



Refaktorisierung

CODE SMELLS



Saxonia Systems
So geht Software.

Der Sprecher



Hendrik Lösch

Senior Consultant & Coach

Hendrik.Loesch@saxsys.de

@HerrLoesch

Just-About.Net



WPF-Anwendungen mit MVVM und Prism

Modulare Architekturen verstehen und umsetzen



Windows 8 Store Apps mit MVVM und Prism

XAML-Entwurfsmuster, Bootstrapping, Navigation, Messaging



Test Driven Development – Praxisworkshop

Business-Applikationen testgetrieben entwickeln



Inversion of Control und Dependency Injection

Prinzipien der modernen Software-Architektur ...



Test Driven Development mit C#

Grundlagen, Frameworks, best Practices



Automatisiertes Testen mit Visual Studio 2012

Grundlagen, Testarten und Strategien



Saxonia Systems
So geht Software.

Refactoring CODE SMELLS

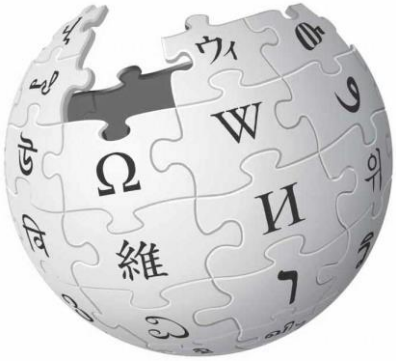


Saxonia Systems
So geht Software.

Schlechtes Design

1. It is hard to change because every change affects too many other parts of the system. (Rigidity - Starr)
2. When you make a change, unexpected parts of the system break. (Fragility - Zerbrechlich)
3. It is hard to reuse in another application because it cannot be disentangled from the current application. (Immobility - Unbeweglich)

Code Smells

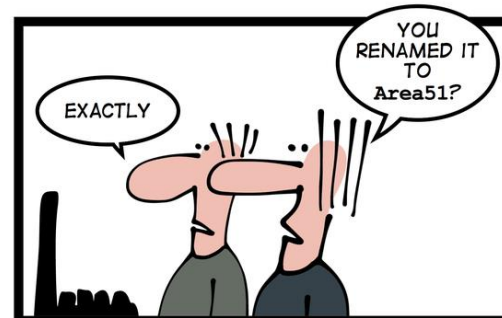
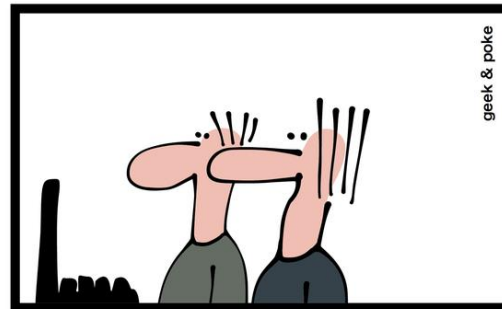


Unter **Code-Smell**, kurz **Smell** (deutsch ‚[schlechter] Geruch‘) oder deutsch **übelriechender Code** versteht man in der Programmierung ein Konstrukt, das eine Überarbeitung des Programm-Quelltextes nahelegt. Dem Vernehmen nach stammt die Metapher *Smell* von Kent Beck und erlangte weite Verbreitung durch das Buch *Refactoring* von Martin Fowler. Unter dem Begriff sollten handfestere Kriterien für Refactoring beschrieben werden, als das durch den vagen Hinweis auf Programmästhetik geschehen würde.

Bei Code-Smell geht es nicht um Programmfehler, sondern um funktionierenden Programmcode, der aber schlecht strukturiert ist. Das größte Problem liegt darin, dass der Code für den Programmierer schwer verständlich ist, so dass sich bei Korrekturen und Erweiterungen häufig wieder neue Fehler einschleichen. Code-Smell kann auch auf ein tieferes Problem hinweisen, das in der schlechten Struktur verborgen liegt und erst durch eine Überarbeitung erkannt wird.

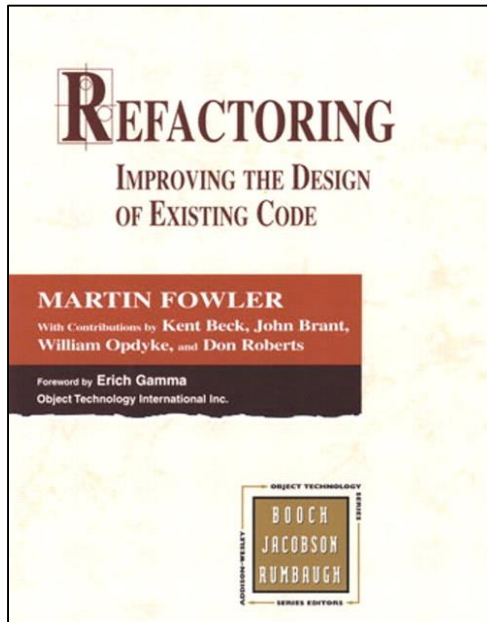
Code Smells

REFACTORING IS KEY



Saxonia Systems
So geht Software.

Refactoring Arten



refactoring.com

Add Parameter
Change Bidirectional Association to Unidirectional
Change Reference to Value
Change Unidirectional Association to Bidirectional
Change Value to Reference
Collapse Hierarchy
Consolidate Conditional Expression
Consolidate Duplicate Conditional Fragments
Decompose Conditional
Duplicate Observed Data
Dynamic Method Definition
Eagerly Initialized Attribute
Encapsulate Collection
Encapsulate Downcast
Encapsulate Field
Extract Class
Extract Interface
Extract Method
Extract Module
Extract Subclass
Extract Superclass
Extract Surrounding Method
Extract Variable
Form Template Method
Hide Delegate
Hide Method
Inline Class
Inline Method
Inline Module
Inline Temp
Introduce Assertion

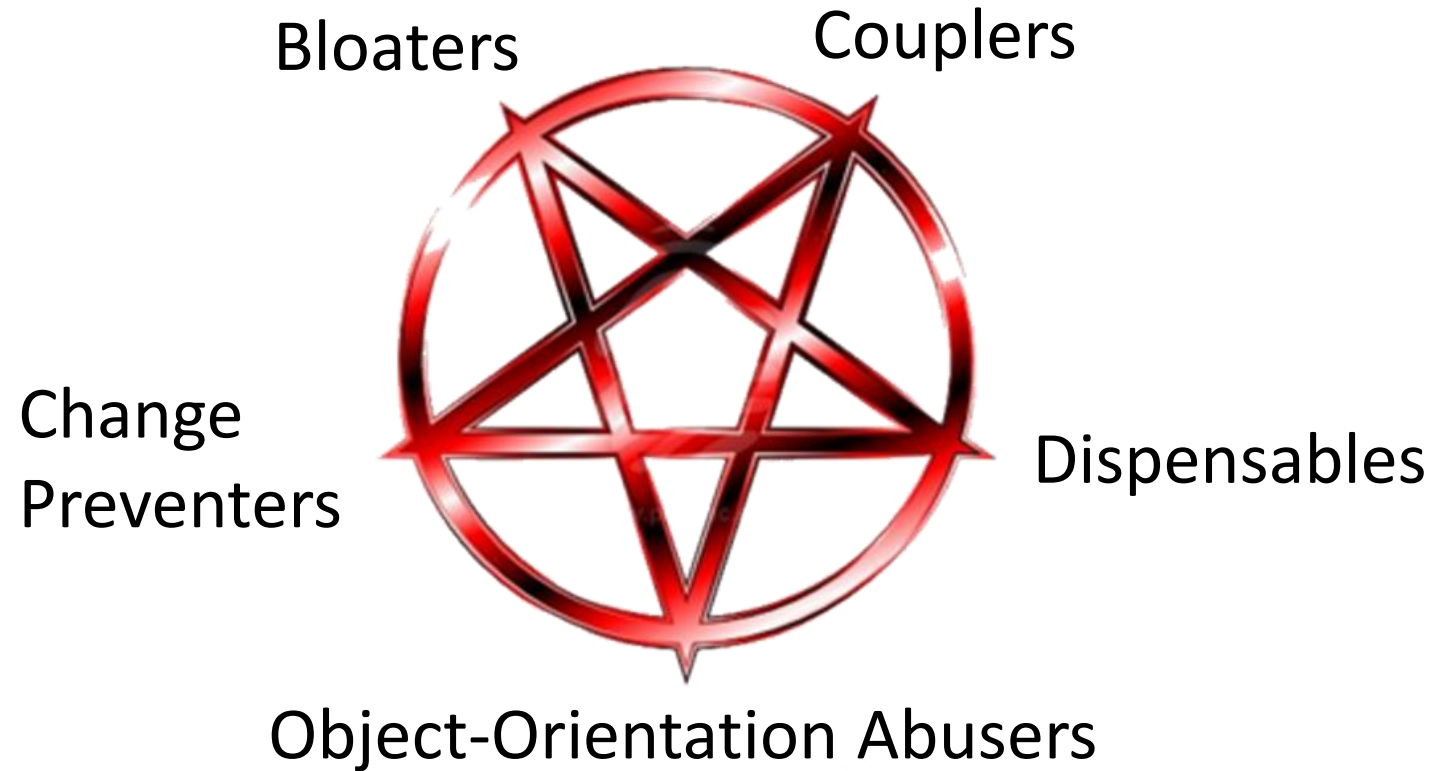
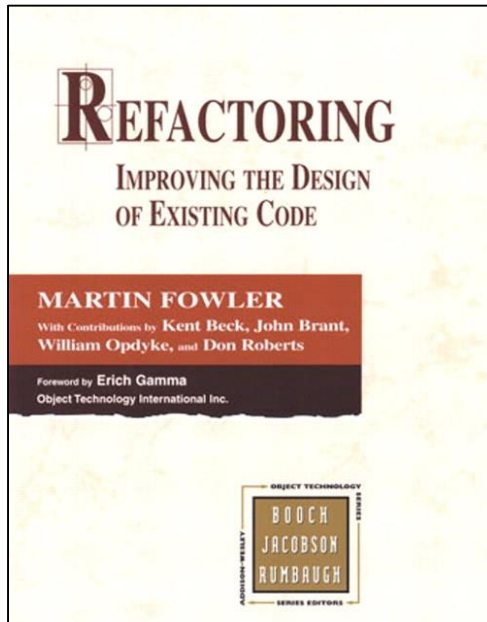
Introduce Class Annotation
Introduce Expression Builder
Introduce Foreign Method
Introduce Gateway
Introduce Local Extension
Introduce Named Parameter
Introduce Null Object
Introduce Parameter Object
Isolate Dynamic Receptor
Lazily Initialized Attribute
Move Eval from Runtime to Parse Time
Move Field
Move Method
Parameterize Method
Preserve Whole Object
Pull Up Constructor Body
Pull Up Field
Pull Up Method
Push Down Field
Push Down Method
Recompose Conditional
Remove Assignments to Parameters
Remove Control Flag
Remove Middle Man
Remove Named Parameter
Remove Parameter
Remove Setting Method
Remove Unused Default Parameter
Rename Method
Replace Abstract Superclass with Module
Replace Array with Object

Replace Conditional with Polymorphism
Replace Constructor with Factory Method
Replace Data Value with Object
Replace Delegation With Hierarchy
Replace Delegation with Inheritance
Replace Dynamic Receptor with Dynamic Method Definition
Replace Error Code with Exception
Replace Exception with Test
Replace Hash with Object
Replace Inheritance with Delegation
Replace Loop with Collection Closure Method
Replace Magic Number with Symbolic Constant
Replace Method with Method Object
Replace Nested Conditional with Guard Clauses
Replace Parameter with Explicit Methods
Replace Parameter with Method
Replace Record with Data Class
Replace Subclass with Fields
Replace Temp with Chain
Replace Temp with Query
Replace Type Code with Class
Replace Type Code with Module Extension
Replace Type Code With Polymorphism
Replace Type Code with State/Strategy
Replace Type Code with Subclasses
Self Encapsulate Field
Separate Query from Modifier
Split Temporary Variable
Substitute Algorithm



Saxonia Systems
So geht Software.

Code Smells



BLOATERS



Saxonia Systems
So geht Software.

Bloaters

ERLÄUTERUNG

Bloater sind üblicherweise Methoden oder Klassen die eine schwer zu überblickende Größe erreicht haben und meist unterschiedlichen Zwecken dienen. Die Größe wächst mit der Zeit an, da zu wenig refaktoriert wird und vorrangig bestehende Strukturen genutzt statt neue geschaffen werden.



Bloaters

ERLÄUTERUNG

Bloater sind üblicherweise Methoden oder Klassen die eine schwer zu überblickende Größe erreicht haben und meist unterschiedlichen Zwecken dienen. Die Größe wächst mit der Zeit an, da zu wenig refaktoriert wird und vorrangig bestehende Strukturen genutzt statt neue geschaffen werden.



Saxonia Systems
So geht Software.

LONG CLASS / METHOD

Klasse < 500 LoC
Methode < 50 LoC



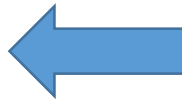
Saxonia Systems
So geht Software.

Bloaters

LONG CLASS / METHOD

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    if (!string.IsNullOrEmpty(this.id.Text))
    {
        UpdateExistingPublication();
    }
    else
    {
        CreateNewPublication();
    }

    ClearUI();
}
```



```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var publication = new Publication();

    if (!string.IsNullOrEmpty(this.id.Text))
    {
        var id = int.Parse(this.id.Text);
        Publication foundPublication = null;
        foreach (Publication item in this.publikationen.Items)
        {
            if (item.Id == id)
            {
                foundPublication = item;
            }
        }

        foundPublication.Link = this.link.Text;
        foundPublication.Date = this.DatePicker.Text;
        foundPublication.Description = this.Beschreibung.Text;
        foundPublication.Name = this.name.Text;
        foundPublication.Type = this.typ.Text;
        try
        {
            foundPublication.MediumId = ((Medium)this.medium.SelectedItem).Id;
        }
        catch (NullReferenceException ex)
        {
            return;
        }
    }
    else
    {
        publication.Id = GetId();

        publication.Link = this.link.Text;
        publication.Date = this.DatePicker.Text;
        publication.Description = this.Beschreibung.Text;
        publication.Name = this.name.Text;
        publication.Type = this.typ.Text;
        try
        {
            publication.MediumId = ((Medium)this.medium.SelectedItem).Id;
        }
        catch (NullReferenceException ex)
        {
            return;
        }

        this.publikationen.Items.Add(publication);
    }

    this.Beschreibung.Text = "";
    this.DatePicker.SelectedDate = null;
    this.link.Text = "";
    this.medium.SelectedIndex = -1;
    this.name.Text = "";
    this.typ.SelectedIndex = -1;
}
```



Saxonia Systems
So geht Software.

Bloaters

LONG PARAMETER LIST

Methoden haben sehr viele Parameter wodurch sie schwer zu verwenden sind und die Bedeutung einzelner Parameter nicht klar ist.

Beispiel Win API (C++):

```
BOOL WINAPI CreateProcess(  
    _In_opt_ LPCTSTR lpApplicationName,  
    _Inout_opt_ LPTSTR lpCommandLine,  
    _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes,  
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    _In_ BOOL bInheritHandles,  
    _In_ DWORD dwCreationFlags,  
    _In_opt_ LPVOID lpEnvironment,  
    _In_opt_ LPCTSTR lpCurrentDirectory,  
    _In_ LPSTARTUPINFO lpStartupInfo,  
    _Out_ LPPROCESS_INFORMATION lpProcessInformation );
```



PRIMITIVE OBSESSION

Es werden vor allem primitive Datentypen eingesetzt statt Objekte. Dadurch geht der Kontext der Daten verloren, was zu Verständnisschwierigkeiten und Programmierfehlern führt.

```
private void DrawRectangle(int x, int y, int width, int height)
```



```
var rectangle = new Rectangle  
{  
    Size = new Size { Height = 10, Width = 20 },  
    Position = new Point { X = 1, Y = 1 }  
};  
  
Draw(rectangle);
```

ACHTUNG: Durch schlecht gewählte Datenobjekte können Abhängigkeiten entstehen!!!



Bloaters

PRIMITIVE OBSESSION

Es werden vor allem primitive Datentypen eingesetzt statt Objekte. Dadurch geht der Kontext der Daten verloren, was zu Verständnisschwierigkeiten und Programmierfehlern führt.

```
private string command;

private void connect(object sender, EventArgs e)
{
    inputField.Visibility = Visibility.Hidden;
    string user = cSttring1.Text;
    string pass = cSttring2.Text;
    string cStr = cSttring3.Text;
    Properties.Settings.Default.DBUsername = user;
    Properties.Settings.Default.DBPassword = pass;
    Properties.Settings.Default.ConnectionString = cStr;
```

```
    inputField.Visibility = Visibility.Collapsed;
    if (command == "Import")
    {
        ImportDB1();
    } else if (command == "Export")
    {
        ExPorterDB1();
    }
}
```

Importieren Exportieren		
Username	Password	ConnectionString
<input type="text"/>	<input type="text"/>	<input type="text"/>
Anmelden		

```
private void ImportDB(object sender, EventArgs e)
{
    inputField.Visibility = Visibility.Visible;
    cSttring1.Text = Properties.Settings.Default.DBUsername ?? "";
    cSttring2.Text = Properties.Settings.Default.DBPassword ?? "";
    cSttring3.Text = Properties.Settings.Default.ConnectionString ?? "";

    this.command = "Import";
}
```



DATA CLUMPS

Der Kontext verlangt, dass gewisse Daten gemeinsam zur Verfügung stehen. Diese werden aber immer als eigene Objekte behandelt, wodurch der Kontext verloren geht und Fehler entstehen können weil nur ein Teil der Daten vorhanden bzw. valide ist.

```
private void DrawRectangle(int x, int y, int width, int height)
```



```
var rectangle = new Rectangle
{
    Size = new Size { Height = 10, Width = 20 },
    Position = new Point { X = 1, Y = 1 }
};

Draw(rectangle);
```



Bloaters

DATA CLUMPS

Der Kontext verlangt, dass gewisse Daten gemeinsam zur Verfügung stehen. Diese werden aber immer als eigene Objekte behandelt, wodurch der Kontext verloren geht und Fehler entstehen können weil nur ein Teil der Daten vorhanden bzw. valide ist.

ACHTUNG: Durch schlecht gewählte Datenobjekte können Abhängigkeiten entstehen!!!

```
public class Order
{
    public List<Product> Products { get; set; }
}

public class Customer
{
    public Address Address { get; set; }
}

Ship(order, customer);

Cancel(order, customer);

void Ship(Order order, Customer customer)
{
    if (order == null) throw new Exception();
    if (customer == null) throw new Exception();
}
```



```
public class Order
{
    public List<Product> Products { get; set; }
    public Customer Customer { get; private set; }
    public Order(Customer customer)
    {
        this.Customer = customer;
    }
}

Ship(order);

Cancel(order);
```



Bloaters

TRAIN WRECK*

Methodenaufrufe werden aneinander gekoppelt wie Waggons. Wenn in einem Aufruf ein Fehler auftritt kann dieser schwer geprüft werden da Debugging kaum möglich ist. Zusätzlich kann die Codezeile nur schwer verstanden werden.

```
new NLogFacade(Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location), "Trace.log"));
```



Bloaters

BLOATED CONSTRUCTOR*

Konstruktoren die zu viele Aufgaben übernehmen und daher zu Problemen bei der Instanziierung führen können. Probleme entstehen bspw. durch den Zugriff auf andere Dienste während der Instanziierung.

```
public Projects()
{
    InitializeComponent();
    delete = Delete.GetInstance();
    ProjectRepo.ProjectCollectionImported += ProjectRepo_ProjectCollectionImported;
    projectsList = new List<Project>()
    {
        new Project(...),
        new Project(...),
        new Project(...),
    };

    foreach (var item in projectsList)
    {
        publisherListView.Items.Add(item);
    }

    ProjectRepo.Projects = projectsList;
}
```

Service Zugriffe in eine Initialisierungsmethode
Auslagern und diese gezielt nach der Instanziierung
aufrufen.



Saxonia Systems
So geht Software.

Bloaters

BLOATED UTILS*

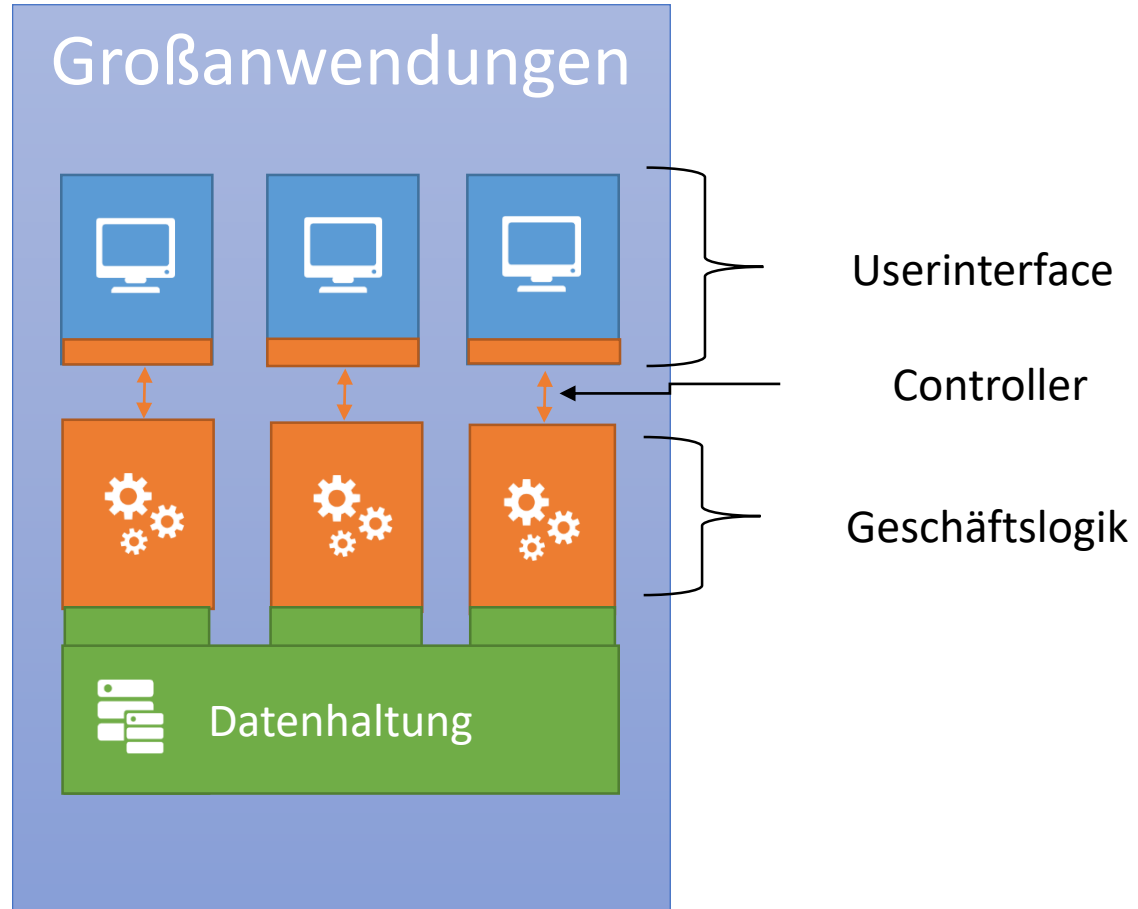
„Util“ Klassen oder Namespace enthalten eigentlich nur Funktionalität die allgemeingültig und frei von jeglichem Fachkontext sind. Entwickler verschieben dorthin aber auch gern Dinge bei denen Sie nicht wissen wo sie sonst hin sollen. Dadurch entsteht ungewollte Kopplung.



Saxonia Systems
So geht Software.

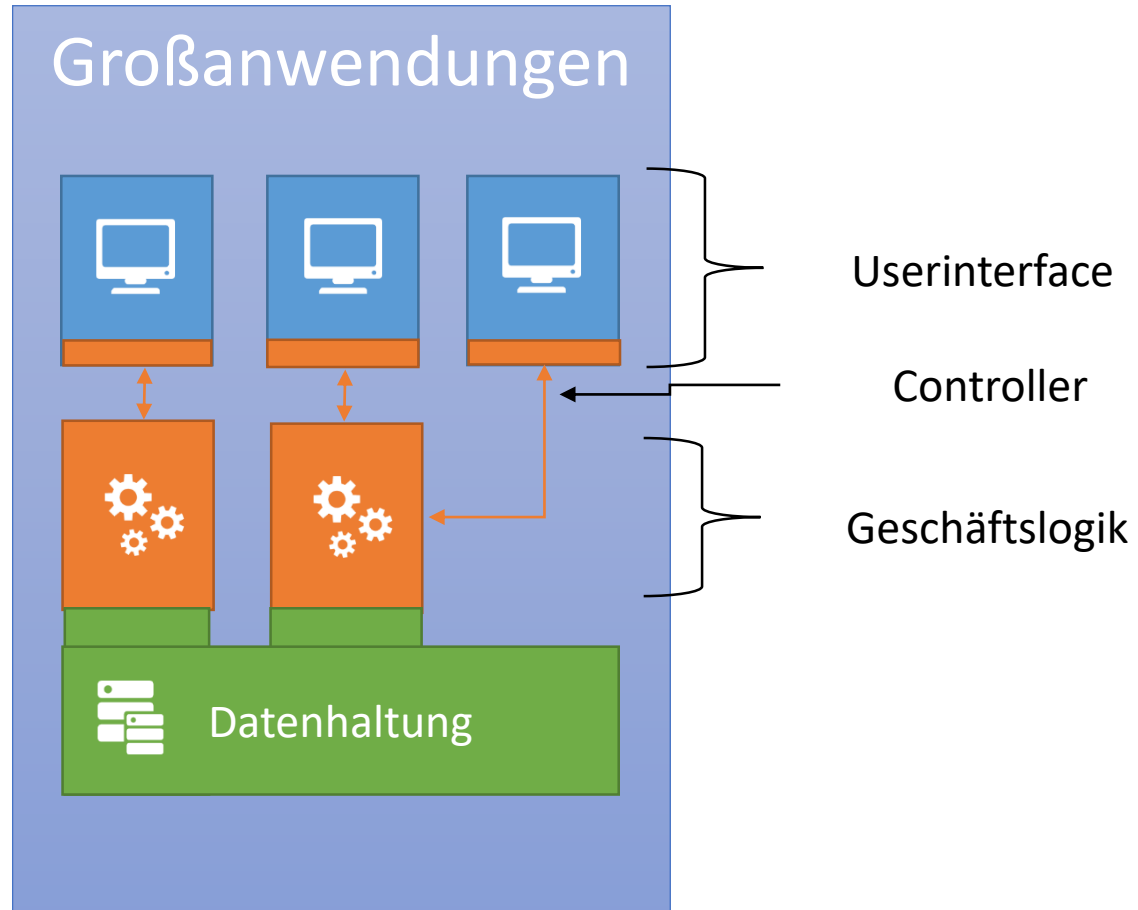
Bloaters

BLOATED UTILS*



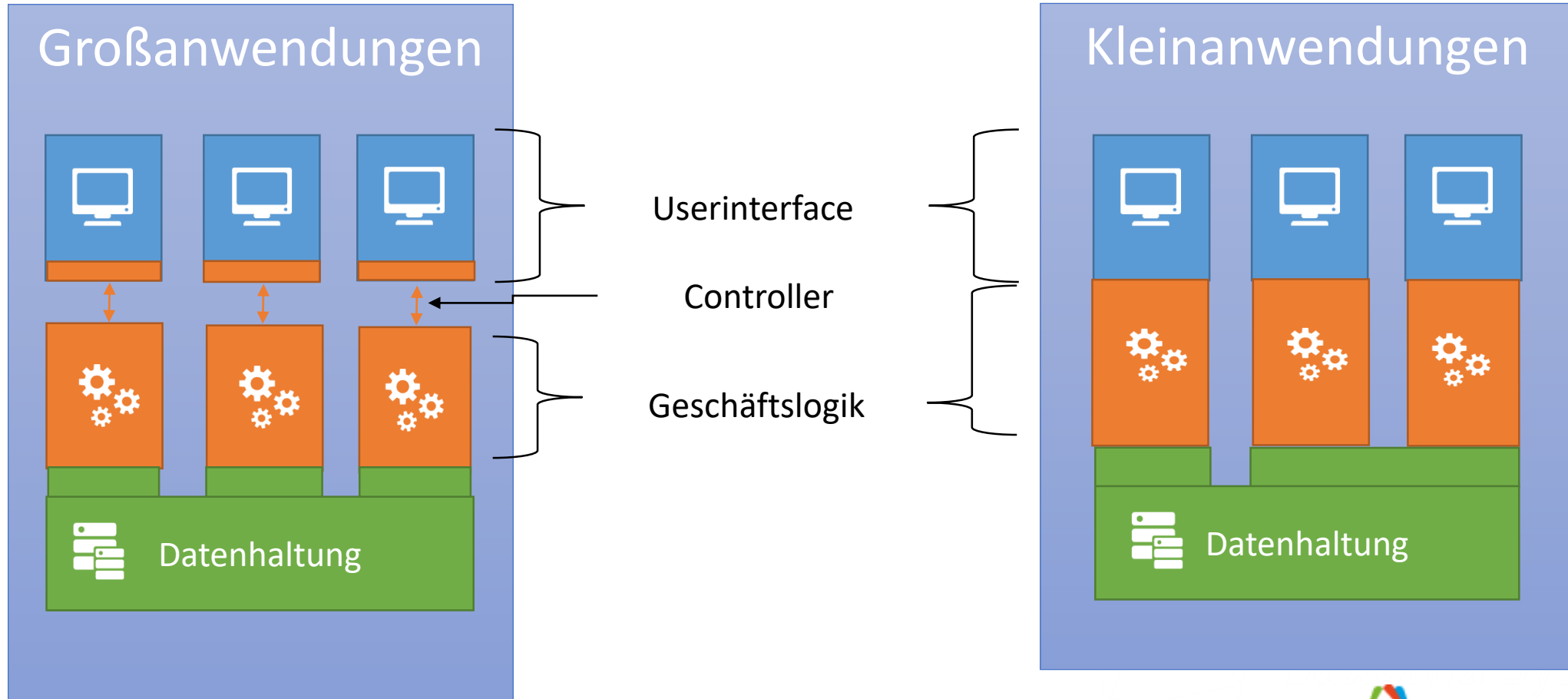
Bloaters

BLOATED UTILS*



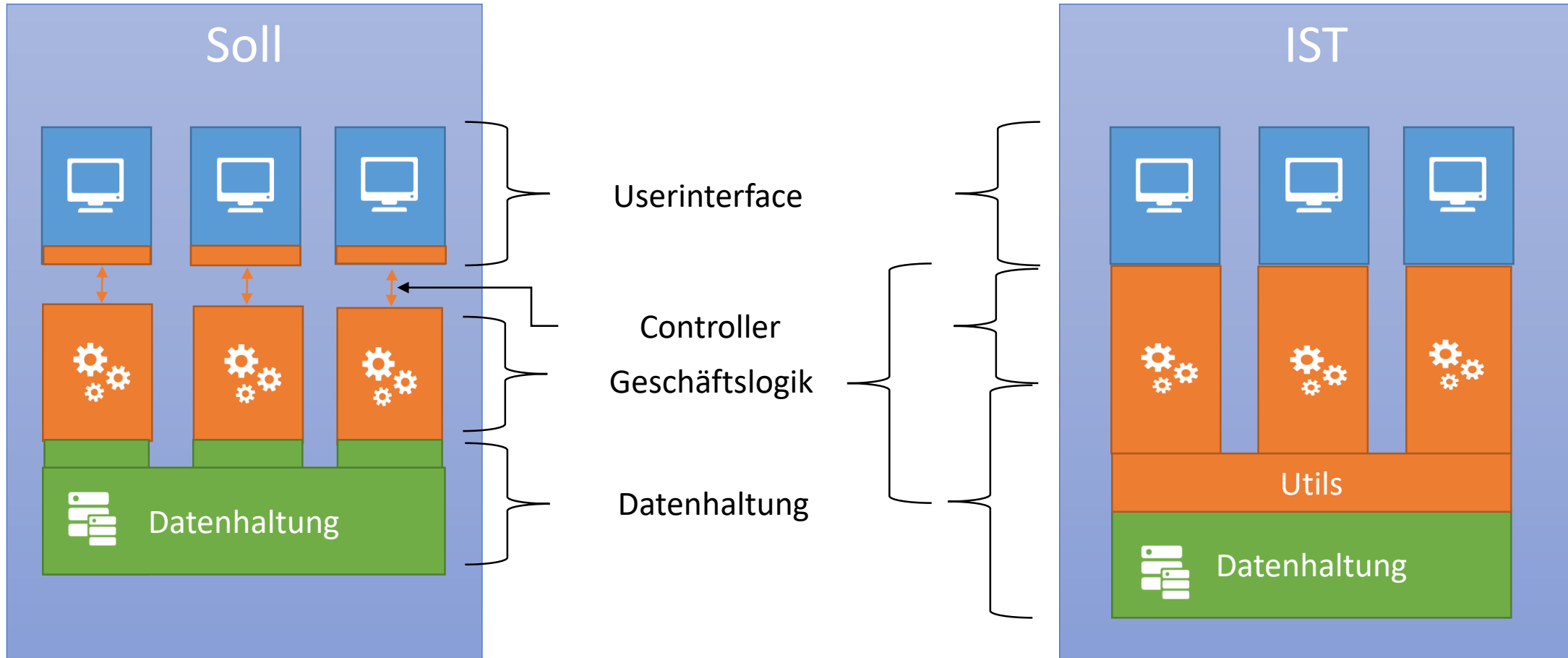
Bloaters

BLOATED UTILS*



Bloaters

BLOATED UTILS*



DISPENSABLES

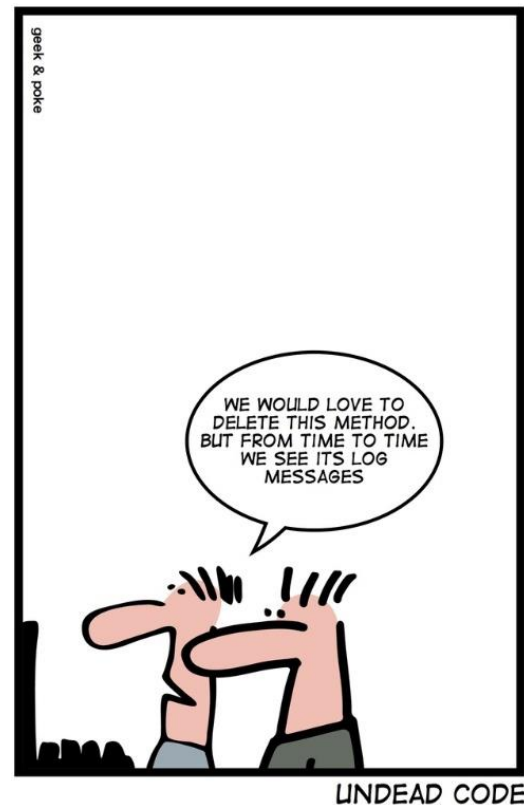


Saxonia Systems
So geht Software.

Dispensables

ERLÄUTERUNG

Bei Dispensables handelt es sich um Code oder Strukturen, die keinem Zweck (mehr) dienen. Diese Dinge werden eigentlich nicht mehr benötigt und erhöhen nur die Komplexität da sie das Volumen der zu beachtenden Strukturen erhöhen und oft nicht zu erkennen ist ob sie noch benötigt werden.



Saxonia Systems
So geht Software.

Dispensables

COMMENTS

Umfangreiche Kommentare altern schnell, sie sind nicht Refaktorisierungssicher und können den Blick auf den eigentlichen Code verstellen.

Code -> Wie?
Kommentar -> Warum?
Methodenname -> Was?



```
// Diese Klasse ermöglicht die Behandlung bestimmter Ereignisse der Einstellungsklasse:  
// Das SettingChanging-Ereignis wird ausgelöst, bevor der Wert einer Einstellung geändert wird.  
// Das PropertyChanged-Ereignis wird ausgelöst, nachdem der Wert einer Einstellung geändert wurde.  
// Das SettingsLoaded-Ereignis wird ausgelöst, nachdem die Einstellungswerte geladen wurden.  
// Das SettingsSaving-Ereignis wird ausgelöst, bevor die Einstellungswerte gespeichert werden.  
internal sealed partial class Settings {  
  
    public Settings() {  
        // // Heben Sie die Auskommentierung der unten angezeigten Zeilen auf, um Ereignishandler zum Speichern und Ändern von Einstellungen  
        //  
        // this.SettingChanging += this.SettingChangingEventHandler;  
        //  
        // this.SettingsSaving += this.SettingsSavingEventHandler;  
        //  
    }  
  
    private void SettingChangingEventHandler(object sender, System.Configuration.SettingChangingEventArgs e) {  
        // Fügen Sie hier Code zum Behandeln des SettingChangingEvent-Ereignisses hinzu.  
    }  
  
    private void SettingsSavingEventHandler(object sender, System.ComponentModel.CancelEventArgs e) {  
        // Fügen Sie hier Code zum Behandeln des SettingsSaving-Ereignisses hinzu.  
    }  
}
```



Saxonia Systems
So geht Software.

Dispensables

DPULICATE CODE AKA CODE CLONES

Geclonter Code ist sich vom Verhalten oder der Struktur sehr ähnlich. Dies behindert die Übersicht über die tatsächliche Funktionalität und macht Anpassungen schwierig, da ggf. mehrere Stellen angepasst werden müssen.

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var publication = new Publication();

    if (!string.IsNullOrEmpty(this.id.Text))
    {
        var id = int.Parse(this.id.Text);
        Publication foundPublication = null;
        foreach (Publication item in this.publikationen.Items)
        {
            if (item.Id == id)
            {
                foundPublication = item;
            }
        }

        foundPublication.Link = this.link.Text;
        foundPublication.Date = this.DatePicker.Text;
        foundPublication.Description = this.Beschreibung.Text;
        foundPublication.Name = this.name.Text;
        foundPublication.Type = this.typ.Text;
        try
        {
            foundPublication.MediumId = ((Medium)this.medium.SelectedItem).Id;
        }
        catch (NullReferenceException ex)
        {
            return;
        }
    }
    else
```

```
        foundPublication.Type = this.typ.Text;
        try
        {
            foundPublication.MediumId = ((Medium)this.medium.SelectedItem).Id;
        }
        catch (NullReferenceException ex)
        {
            return;
        }
    }
    else
    {
        publication.Id = GetId();

        publication.Link = this.link.Text;
        publication.Date = this.DatePicker.Text;
        publication.Description = this.Beschreibung.Text;
        publication.Name = this.name.Text;
        publication.Type = this.typ.Text;
        try
        {
            publication.MediumId = ((Medium)this.medium.SelectedItem).Id;
        }
        catch (NullReferenceException ex)
        {
            return;
        }

        this.publikationen.Items.Add(publication);
```



Dispensables

LAZY CLASS

“Faule Klassen” tun eigentlich kaum etwas. Meist haben sie nur eine Methode, wenige Eigenschaften und delegieren alle Funktionalität an andere Klassen.

ACHTUNG: Durch strikte Schichtentrennung kann der Eindruck von faulen Klassen entstehen!
Wrapper und andere Pattern sehen ebenfalls danach aus.

```
class ProjectRepoInstance
{
    public List<Project> Projects { get; set; }
}
```

```
public class PublicationManager
{
    private IPublicationRepository publicationRepository;

    public PublicationManager(IPublicationRepository publicationRepository)
    {
        this.publicationRepository = publicationRepository;
    }

    public void Save(Publication publication)
    {
        this.publicationRepository.Save(publication);
    }
}
```

Dispensables

DEAD CODE

Toter Code wird zur Laufzeit nicht ausgeführt. Er erhöht nur die Komplexität und das Volumen des Codebasis. Es ist teilweise sehr schwer zu ermitteln ob Code tatsächlich verwendet wird oder nicht.

Code der nichts tut:

```
protected override void OnActivated(EventArgs e)
{
    base.OnActivated(e);
}
```

Code der unnötige Aktionen ausführt:

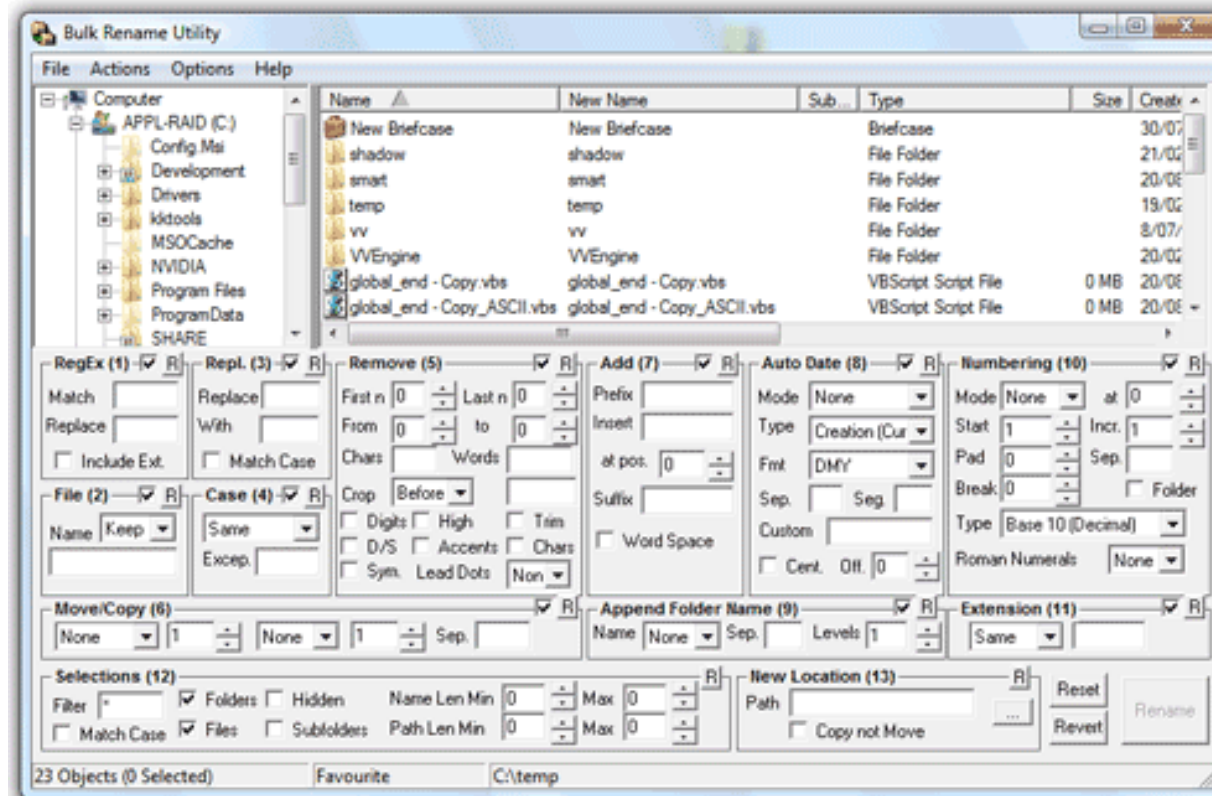
```
var x = MisterDeleteDB.connection.GetSchema("TABLES");
var y = MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Publisher" });
```



Dispensables

SPECULATIVE GENERALITY

Hierbei handelt es sich um Code der auf Verdacht umgesetzt wurde um sich gegen evtl. zukünftige Anforderungen abzusichern. Dadurch ist er komplexer als notwendig, behindert das Verständnis und erhöht den Wartungsaufwand.



Saxonia Systems
So geht Software.

Dispensables

DATA CLASS

ACHTUNG: Gilt nicht unbedingt als Smell...

Data Classes enthalten nur Datenfelder und evtl. getter und setter. Sie sind nur Container ohne Funktionalität und haben keine Möglichkeit selbst auf den Daten zu operieren.

```
public class MediaData
{
    public List<Publication> Media { get; set; }
}
```

```
public class Publisher
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Link { get; set; }
}
```



OBJECT-ORIENTATION ABUSERS



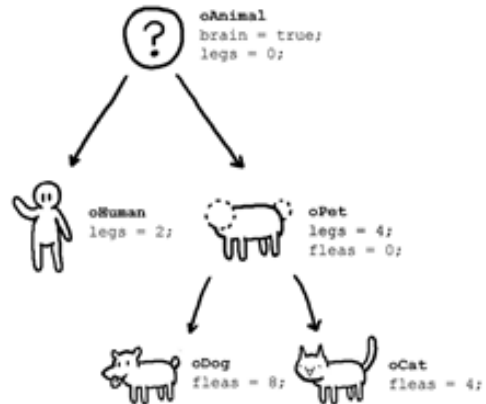
Saxonia Systems
So geht Software.

Object-Orientation Abusers

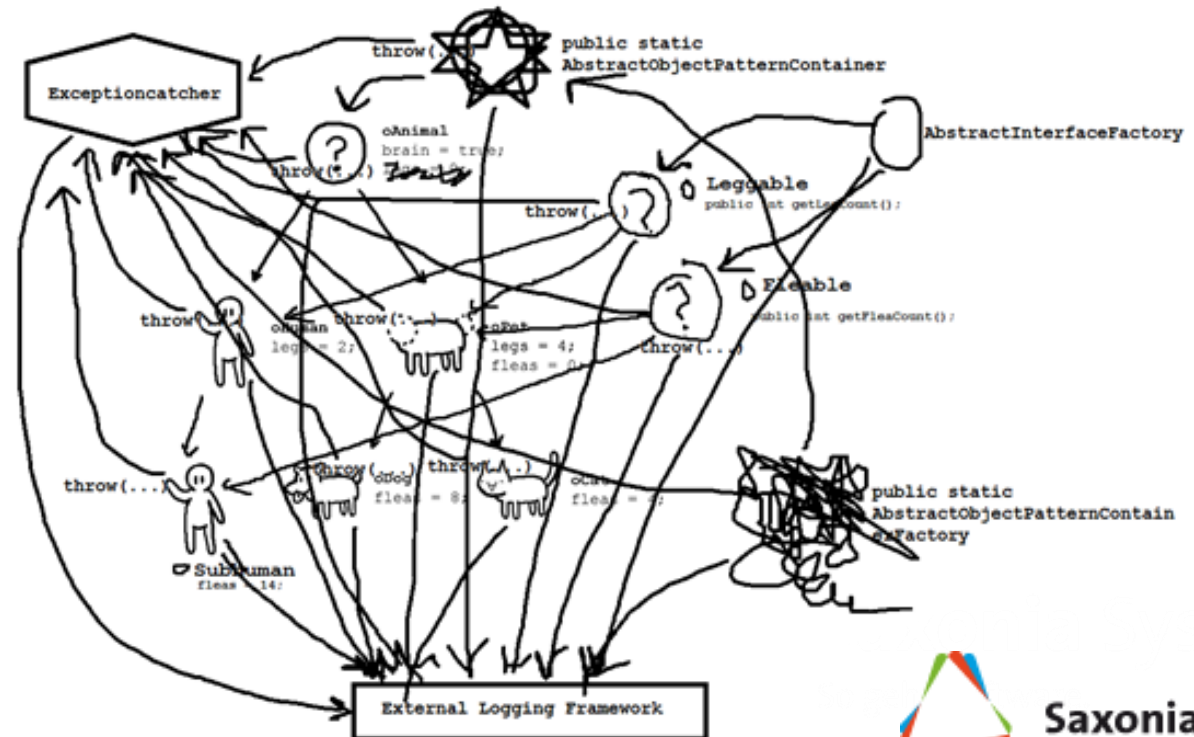
ERLÄUTERUNG

Nicht alles was uns die objektorientierte Programmierung erlaubt ist auch gut. Falsch angewendet können Objekte und ihre Vererbungshierarchien den Code unleserlich machen und zusätzliche Komplexität verursachen.

What OOP users claim



What actually happens



Object-Orientation Abusers

ALTERNATIVE CLASSES WITH DIFFERENT INTERFACES

Zwei Klassen machen nahezu das Gleiche aber mit unterschiedlichen Schnittstellen.

```
public interface IPublicationRepository
{
    int Store(Publication publication);
}

public interface IPublicationStore
{
    int Store(Publication publication);
}
```



Object-Orientation Abusers

REFUSED BEQUEST

Abgeleitete Klassen stellen nur einen Teil der Funktionalität ihrer Elternklasse bereit. Auf die Weise verletzen sie das Liskowsche Substitutions Prinzip, wonach sich alle Elemente einer Vererbungshierarchie erwartungsgemäß gleich darstellen müssen.

```
public class PublicationList : IReadOnlyCollection<Publication>
{
    List<Publication> internalList = new List<Publication>();

    public void Add(Publication publication)
    {
        internalList.Add(publication);
    }

    public IEnumerator<Publication> GetEnumerator()
    {
        return internalList.GetEnumerator();
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }

    public int Count { get; }
}
```

„Replace inheritance with delegation“

```
public class PublicationList : List<Publication>
{
    public new int GetHashCode()
    {
        return 42;
    }
}
```

Object-Orientation Abusers

TEMPORARY FIELD

Felder von Klassen werden als „Zwischenspeicher“ innerhalb eines komplexen Workflows genutzt. Teilweise sind sie leer, Teilweise aber nicht. Sie koppeln die Methoden aneinander und definieren einen Status der Klasse der zu Fehlern führt, wenn der Status unvorhergesehen geändert wird.

```
public class PublishingDataManager
```

```
{  
    private List<PublicationData> publicationData;  
    private List<Medium> media;  
    private List<Publication> publications;  
  
    public List<PublicationData> ImportData()  
    {  
        this.ImportMedia();  
        this.ImportPublications();  
  
        return this.publicationData;  
    }  
}
```



```
public class PublicationImporter
```

```
{  
    public List<PublicationData> ImportData()  
    {  
        var media = this.ImportMedia();  
        var importPublications = this.ImportPublications(media);  
  
        return importPublications;  
    }  
}
```



Eigenständige Klasse die nur eine Aufgabe hat.



Saxonia Systems
So geht Software.

Object-Orientation Abusers

TEMPORARY FIELD

Felder von Klassen werden als „Zwischenspeicher“ innerhalb eines komplexen Workflows genutzt. Teilweise sind sie leer, Teilweise aber nicht. Sie koppeln die Methoden aneinander und definieren einen Status der Klasse der zu Fehlern führt, wenn der Status unvorhergesehen geändert wird.

```
private string command;  
  
private void connect(object sender, EventArgs e)  
{  
    inputField.Visibility = Visibility.Hidden;  
    string user = cStgring1.Text;  
    string pass = cStgring2.Text;  
    string cStr = cStgring3.Text;  
    Properties.Settings.Default.DBUsername = user;  
    Properties.Settings.Default.DBPassword = pass;  
    Properties.Settings.Default.ConnectionString = cStr;  
  
    inputField.Visibility = Visibility.Collapsed;  
    if (command == "Import")  
    {  
        ImportDB1();  
    } else if (command == "Export")  
    {  
        ExPorterDB1();  
    }  
}
```

Importieren Exportieren		
Username	Password	ConnectionString
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="button" value="Anmelden"/>		

```
private void ImportDB(object sender, EventArgs e)  
{  
    inputField.Visibility = Visibility.Visible;  
    cStgring1.Text = Properties.Settings.Default.DBUsername ?? "";  
    cStgring2.Text = Properties.Settings.Default.DBPassword ?? "";  
    cStgring3.Text = Properties.Settings.Default.ConnectionString ?? "";  
  
    this.command = "Import";  
}
```



Object-Orientation Abusers

TEMPORARY FIELD

```
public class PublishingDataManager
{
    private List<PublicationData> publicationData;
    private List<Medium> media;
    private List<Publication> publications;

    public List<PublicationData> ImportData()
    {
        this.ImportMedia();
        this.ImportPublications();

        return this.publicationData;
    }

    private void ImportMedia()
    {
        this.media = new List<Medium>();
        // ...
    }

    private void ImportPublications()
    {
        this.GetPublications();
        this.publicationData = new List<PublicationData>();

        foreach (var publication in publications)
        {
            var publicationData = new PublicationData();
            publicationData.Medium = this.media.Where(x => x.Id == publication.MediumId);
            //...
        }
    }

    private void GetPublications()
    {
        this.publications = new List<Publication>();
        // ...
    }
}
```



„Method Object“



```
public class PublicationImporter
{
    public List<PublicationData> ImportData()
    {
        var media = this.ImportMedia();
        var importPublications = this.ImportPublications(media);

        return importPublications;
    }

    private List<Medium> ImportMedia()
    {
        return new List<Medium>();
        // ...
    }

    private List<PublicationData> ImportPublications(List<Medium> media)
    {
        var publications = this.GetPublications();

        var resultPublications = new List<PublicationData>();

        foreach (var publication in publications)
        {
            var publicationData = new PublicationData();
            publicationData.Medium = media.Where(x => x.Id == publication.MediumId);
            //...
        }

        return resultPublications;
    }

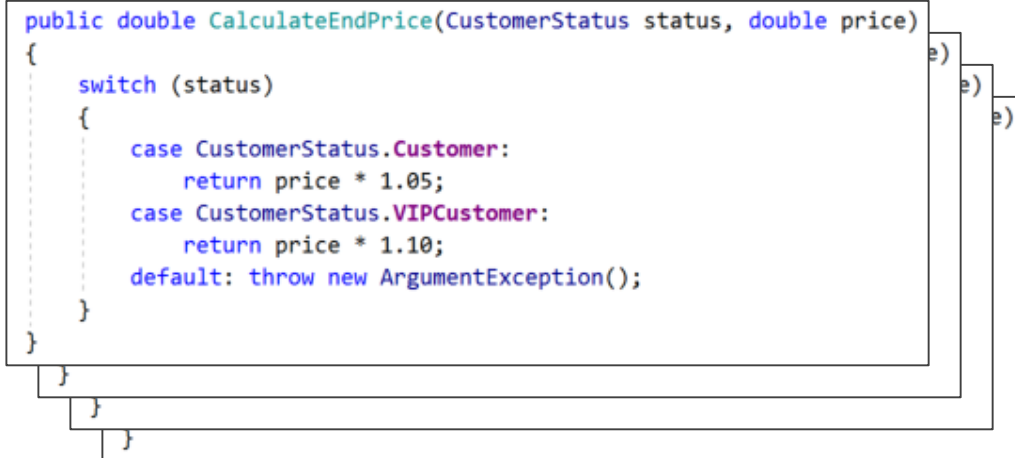
    private List<Publication> GetPublications()
    {
        return new List<Publication>();
        // ...
    }
}
```

Object-Orientation Abusers

SWITCH STATEMENTS

Sehr ähnliche Fallunterscheidungen werden an unterschiedlichen Stellen im Code vorgenommen. Wird eine vergessen, kommt es zu Problemen.

```
public double CalculateEndPrice(CustomerStatus status, double price)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return price * 1.05;
        case CustomerStatus.VIPCustomer:
            return price * 1.10;
        default: throw new ArgumentException();
    }
}
```



```
public Customer CreateCustomer(CustomerStatus status)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return new Customer { PriceFactor = 1.5 };
        case CustomerStatus.VIPCustomer:
            return new Customer { PriceFactor = 1.10 };
        default:
            throw new ArgumentException();
    }
}

public class Customer
{
    public double PriceFactor { get; set; }
}

public double CalculateEndPrice(Customer customer, double price)
{
    return price * customer.PriceFactor;
}
```

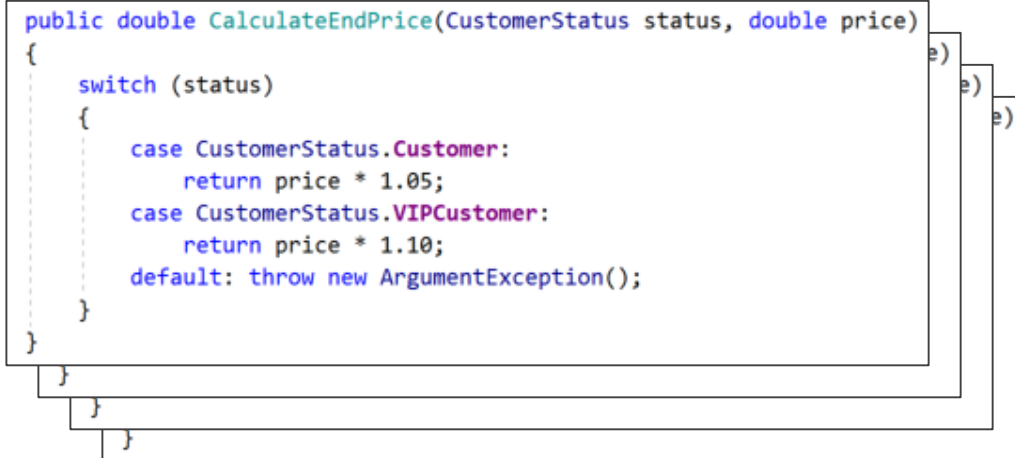


Object-Orientation Abusers

SWITCH STATEMENTS

Sehr ähnliche Fallunterscheidungen werden an unterschiedlichen Stellen im Code vorgenommen. Wird eine vergessen, kommt es zu Problemen.

```
public double CalculateEndPrice(CustomerStatus status, double price)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return price * 1.05;
        case CustomerStatus.VIPCustomer:
            return price * 1.10;
        default: throw new ArgumentException();
    }
}
```



```
public Customer CreateCustomer(CustomerStatus status)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return new Customer { PriceFactor = 1.5 };
        case CustomerStatus.VIPCustomer:
            return new Customer { PriceFactor = 1.10 };
        default:
            throw new ArgumentException();
    }
}

public class Customer
{
    public double PriceFactor { get; set; }
}

public double CalculateEndPrice(Customer customer, double price)
{
    return price * customer.PriceFactor;
}
```

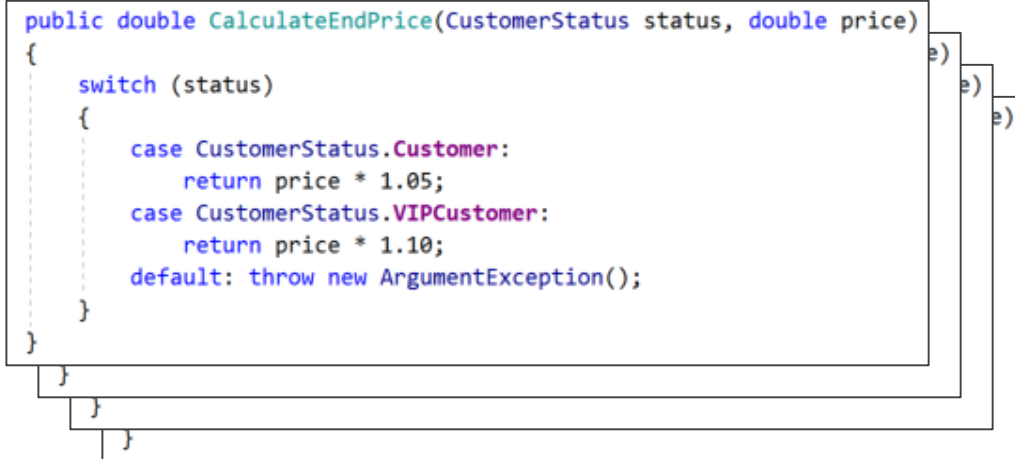


Object-Orientation Abusers

SWITCH STATEMENTS

Sehr ähnliche Fallunterscheidungen werden an unterschiedlichen Stellen im Code vorgenommen. Wird eine vergessen, kommt es zu Problemen.

```
public double CalculateEndPrice(CustomerStatus status, double price)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return price * 1.05;
        case CustomerStatus.VIPCustomer:
            return price * 1.10;
        default: throw new ArgumentException();
    }
}
```



```
public Customer CreateCustomer(CustomerStatus status)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return new Customer();
        case CustomerStatus.VIPCustomer:
            return new VipCustomer();
        default:
            throw new ArgumentException();
    }
}

public class Customer
{
    public double PriceFactor { get; protected set; }

    public Customer()
    {
        PriceFactor = 1.5;
    }
}

public class VipCustomer : Customer
{
    public VipCustomer()
    {
        PriceFactor = 1.10;
    }
}
```

Object-Orientation Abusers

SWITCH STATEMENTS

Fallunterscheidungen werden innerhalb einer Methode vorgenommen um einen Workflow zu realisieren.

```
private void ExPorterDB1()
{
    var x = MisterDeleteDB.connection.GetSchema("TABLES");
    var y = MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Publisher" });
    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Publisher" }).Rows.Count <= 0) ...

    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Medien" }).Rows.Count <= 0) ...

    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Publikationen" }).Rows.Count <= 0) ...

    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Projekte" }).Rows.Count <= 0) ...

    new SqlCommand("DELETE FROM Publikationen;", MisterDeleteDB.connection).ExecuteNonQuery();
    foreach (Publication p in this.publikationen.Items) ...

    new SqlCommand("DELETE FROM Medien;", MisterDeleteDB.connection).ExecuteNonQuery();
    foreach (Medium m in MediaRepo.Media) ...

    new SqlCommand("DELETE FROM Publisher;", MisterDeleteDB.connection).ExecuteNonQuery();
    foreach (Publisher p in PublisherRepo.Publisher) ...

    new SqlCommand("DELETE FROM Projekte;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Aufgaben;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Werkzeuge;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Projekte_Aufgaben;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Projekte_Werkzeuge;", MisterDeleteDB.connection).ExecuteNonQuery();

    foreach (Project item in ProjectRepo.Projects) ...
}
```



```
private void ExPorterDB1()
{
    ExportPublisher();
    ExportMedia();
    ExportPublications();
    ExportProjects();
    CleanupDataBase();
}
```



Object-Orientation Abusers

SWITCH STATEMENTS

Guardclauses prüfen an vielen Stellen im Code ob der übergebene Wert Null ist.

```
private void ExPorterDB1()
{
    var x = MisterDeleteDB.connection.GetSchema("TABLES");
    var y = MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Publisher" });
    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Publisher" }).Rows.Count <= 0) ...

    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Medien" }).Rows.Count <= 0) ...

    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Publikationen" }).Rows.Count <= 0) ...

    if (MisterDeleteDB.connection.GetSchema("TABLES", new string[] { null, null, "Projekte" }).Rows.Count <= 0) ...

    new SqlCommand("DELETE FROM Publikationen;", MisterDeleteDB.connection).ExecuteNonQuery();
    foreach (Publication p in this.publikationen.Items) ...

    new SqlCommand("DELETE FROM Medien;", MisterDeleteDB.connection).ExecuteNonQuery();
    foreach (Medium m in MediaRepo.Medien) ...

    new SqlCommand("DELETE FROM Publisher;", MisterDeleteDB.connection).ExecuteNonQuery();
    foreach (Publisher p in PublisherRepo.Publisher) ...

    new SqlCommand("DELETE FROM Projekte;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Aufgaben;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Werkzeuge;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Projekte_Aufgaben;", MisterDeleteDB.connection).ExecuteNonQuery();
    new SqlCommand("DELETE FROM Projekte_Werkzeuge;", MisterDeleteDB.connection).ExecuteNonQuery();

    foreach (Project item in ProjectRepo.Projects) ...
}
```



```
private void ExPorterDB1()
{
    ExportPublisher();
    ExportMedia();
    ExportPublications();
    ExportProjects();
    CleanupDataBase();
}
```



Object-Orientation Abusers

NULL*

Guardclauses prüfen an vielen Stellen im Code ob der übergebene Wert Null ist da Null im Fehlerfall zurück gegeben wird. Kann durch das **Try-Pattern** verhindert werden.

```
public Invoice SendInvoice(Order order)
{
    if (order == null)
    {
        return null;
    }

    var invoice = new Invoice();

    try
    {
        // ...
    }
    catch (Exception e)
    {
        return null;
    }

    return invoice;
}
```



```
public bool TrySendInvoice(Order order, out Invoice invoice)
{
    invoice = new Invoice();

    if (order == null)
    {
        return false;
    }

    try
    {
        // ...
    }
    catch (Exception e)
    {
        return false;
    }

    return true;
}
```



Object-Orientation Abusers

NULL*

ACHTUNG: üblicherweise werden für Null objects eigene Datentypen angelegt.

Guardclauses prüfen an vielen Stellen im Code ob der übergebene Wert Null ist da Null im Fehlerfall zurück gegeben wird. Kann durch das **Null-Object Pattern** verhindert werden.

```
public Invoice SendInvoice(Order order)
{
    if (order == null)
    {
        return null;
    }

    var invoice = new Invoice();

    try
    {
        // ...
    }
    catch (Exception e)
    {
        return null;
    }

    return invoice;
}
```



```
public Invoice SendInvoice(Order order)
{
    var invoice = new Invoice();

    try
    {
        // ...
    }
    catch (Exception e)
    {
        return null;
    }

    invoice.IsValid = true;
    return invoice;
}
```

Selbst ein Null-Object



Saxonia Systems
So geht Software.

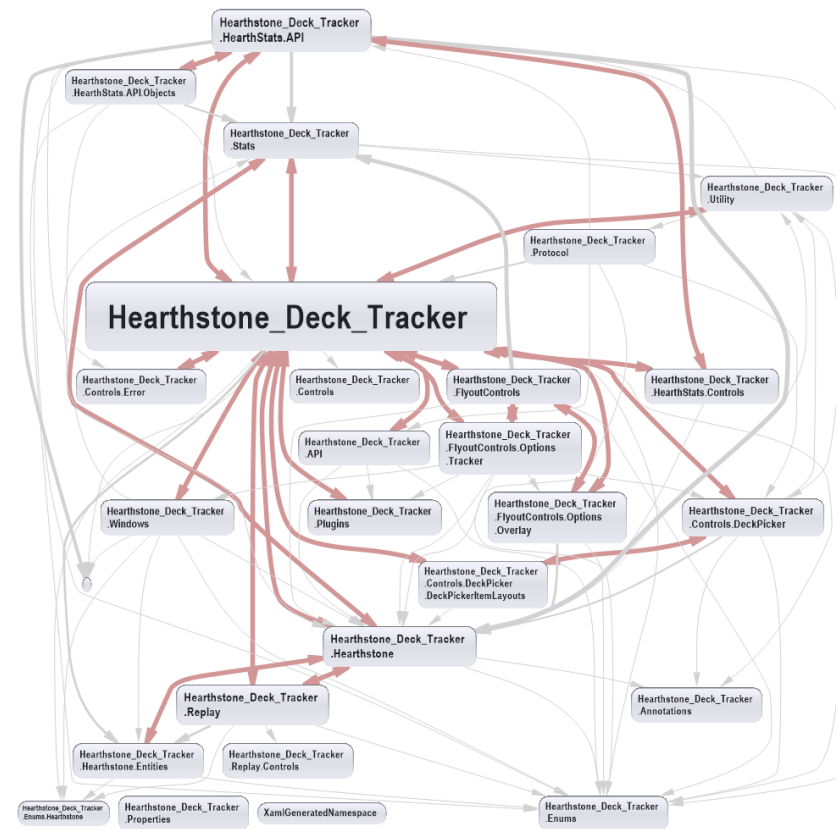
COUPLERS



Saxonia Systems
So geht Software.

ERLÄUTERUNG

Couplers führen zu unnötiger Kopplung innerhalb des Codes. Dadurch werden Änderungen schwierig, automatisierte Tests behindert und Änderungen können unerwartete Auswirkungen haben.



Couplers

FEATURE ENVY

Wenn sich eine Methode theoretisch in der falschen Klasse befindet und deshalb mehr auf den Daten einer anderen Klasse arbeitet. Dies lässt sich auf falsch gestaltete Separation of Concernce zurück führen.

```
public class Invoice
{
    public Order Order { get; set; }

    public double VAT { get; set; }

    public double CalculatePrice()
    {
        var price = this.Order.Article.Sum(x => x.Quantity * x.Price);

        return price * this.VAT;
    }
}
```



```
public class Invoice
{
    public Order Order { get; set; }

    public double VAT { get; set; }

    public double CalculatePrice()
    {
        var price = this.Order.GetPrice();

        return price * this.VAT;
    }
}
```

Envy = Neid



Saxonia Systems
So geht Software.

Couplers

INAPPROPRIATE INTIMACY

Eine Klasse nutzt interne Funktionen einer anderen Klasse oder zieht Rückschlüsse auf die internen Strukturen einer anderen. Sollten sich diese intern ändern kommt es zu Fehlern. Zusätzlich machen Änderungen an der einen Klasse meist auch Änderungen an der anderen notwendig.

```
[TestMethod]
public void PublicationDataMustBeSavedInRepository()
{
    var repoFake = A.Fake<IPublicationRepository>();
    var sut = new PublicationManager(repoFake);

    sut.Save(new Publication());

    A.CallTo(() => repoFake.Save(A<Publication>.Ignored)).MustHaveHappened();
}
```



MESSAGE CHAINS

Bei einer Message Chain wird ein Aufruf entlang einer Objekthierarchie weitergereicht. Der Aufrufer ist damit nicht nur an ein Objekt gebunden, sondern an die gesamte Hierarchie.

```
((ComboBox)((StackPanel)((StackPanel)((Page1)((Frame)((TabItem)e.AddedItems[0])).Content).Content).Content).Children[3]).Children[1]).ItemsSource = x;
```



Couplers

MIDDLE MAN

ACHTUNG: Bei strikter Schichtentrennung werden Middle Men absichtlich verwendet!!!

Ein Middle Man hat selbst keine Logik und reicht Aufgaben eigentlich nur an andere Klassen weiter. Dies erhöht nur das Volumen des Codes und sorgt dafür, dass Aufgaben schwieriger zu verstehen sind.

```
class ProjectViewModel
{
    private readonly IProjectRepository projectRepository;

    public ProjectViewModel(IProjectRepository projectRepository)
    {
        this.projectRepository = projectRepository;
    }

    public Project SelectedProject { get; set; }

    public void Save()
    {
        this.projectRepository.Save(this.SelectedProject);
    }
}
```



CHANGE PREVENTERS

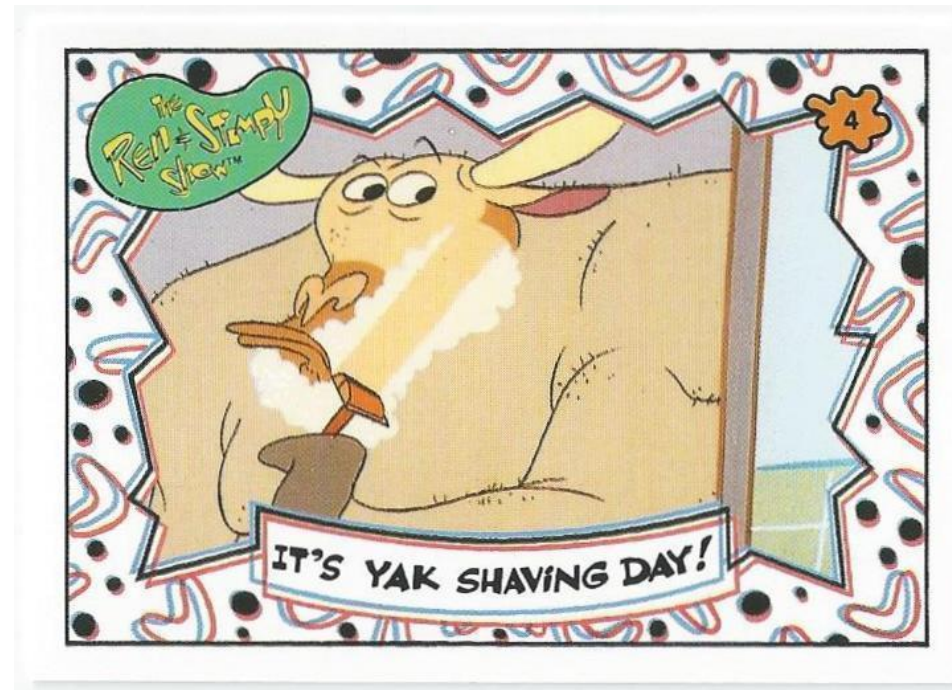


Saxonia Systems
So geht Software.

Change Preventers

ERLÄUTERUNG

Change preventers behindern Änderungen da Anpassungen an einer Stelle des Codes direkt oder indirekt Änderungen an anderen Stellen notwendig machen. Sind die notwendigen Änderungen nicht sofort sichtbar kommt es zu unerwarteten Fehlern.

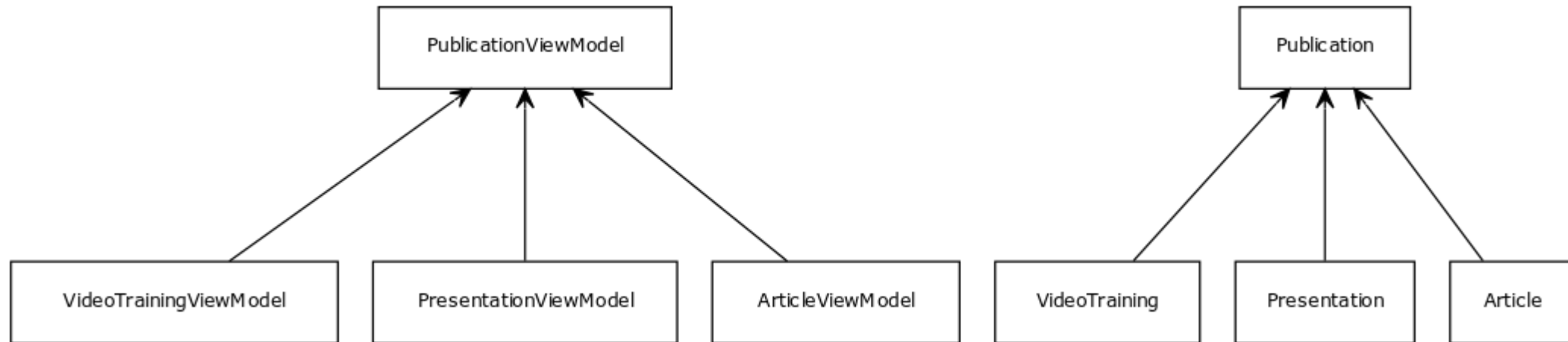


Change Preventers

PARALLEL INHERITANCE HIERARCHIES

Favour Composition over Inheritance...

Vererbungshierarchien sind so komplex geworden, dass die gleichen Objekte in unterschiedlichen Vererbungsweisen gebraucht werden oder zwei Vererbungshierarchien sind so stark von einander abhängig, dass sie sich gegenseitig zur Vererbung zwingen.



Saxonia Systems
So geht Software.

Change Preventers

DIVERGENT CHANGE

Es sind Änderungen **an einer Klasse** notwendig, die nichts mit dem eigentlich Änderungsgrund zu tun haben sollten. Entsteht häufig durch Verletzung des Single Responsibility Principle, wodurch eine Klasse mehrere Aufgaben hat.

```
public class PublicationStorage
{
    public bool StoreInDatabase(Publication publication)
    {
        return true;
    }

    public bool StoreAsJson(Publication publication)
    {
        return true;
    }
}
```



```
public interface IPublicationStorage
{
    bool Store(Publication publication);
}

public class PublicationDatabaseStorage : IPublicationStorage
{
    public bool Store(Publication publication)
    {
        // ...
        return true;
    }
}

public class PublicationJsonStorage : IPublicationStorage
{
    public bool Store(Publication publication)
    {
        // ...
        return true;
    }
}
```



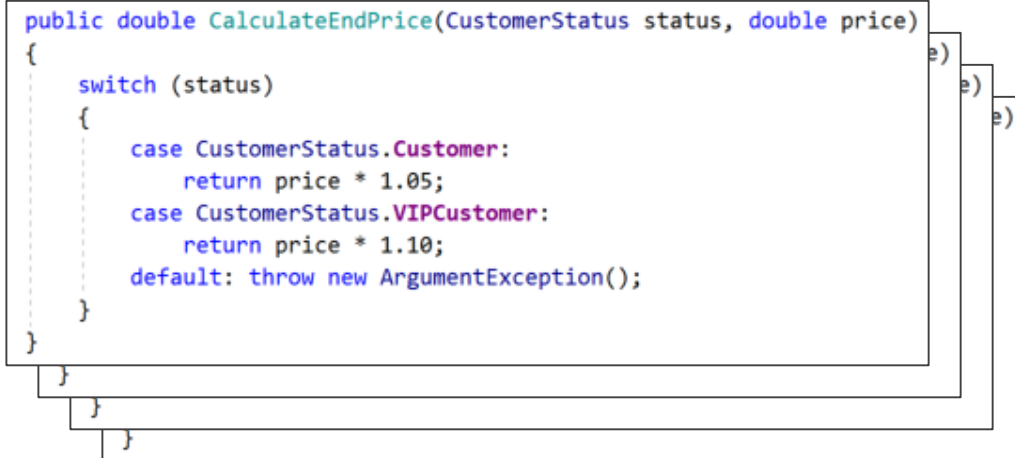
Saxonia Systems
So geht Software.

Change Preventers

SHOTGUN SURGERY

Es ist eine **Vielzahl von kleinen Änderungen an vielen einzelnen Klasse** notwendig. Entsteht häufig durch Verletzung des Single Responsibility Principle, wodurch mehrere Klassen die gleiche Aufgabe haben, zum Beispiel durch Code Clones.

```
public double CalculateEndPrice(CustomerStatus status, double price)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return price * 1.05;
        case CustomerStatus.VIPCustomer:
            return price * 1.10;
        default: throw new ArgumentException();
    }
}
```



```
public Customer CreateCustomer(CustomerStatus status)
{
    switch (status)
    {
        case CustomerStatus.Customer:
            return new Customer { PriceFactor = 1.5 };
        case CustomerStatus.VIPCustomer:
            return new Customer { PriceFactor = 1.10 };
        default:
            throw new ArgumentException();
    }
}

public class Customer
{
    public double PriceFactor { get; set; }
}

public double CalculateEndPrice(Customer customer, double price)
{
    return price * customer.PriceFactor;
}
```



Der Sprecher



Hendrik Lösch

Senior Consultant & Coach

Hendrik.Loesch@saxsys.de

@HerrLoesch

Just-About.Net



WPF-Anwendungen mit MVVM und Prism

Modulare Architekturen verstehen und umsetzen



Windows 8 Store Apps mit MVVM und Prism

XAML-Entwurfsmuster, Bootstrapping, Navigation, Messaging



Test Driven Development – Praxisworkshop

Business-Applikationen testgetrieben entwickeln



Inversion of Control und Dependency Injection

Prinzipien der modernen Software-Architektur ...



Test Driven Development mit C#

Grundlagen, Frameworks, best Practices



Automatisiertes Testen mit Visual Studio 2012

Grundlagen, Testarten und Strategien



Saxonia Systems
So geht Software.