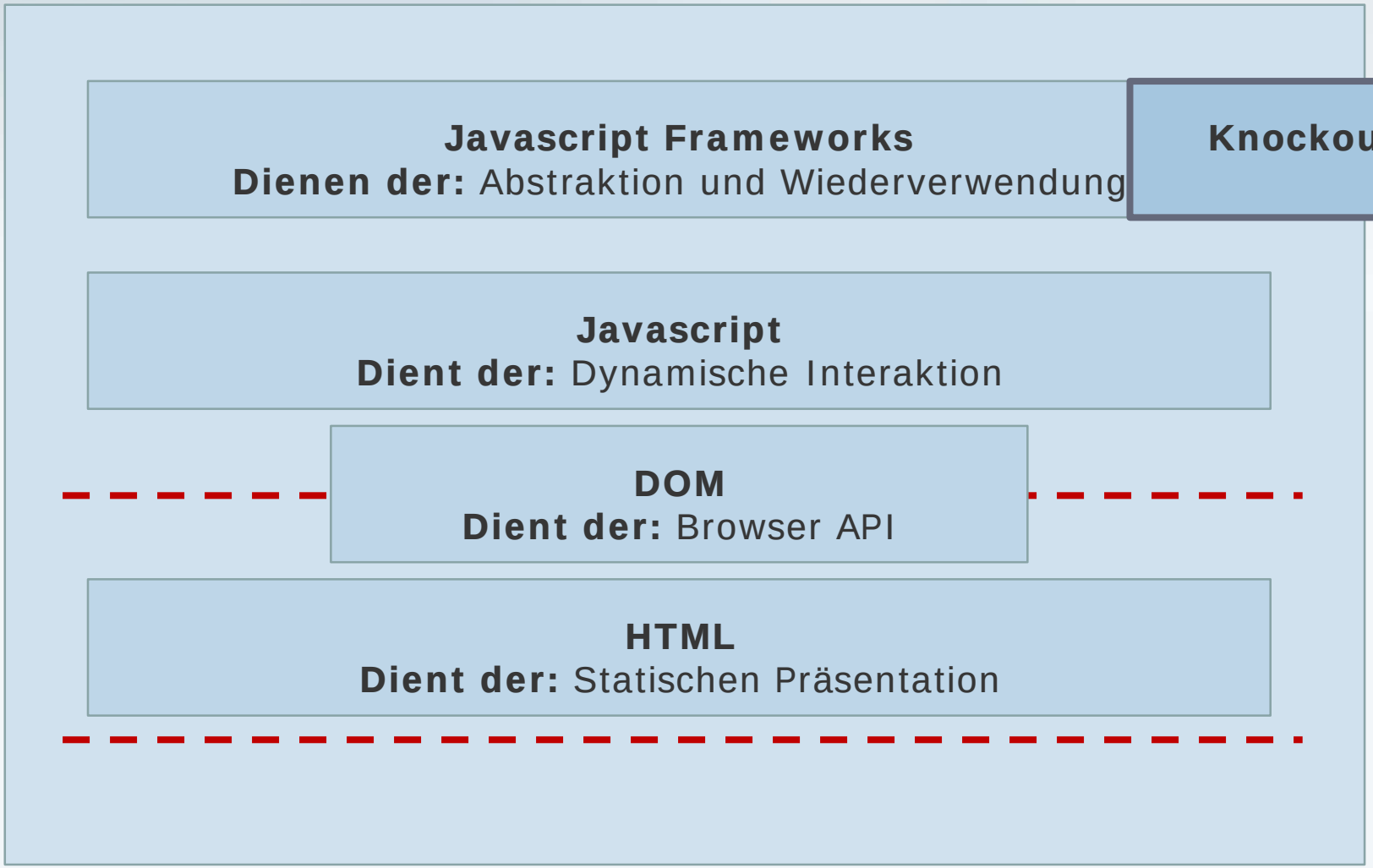
W3L AG
info@W3L.de

2012

Inhalt

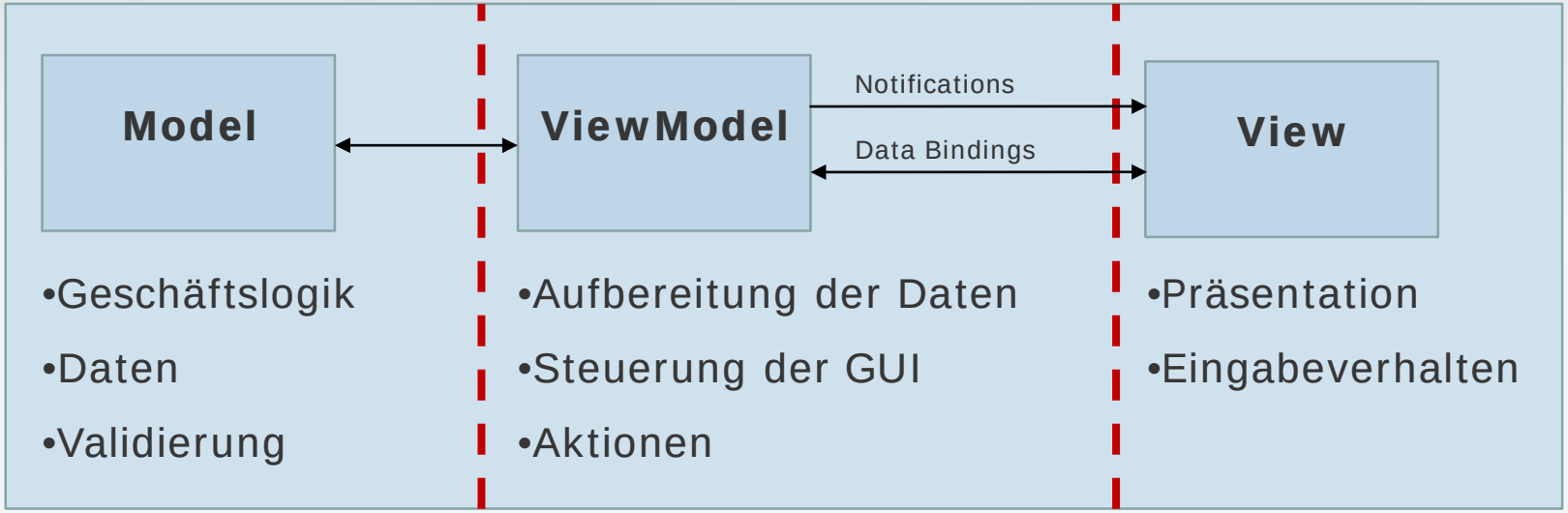
zurück

Begriffliche Einordnung



Was ist Knockout?

- **Umsetzung des MVVM-Patterns in Javascript**



- **Für was ist das gut?**
  - Keine kaskadierten Javascript-Ereignisbehandlungsfunktionen
  - Keine DOM-basierte Aktualisierung der GUI
  - Implementierung komplexer, interaktiver Web-Anwendungen wird spürbar einfacher!

Was ist Knockout?

- **Open Source nach der MIT-Lizenz**
- **Vollständig in Javascript entwickelt**
- **Funktioniert im Beisein anderer Javascript-Frameworks**
  - jQuery
  - Prototype
- **Besitzt keine Abhängigkeiten zu anderen Frameworks**
  - Klein und leichtgewichtig
  - 40kb
- **Unterstützt alle gängigen Browser**
  - IE 6+
  - Firefox 2+
  - Chrome, Opera, Safari

Hintergrund

- **Entwickler**
  - Steve Sanderson
    - Program Manager bei Microsoft
    - ASP.NET MVC
- **Community Projekt**
  - Microsoft steckt offiziell nicht dahinter

Motivierendes Beispiel

jQuery vs. Knockout

- **jQuery abstrahiert vom DOM API**
  - Inkonsistenzen zwischen Browsern werden eliminiert
  - Vereinfacht die Programmierung

- **jQuery besitzt kein Konzept für zugrundeliegendes Datenmodell**
  - Verknüpfung der Daten und der Präsentation muss manuell erfolgen
  - Beispiel
    - Liste

Grundlagen

- **Konzepte**
  - Deklaratives Binding
    - Verknüpfung von DOM-Elementen mit ViewModel-Daten
  - Automatische UI-Aktualisierung
    - Wenn sich das ViewModel ändert, wird die Oberfläche automatisch aktualisiert
  - Dependency-Tracking
    - Abgeleitete Eigenschaften werden automatisch neu berechnet, sobald sich Eigenschaften ändern, auf die sie sich beziehen.
  - Templating
    - Vorlagenbasierte Listendarstellung über Assoziationen
  - Einfache Erweiterbarkeit

Grundlagen – Definition des ViewModels

- **Wiederholung**
  - Einfaches ViewModel auf Basis von Javascript-Eigenschaften

```
function ViewModel() {
    var that = this;
    that.Vorname = „Max";
    that.Name = „Mustermann";
}
```

  - ViewModel auf Basis von Observables

```
function ViewModel() {
    var that = this;
    that.Vorname = ko.observable("");
    that.Name = ko.observable("");
    that.FullName = ko.computed(function () {
        return that.Vorname() + " " + that.Name();
    });
    that.Reset = function () {
        that.Vorname(„Max");
        that.Name(„Mustermann");
    };
    that.Reset();
}
```

Grundlagen – Definition des ViewModels

- **Observables**
  - Javascript Eigenschaften haben kein Benachrichtigungskonzept bei Änderungen
    - Stichwort: Observer Pattern
  - Observables bringen dieses Konzept in die Javascript-Welt.
  - Observables bilden die **Grundlage** der automatischen UI-Aktualisierung und des Dependency-Trackings.

Grundlagen – Definition des ViewModels

- **Erzeugung von neuen Observable-Instanzen**

```
that.Vorname = ko.observable("");
```

- **Lesen und Schreiben in Observables**
  - Lesen

```
vm.Vorname()
```

  - Schreiben

```
vm.Vorname(„Max")
```

  - Folgende Schreibweise ist falsch!

```
alert(vm.Vorname)
vm.Vorname = „Max";
```

- **Explizite Registrierung von Ereignisabhörern**

```
that.Name.subscribe(function (newValue) {
   alert(newValue);
});
```

Grundlagen – Definition des ViewModels

- **Berechnete bzw. abgeleitete Eigenschaften**
  - Können über eine Javascript-Funktion definiert werden.
  - Beispiel

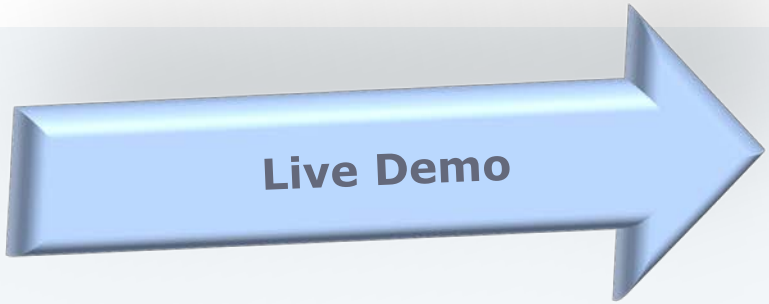
```
that.FullName = ko.computed(function () {
    return that.Vorname() + " " + that.Name();
});
```

  - Dependency Tracking
    - Bei der Definition wird die abgeleitete Eigenschaft erstmal berechnet.
    - Knockout protokolliert den Zugriff auf andere Observables und erstellt einen Änderungspfad für zukünftige Aktualisierungen.

Grundlagen – Definition des ViewModels

Beschreibare abgeleitete Eigenschaften

```
function ViewModel() {
     var that = this;
     that.Vorname = ko.observable("");
     that.Name = ko.observable("");
     that.FullName = ko.computed({
         read: function () {
             return that.Vorname() + " " + that.Name();
         },
         write: function (value) {
             var trim = value.lastIndexOf(" ");
             if (trim > 0) {
                 that.Vorname(value.substring(0, trim));
                 that.Name(value.substring(trim + 1));
             } else {
                 that.Vorname(value);
             }
         }
     });
 }
```



Grundlagen – Deklaratives Binding

- **Wiederholung**
  - Value-Binding

```
<input data-bind="value: Vorname"/>
```

  - Text-Binding

```
<span data-bind="text: FullName"></span>
```

  - Foreach-Binding

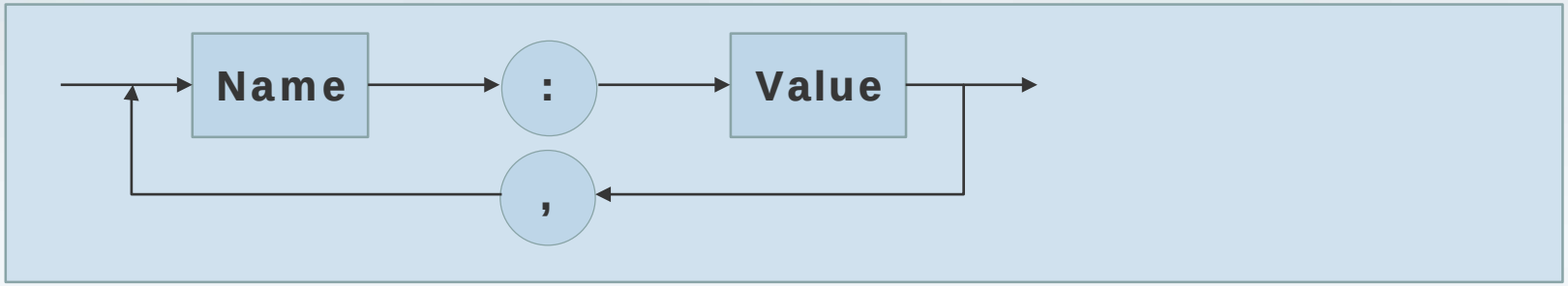
```
<table>
  <thead>
     <tr><th>Vorname</th><th>Name</th><th></th></tr>
  </thead>
  <tbody data-bind="foreach: Freunde">
     <tr>
       <td data-bind="text: Vorname"></td>
       <td data-bind="text: Name"></td>
       <td><a href="#" data-bind="click: $parent.removeFreund">Remove</a></td>
     </tr>
  </tbody>
</table>
```

Grundlagen – Deklaratives Binding

- **Klassen von Bindings**
  - Text und visuelle Erscheinung
    - `visible, text, html, css, style, attr`
  - Kontrollfluss
    - `foreach, if, ifnot, with`
  - Formularfelder
    - `click, event, submit, enable, disable, value, hasfocus, checked, options, selectedOptions, uniqueName`
  - Templates
    - `template`

Grundlagen – Deklaratives Binding

Binding-Syntax



Value-Syntax

- Als Value-Definition kann beinahe jede beliebige Javascript-Expression verwendet werden.

Grundlagen – Deklaratives Binding

- **Binding-Syntax**
  - Beispiele
    - Einfache Eigenschaft

```
<input data-bind="value: Vorname"/>
```

    - Expressions

```
<span data-bind="text: Freunde().length > 15 ? 'Viele' : 'Wenige' "/>
<span data-bind="click: function(data) {alert('Hallo Welt')} "/>
```

Grundlagen – Deklaratives Binding

- **Binding Kontext**
  - **`$parent`**
    - Bei einem foreach-Binding kann auf über diese Variable auf den umgebenden Kontext zugegriffen werden.
    - Speziellere Variante: `$parents[n]`, wobei `$parents[0] == $parent`
  - **`$root`**
    - Zeigt auf das Wurzel ViewModel, welches über `applyBindings()` an Knockout übergeben wurde.
  - **`$data`**
    - Bei einem foreach-Binding handelt es sich um das ViewModel des dargestellten Listenelements.
  - **`$index`**
    - Bei einem foreach-Binding handelt es sich um den nullbasierten Index.
  - **`$element`**
    - Aktuelle DOM-Element
    - Beispiel

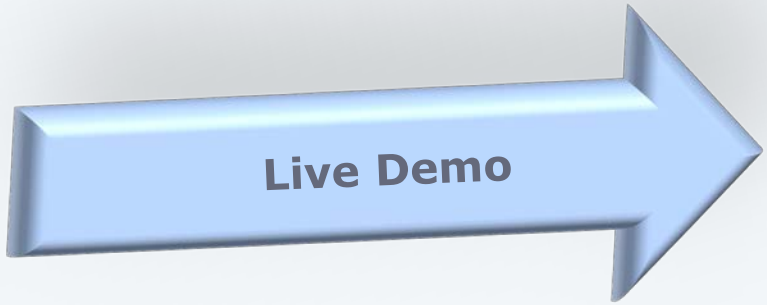
```
<span id="item1" data-bind="text: $element.id"></span>
```

Grundlagen – Deklaratives Binding

- **Text und visuelle Erscheinung**
  - `visible`-Binding
    - Erlaubt das Ein- und Ausblenden von DOM-Elementen
  - `text`-Binding
    - Setzt den Text des zugehörigen DOM-Elements
  - `html`-Binding
    - Kontrolliert das InnerHTML des zugehörigen DOM-Elements
  - `css`-Binding
    - Kontrolliert die CSS-Klassen des zugehörigen DOM-Elements
  - `style`-Binding
    - Kontrolliert und steuert die Style-Eigenschaft des zugehörigen DOM-Elements
  - `attr`-Binding
    - Kontrolliert und steuert Attribute de

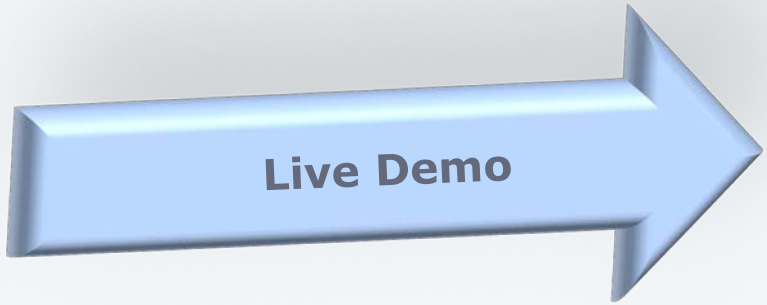
Grundlagen – Deklaratives Binding

- **Kontrollfluss**
  - **`foreach`**-Binding
    - Dupliziert einen Abschnitt in Abhängigkeit zum ViewModel. Sinnvoll bei der Darstellung von Observable Arrays.
    - Innerhalb des Abschnitts kann auf das Element des Observable Arrays durch den Binding-Kontext **`$data`** zugegriffen werden.
  - **`if`**-Binding
    - Verhält sich ähnlich wie `visible`-Binding.
    - Beim `if`-Binding wird das DOM-Element jedoch vollständig entfernt.
  - **`ifnot`**-Binding



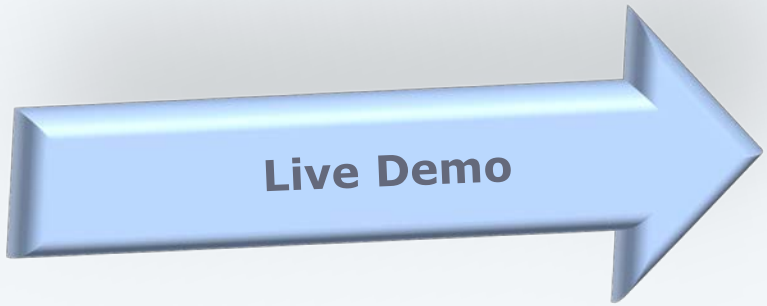
Grundlagen – Deklaratives Binding

- **Formular**
  - **`click`**-Binding
    - Ereignisabhörer für das Mouseclick-Ereignis
  - **`enable`**-Binding
  - **`options`**-Binding
    - Steuert die dargestellten Element bei einer Auswahl-Box (`<select></select>`).
  - **`selectedOptions`**-Binding
    - Eigenschaft, in der die Auswahl gespeichert werden soll.



Grundlagen – Deklaratives Binding

- **Rendering Template**
  - `template`-Binding
    - Ähnlich wie das `foreach`-Binding, jedoch ist die Definition des Abschnitts ausgelagert.
    - Dadurch bessere Wiederverwendbarkeit



Grundlagen – Custom Binding

- **Knockout besitzt ein sehr flexibles Erweiterungskonzept**
- **Entwickler können eigene Bindings entwickeln**
  - Über Bindings wird das ViewModel auf ein oder mehrere DOM-Elemente abgebildet.
- **Registrierung eigener Bindings**

```
ko.bindingHandlers.yourBindingName = {
     init: function (element, valueAccessor, allBindingsAccessor, viewModel, bindingContext) {
         // Wird beim ersten Verbindungsaufbau aufgerufen.

     },
     update: function (element, valueAccessor, allBindingsAccessor, viewModel, bindingContext) {
         // Wird beim ersten Verbindungsaufbau sowie bei jeder Änderung des ViewModels
         // aufgerufen. Die Aufgabe ist, das DOM-Element gemäß ViewModel zu ändern!
    }
};
```

  - Parameter
    - element: DOM-Element
    - valueAccessor: Eigenschaftsobjekt des Cutom-Bindings
    - allBindingsAccessor: Eigenschaftsobjekt aller an diesem DOM-Element definierten Bindings
    - viewModel: Das an `applyBindings()` übergebene ViewModel.

Grundlagen – Custom Binding

Beispiel: Autovervollständigung

element

```
<input class="small" data-bind="value: $data.Parameter, autocomplete: {url: '<URL>'}"/>
```

valueAccessor

allBindingsAccessor

Implementierung

```
/* Autocomplete Custom-Binding */
ko.bindingHandlers.autocomplete = {
 init: function (element, valueAccessor, allBindingsAccessor, viewModel) {
    var autocomplete = valueAccessor();
    if (autocomplete.hasOwnProperty("url")) {
        jQuery(element).autocomplete({source: autocomplete.url,
                                      close: function (event, ui) {
                                         var value = allBindingsAccessor().value();
                                         var text = jQuery(element).val();
                                         if (text != value) {
                                            allBindingsAccessor().value(text);
                                         }
                                      });
   }
 }
};
```

Live Demo

Grundlagen – Unobtrusive Javascript

- **Definition**
  - Unaufdringliches Javascript
  - Keine Mischung zwischen Javascript und HTML-Präsentation

- **Vorteile bei Verwendung von Custom Bindings**
  - Javascript-Programme lassen sich bequem auslagern
  - Javascript-Frameworks lassen sich mit Knockout verknüpfen
    - Eindeutige Schnittstelle zwischen ViewModel und DOM-Element
    - Komplexitätsreduktion
  - Bindeglied zwischen Web-Designern und Web-Entwicklern
    - Beide Welten können sich auf Ihre Arbeit konzentrieren

Fazit

- **Zusammenfassung**
  - Knockout besitzt höheren Abstraktionsgrad als jQuery.
    - Ohne Fachwissen über DOM API zu Web-Anwendungen
    - Gutes Bindeglied zwischen Web-Designern und Web-Entwicklern
  - Implementierung interaktiver Web-Applikationen mit zahlreichen abhängigen Daten wird enorm erleichtert.
  - Erweiterungsmechanismen erlauben die Anwendung von Unobstrusive Javascript.
    - Bessere Entkopplung zwischen HTML und Javascript
  - Geringe Laufzeitanforderungen
    - Sehr gutes Laufzeitverhalten
    - Keine bekannten Fehler
  - Schnell erlernbar und sehr gut dokumentiert.
  - Praxistauglich
    - **Sehr empfehlenswert!**

Vielen Dank!

Inhouse-Schulungen

Wir bieten Inhouse-Schulungen und Beratung durch unsere IT-Experten und –Berater.

Schulungsthemen

- Softwarearchitektur (OOD)
- Requirements Engineering (OOA)
- Nebenläufige & verteilte Programmierung

Gerne konzipieren wir auch eine individuelle Schulung zu Ihren Fragestellungen.

Sprechen Sie uns an!
Tel. 0231/61 804-0, info@W3L.de

W3L-Akademie

In Zusammenarbeit mit der Fachhochschule Dortmund bieten wir

zwei Online-Studiengänge

- B.Sc. Web- und Medieninformatik
- B.Sc. Wirtschaftsinformatik

und 7 Weiterbildungen im IT-Bereich an.

Besuchen Sie unsere Akademie!
http://Akademie.W3L.de