Software Evolution

LEGACY CODE REFAKTORISIEREN

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

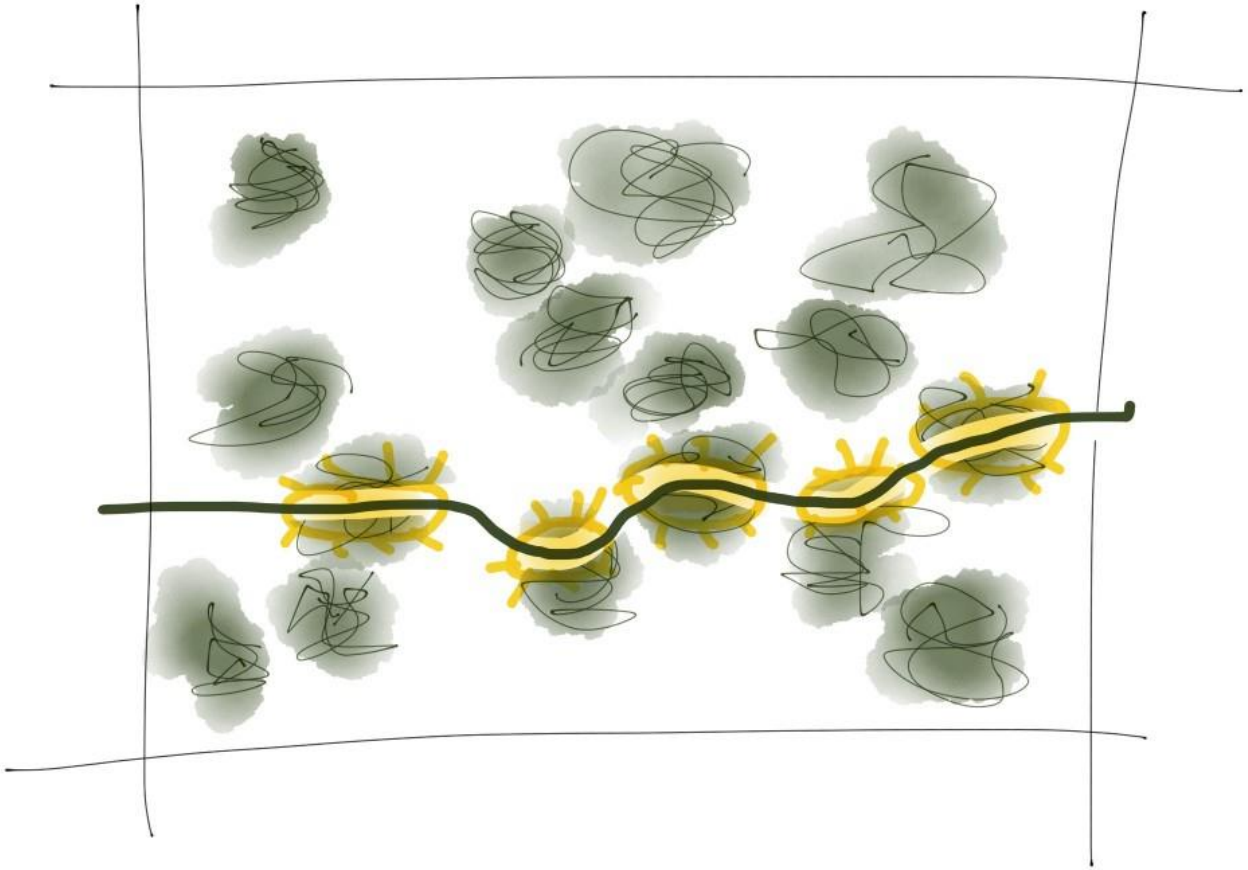
Automatisiertes Testen mit Visual Studio 2012

Grundlagen, Testarten und Strategien

Saxonia Systems
So geht Software.

WAS IST SOFTWARE EVOLUTION

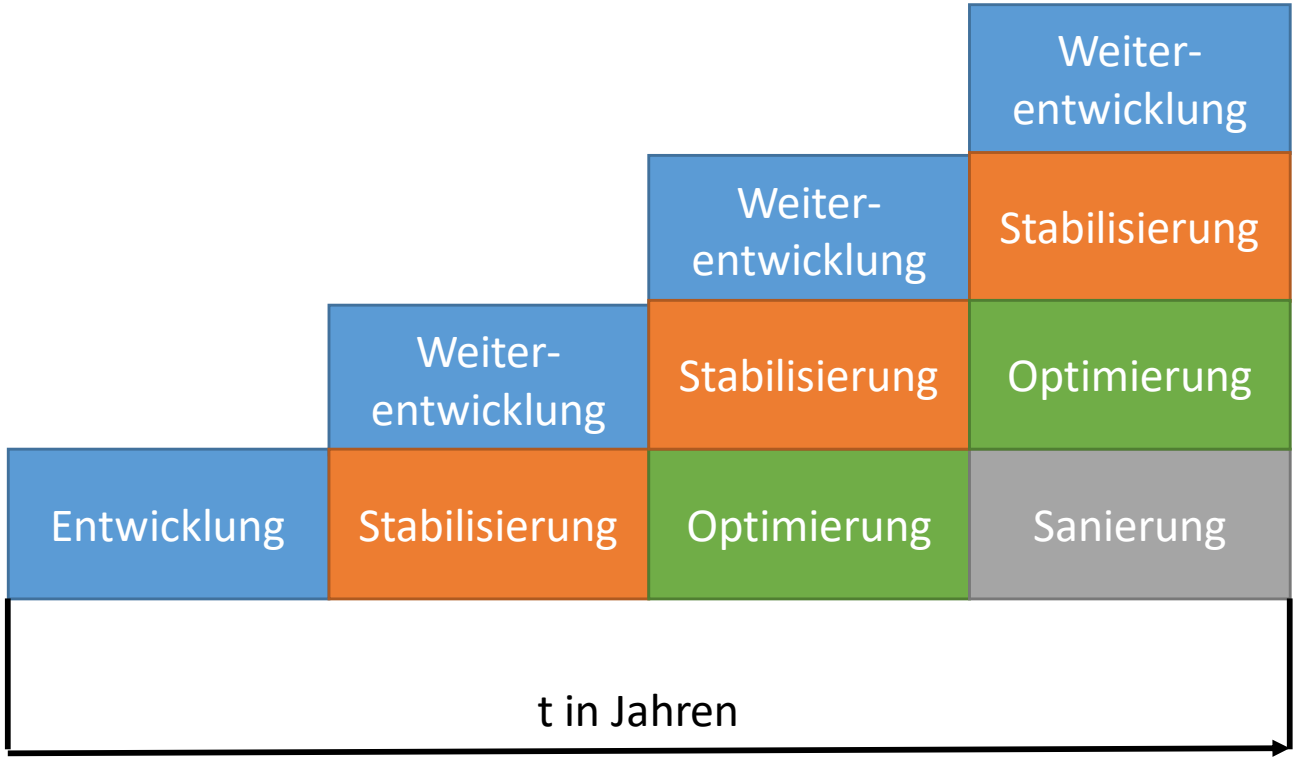
Refactoring



Quelle: Ron Jeffries http://xprogramming.com/articles/refactoring-not-on-the-backlog

Saxonia Systems
So geht Software.

Evolutionäre Softwareentwicklung



Quelle: Softwareevolution Erhaltung und Fortschreibung bestehender Softwaresysteme, Harry M. Sneed, Richard Seidl

Saxonia Systems
So geht Software.


Die Codequalität ist schlecht.
Wir brauchen mehr Automatisierte Tests!
REFAKTORISIEREN !!!!1!1!!!
Wir brauchen Features.
Mehr Features!
FEATURES!! 1!1!11!!!!!

Was ist schlechtes Design?!?

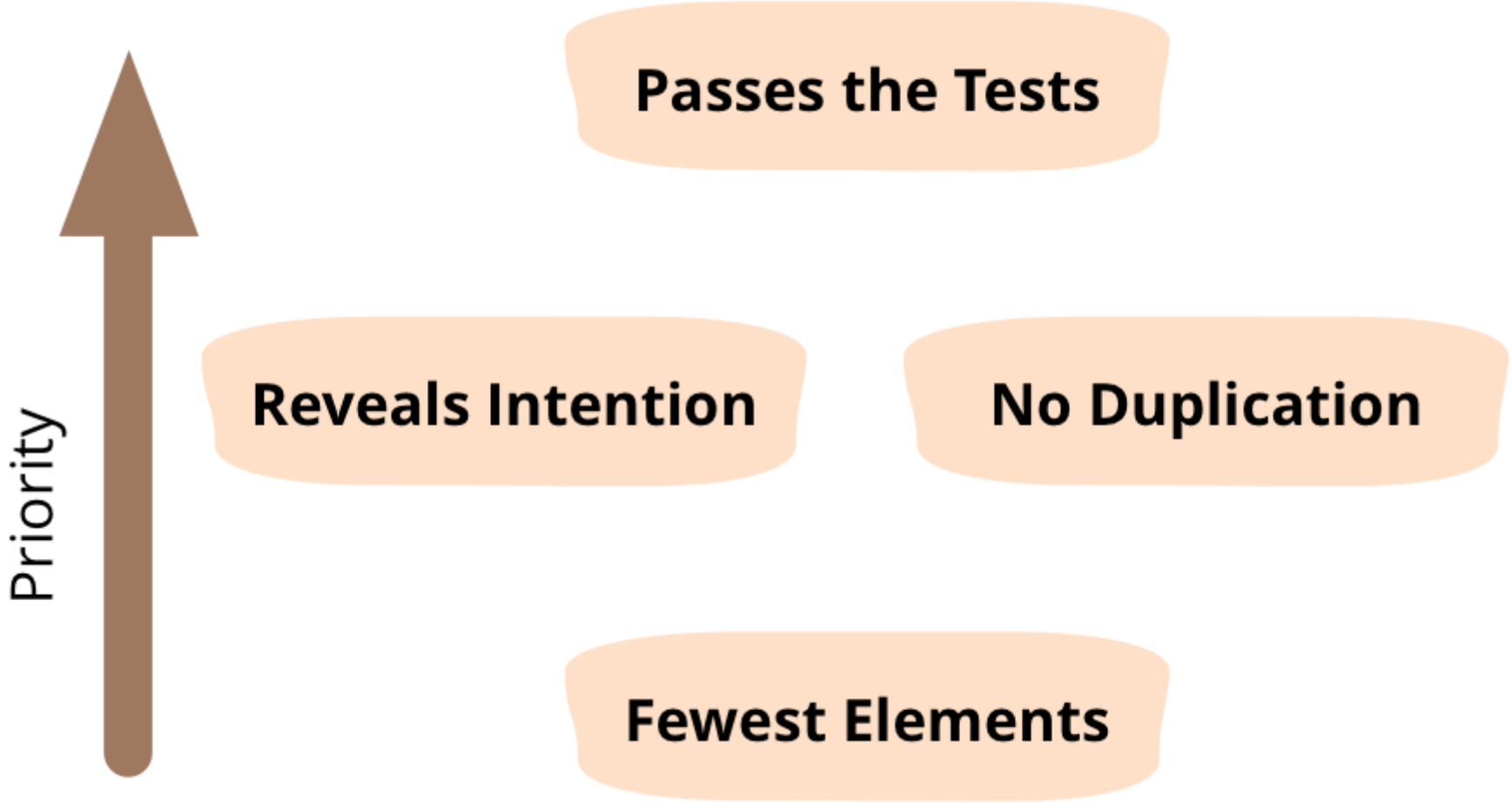
1. It is hard to change because every change affects too many other parts of the system. (Rigidity - Starr)

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)

Quelle: http://www.objectmentor.com/resources/articles/dip.pdf

Was ist gutes Design?!?



Quelle: http://martinfowler.com/bliki/BeckDesignRules.html

Saxonia Systems
So geht Software.

Refactoring

Refactoring bezeichnet in der Software-Entwicklung die manuelle oder automatisierte **Strukturverbesserung von Quelltexten unter Beibehaltung des beobachtbaren Programmverhaltens**.

Dabei sollen die Lesbarkeit, Verständlichkeit, Wartbarkeit und Erweiterbarkeit verbessert werden, mit dem Ziel, den jeweiligen Aufwand für Fehleranalyse und funktionale Erweiterungen deutlich zu senken.

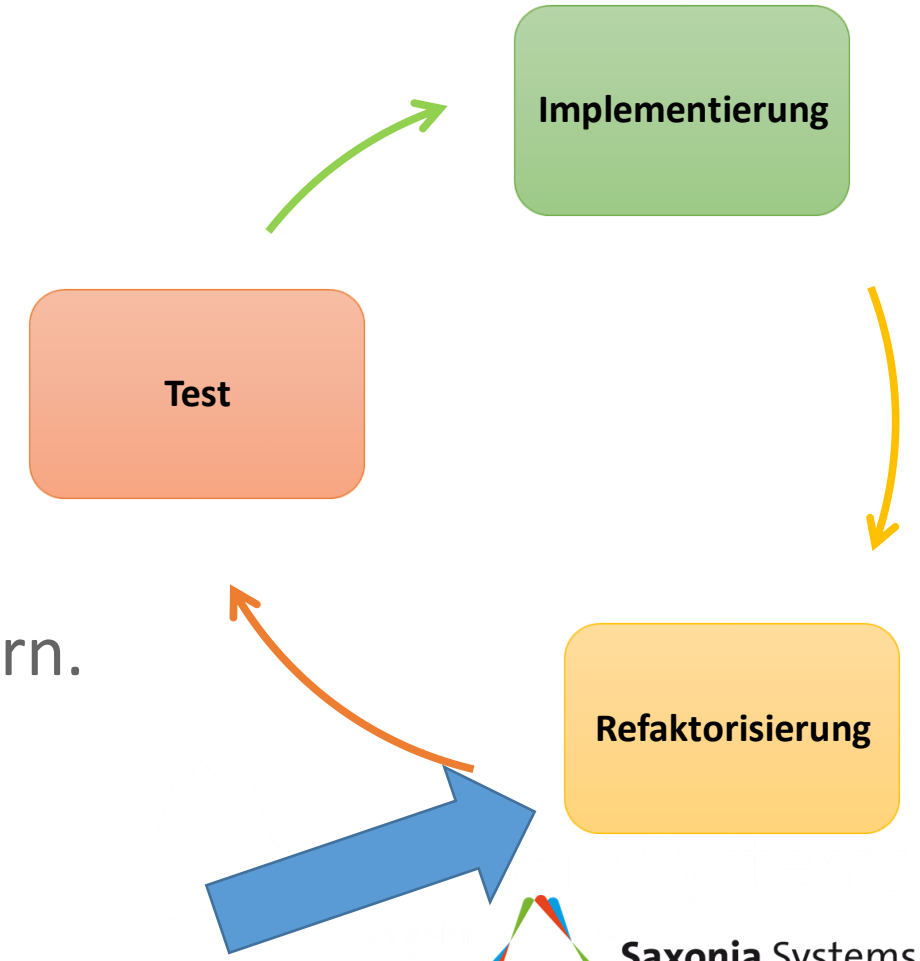
Quelle: https://de.wikipedia.org/wiki/Refactoring

Schreibe nur Code, der verlangt wird.

Entwickle schrittweise Deinen Code.

Wähle möglichst kleine Schritte.

Jeder Schritt muss den Code verbessern.



Saxonia Systems
So geht Software.

DEMO

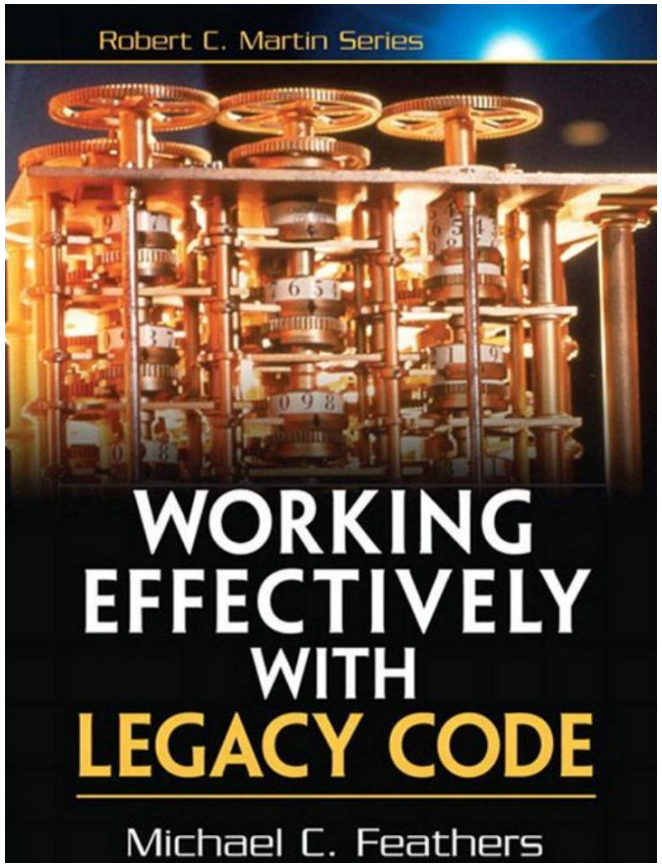
Refactoring

ALLES WIRD GUT

Saxonia Systems
So geht Software.

Das Vorgehen für uns Entwickler

1. Änderungsbereiche aufdecken.
2. Testpunkte aufdecken.
3. Abhängigkeiten aufbrechen.
4. Tests schreiben.
5. Änderungen vornehmen & refaktorisieren.



Wie vorgehen?

1. Quellcode einchecken.
2. Einen Überblick darüber verschaffen was geändert werden soll.
3. Mindestens einen Test schreiben der das bestehende Verhalten charakterisiert.

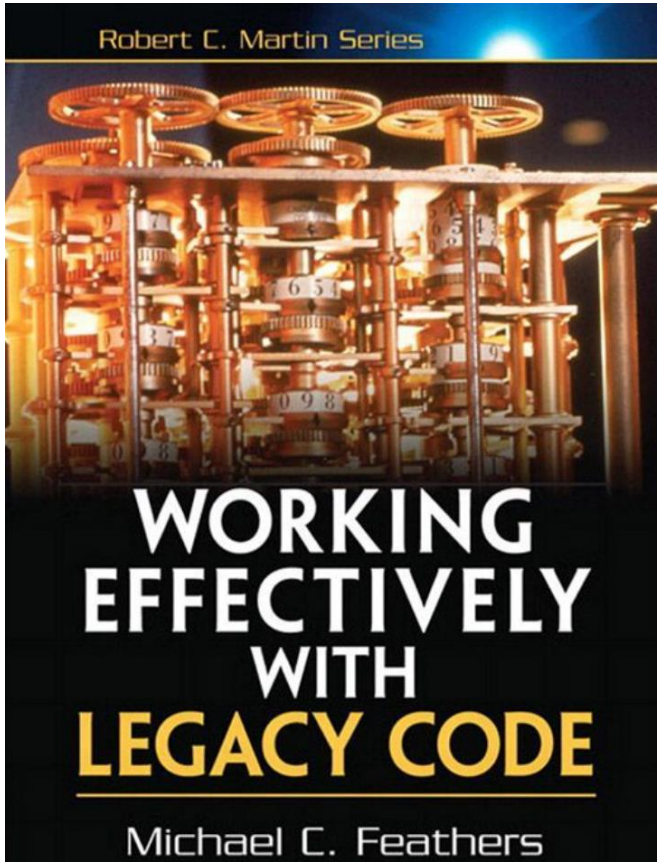
4. Refaktorisieren

5. Prüfen ob das Verhalten sich verändert hat.

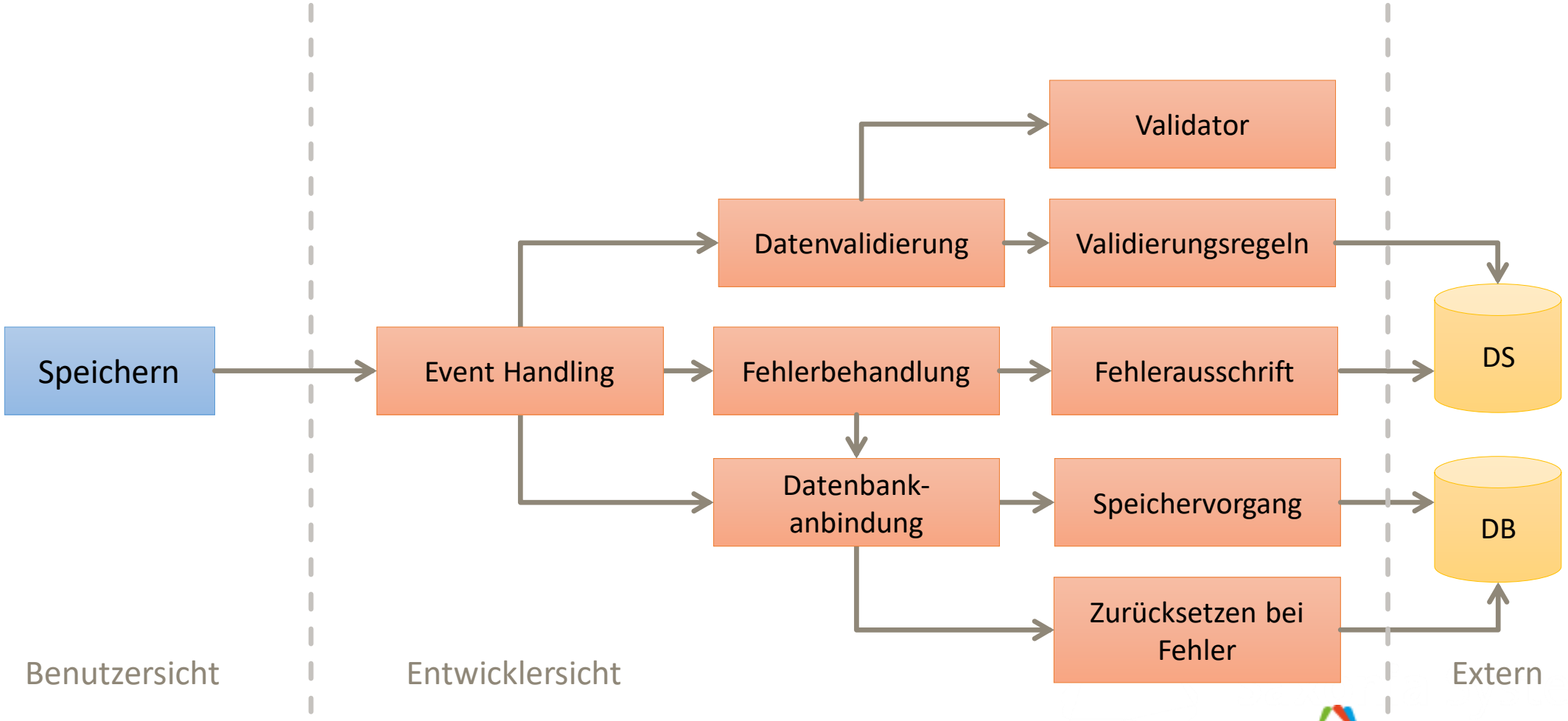
Tests als Grundlage

Code without tests is bad code. It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is.

With tests, we can change the behavior of our code quickly and verifiably. Without them, we really don't know if our code is getting better or worse.

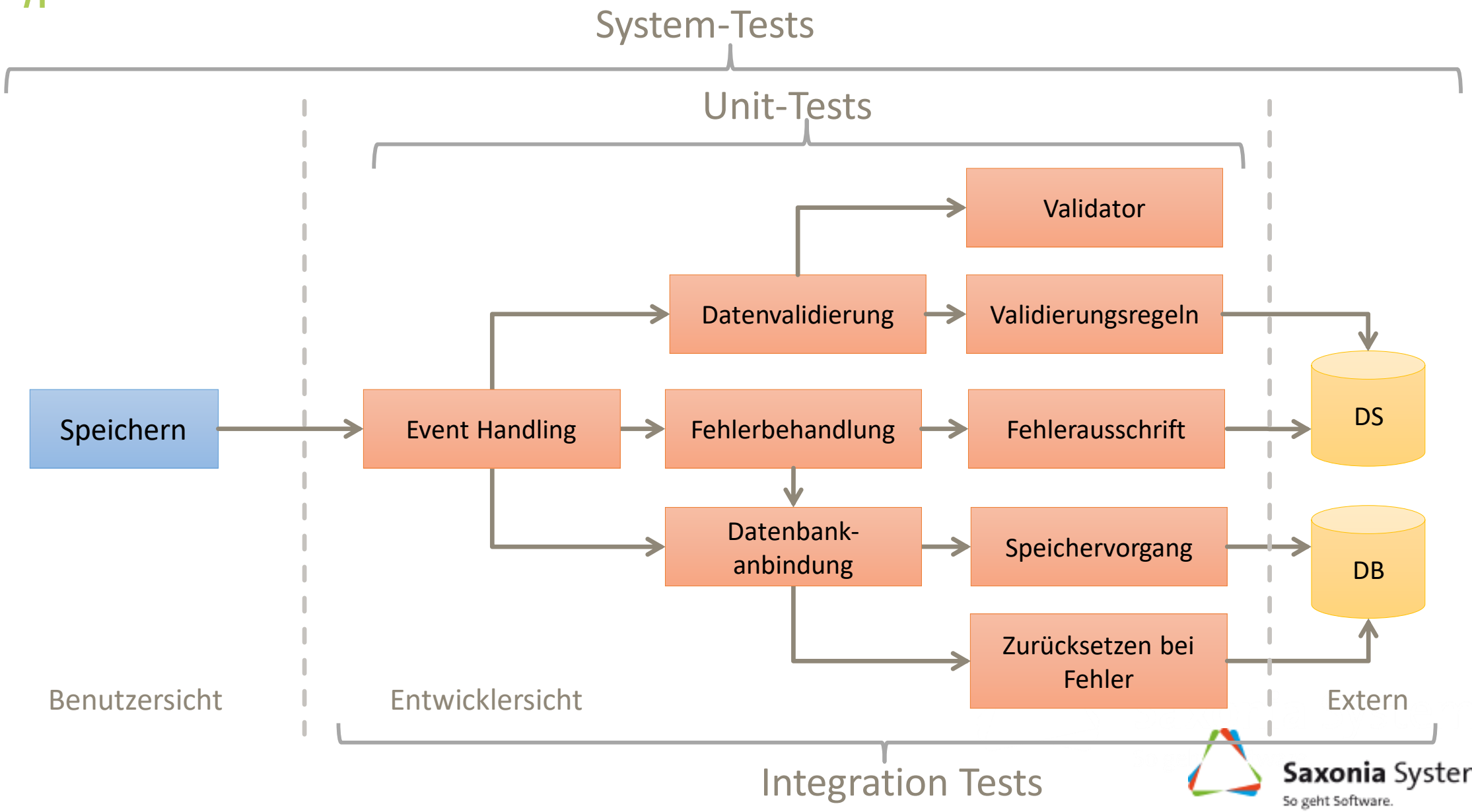


Typische Testarten



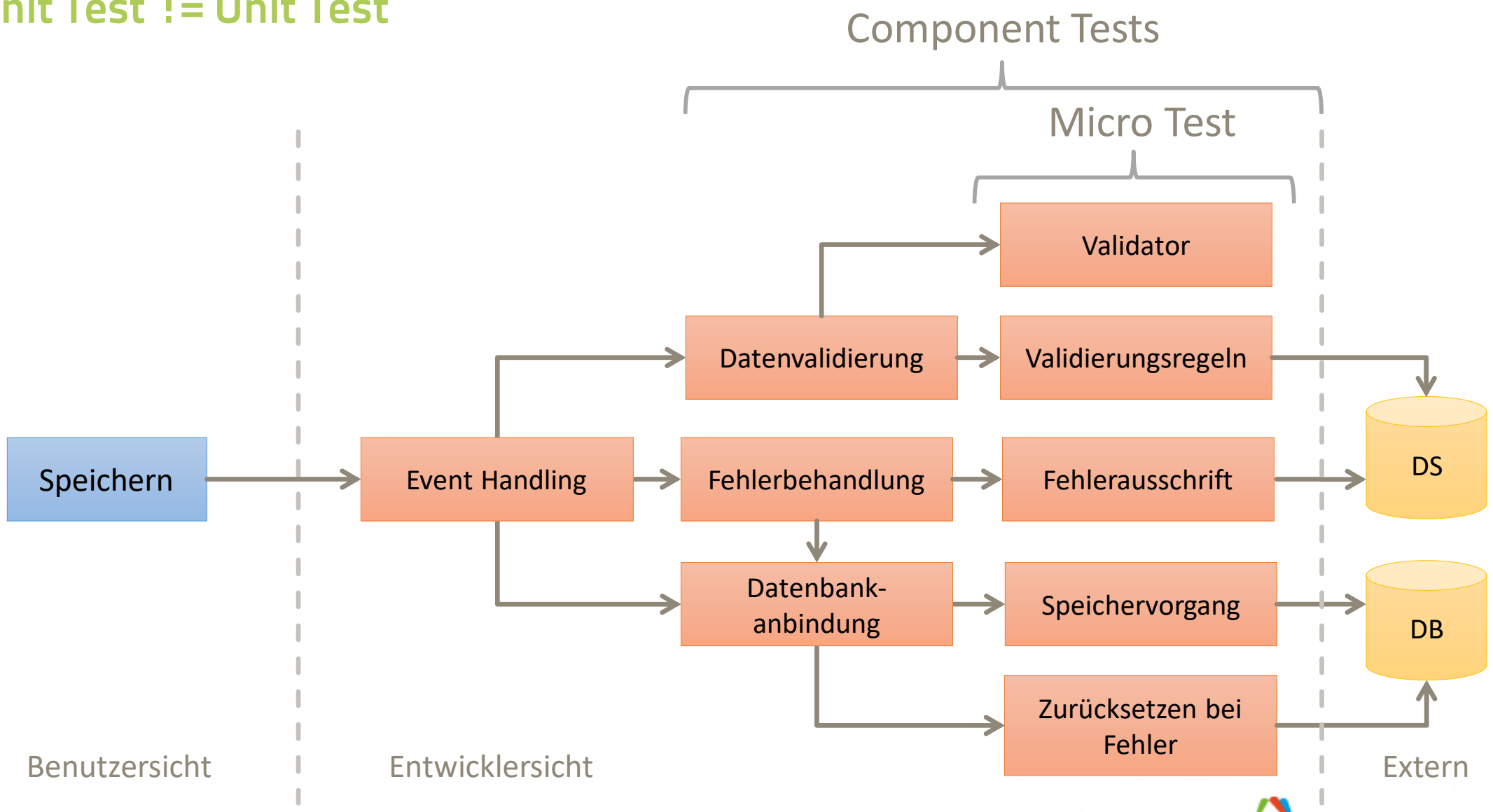
Saxonia Systems
So geht Software.

Typische Testarten



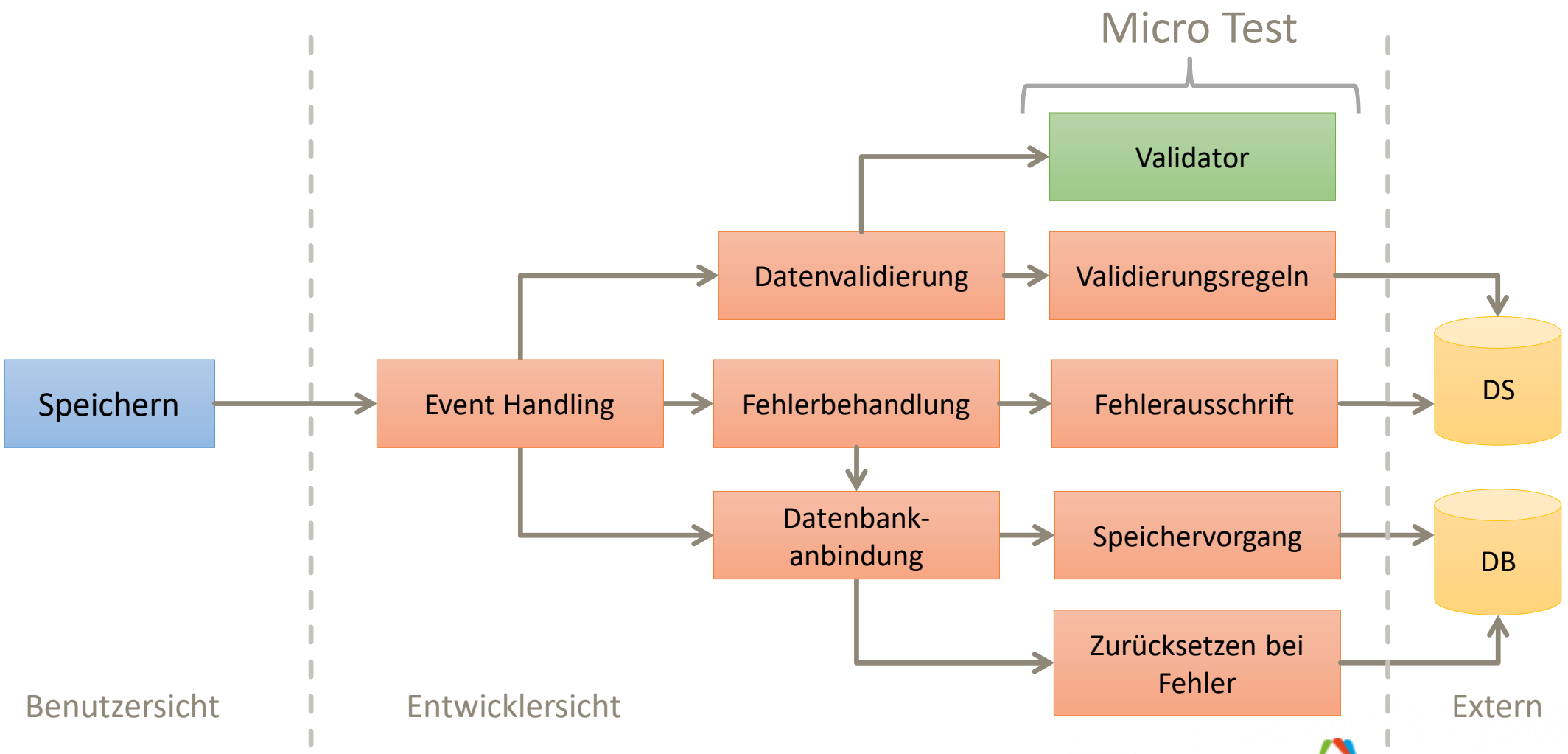
Saxonia Systems
So geht Software.

Unit Test != Unit Test



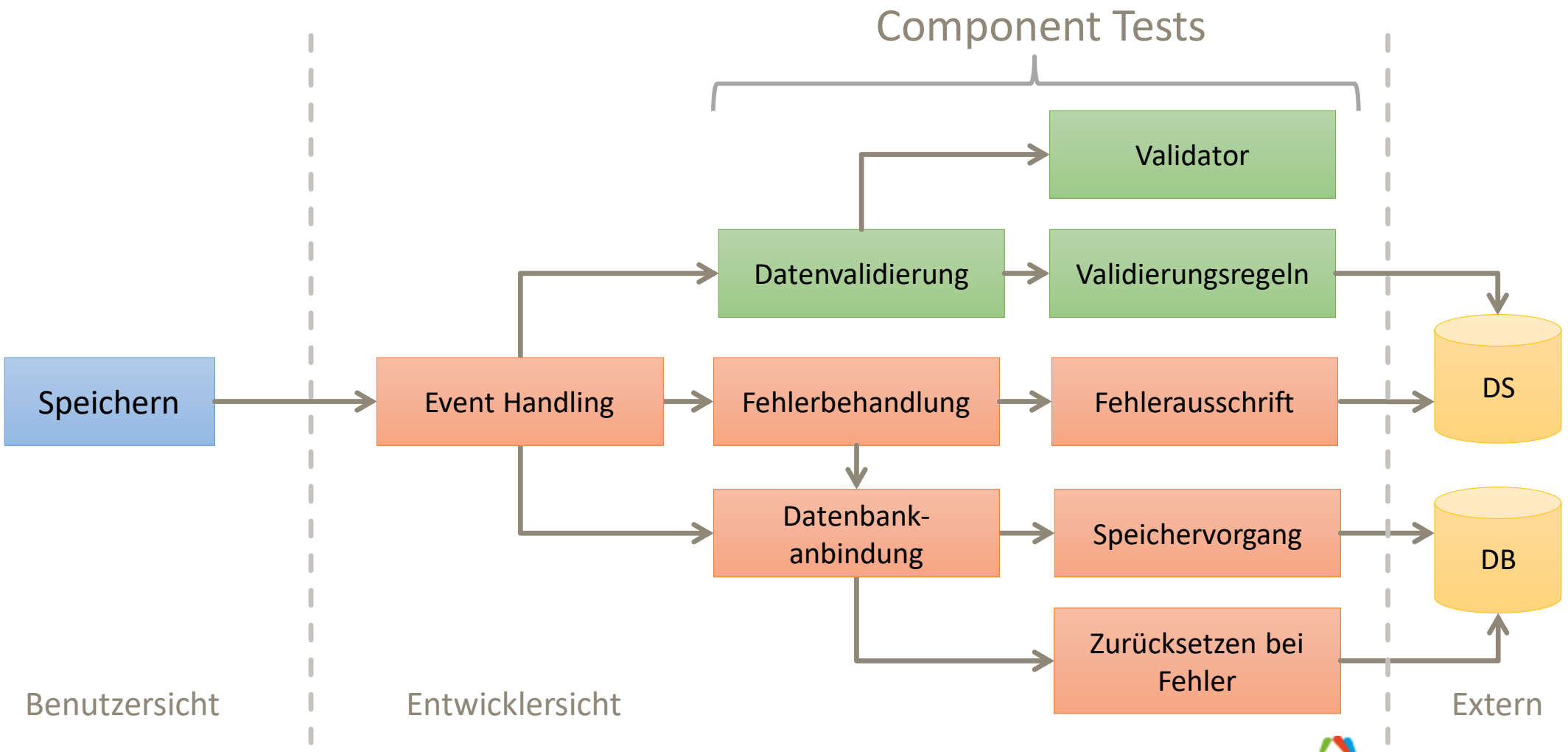
Saxonia Systems
So geht Software.

Baby Step Refactoring



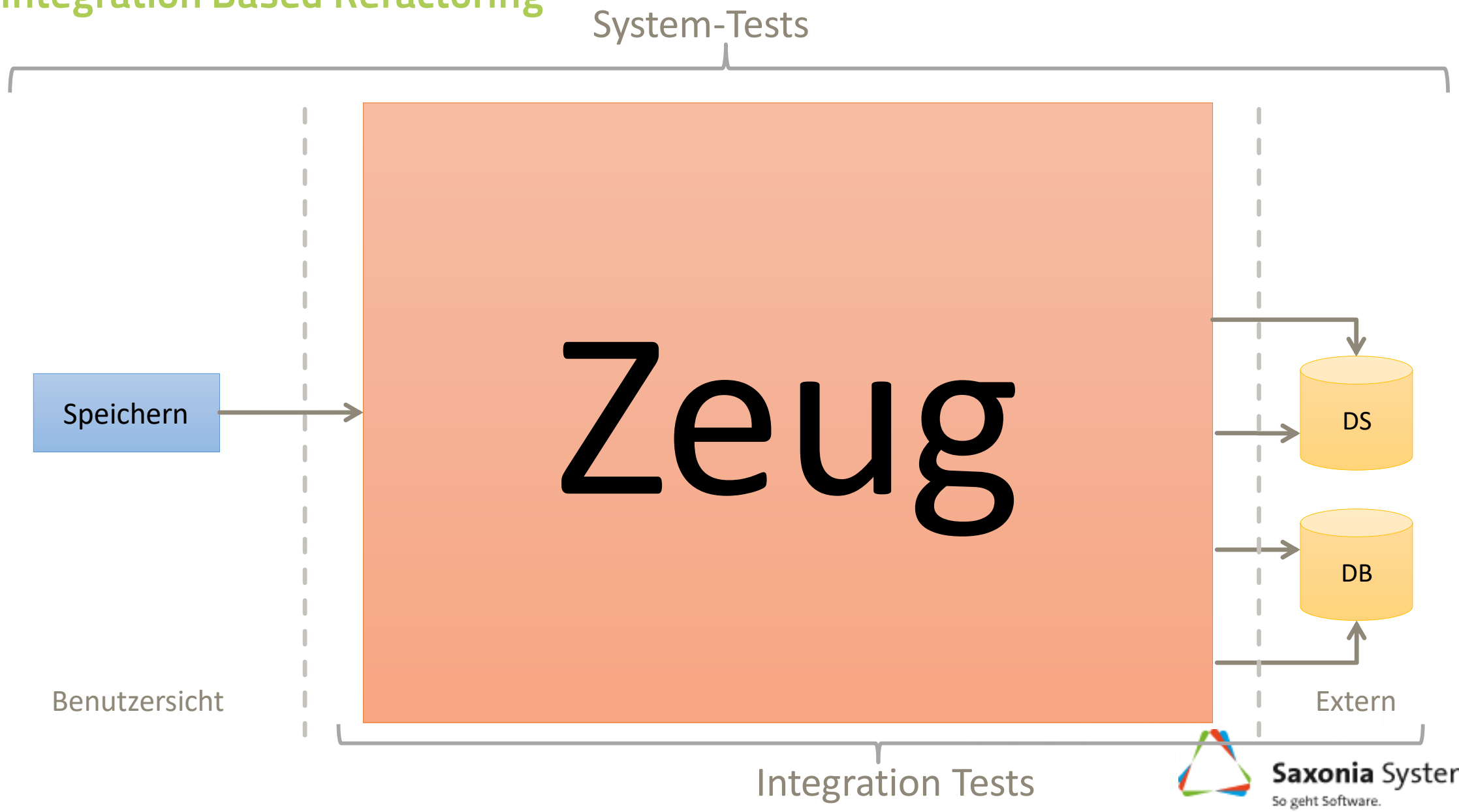
Saxonia Systems
So geht Software.

Component Based Refactoring



Saxonia Systems
So geht Software.

Integration Based Refactoring

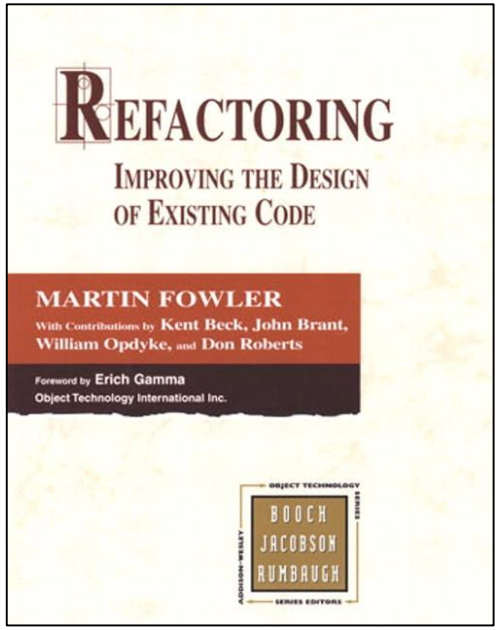


Saxonia Systems
So geht Software.

Refactoring

UND WIE REFAKTORISIERE ICH NUN WIRKLICH?

Refactoring Arten



refactoring.com

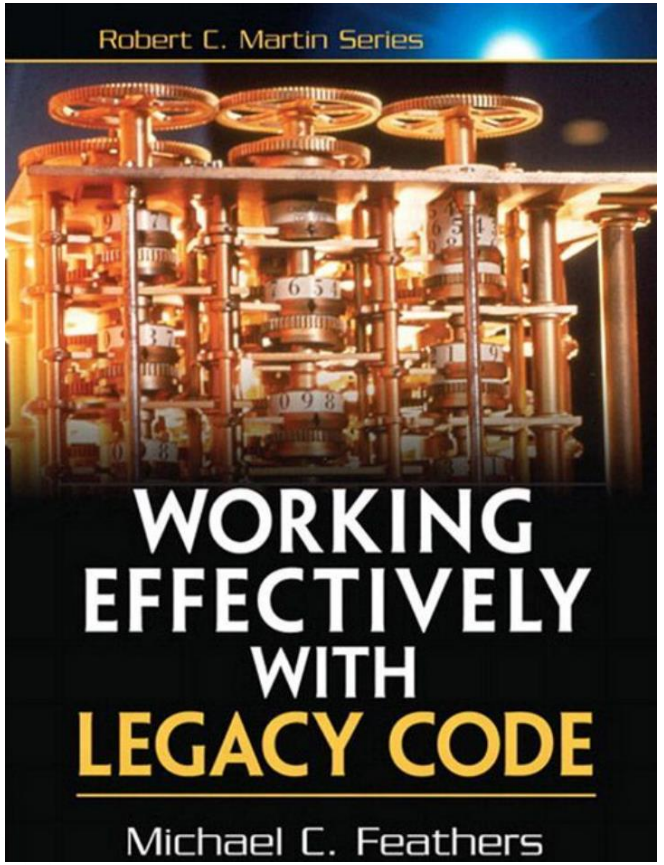
Add Parameter
Change Bidirectional Association to Unidirectional
Change Reference to Value
Change Unidirectional Association to Bidirectional
Change Value to Reference
Collapse Hierarchy
Consolidate Conditional Expression
Consolidate Duplicate Conditional Fragments
Decompose Conditional
Duplicate Observed 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.

Dinge testbar gestalten

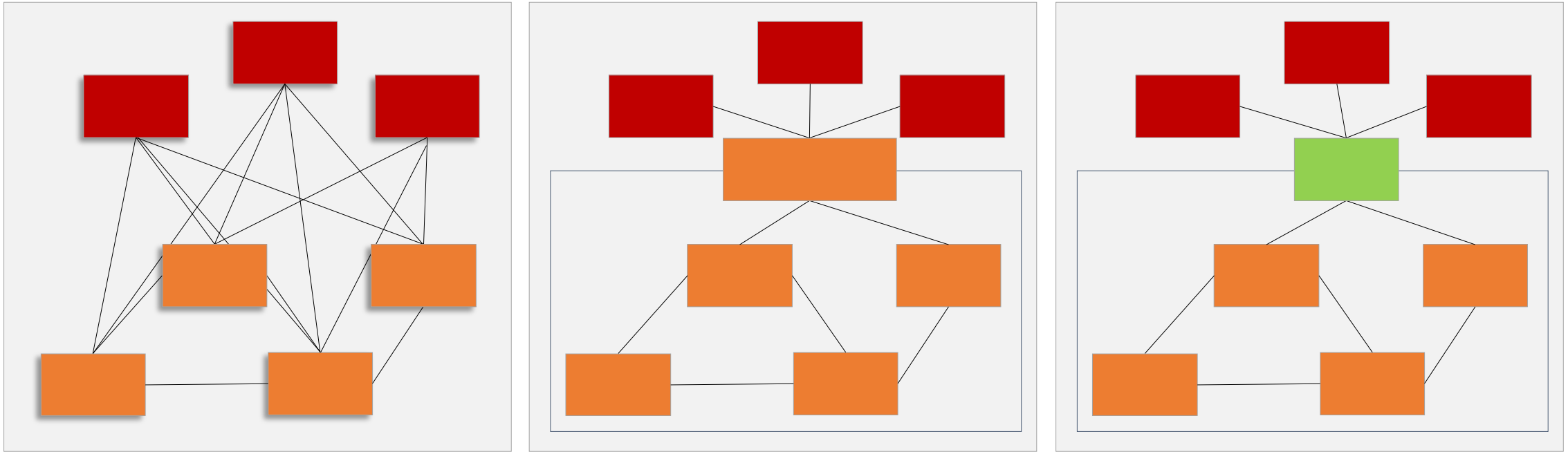
Seam
A seam is a Place where you can alter behavior in your program without editing in that Place.

Enabling Point
Every seam has an enabling point, a Place where you can make the decision to use one behavior or another.



Saxonia Systems
So geht Software.

Echte Abstraktion statt bloßer Indirektion

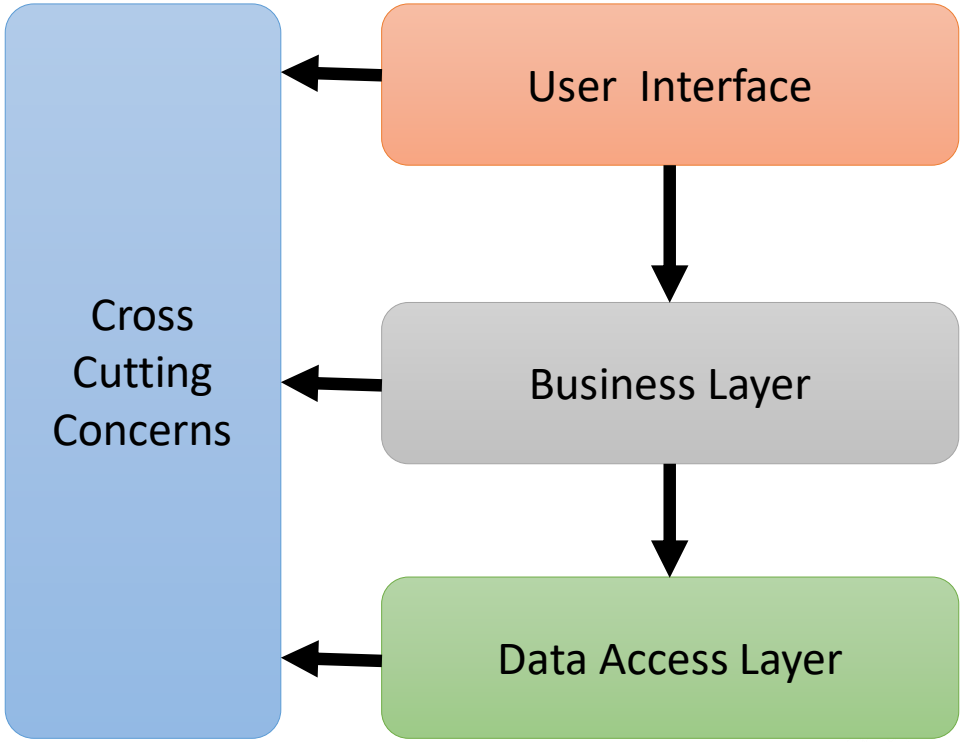


Saxonia Systems
So geht Software.

Code Transformation

Code	<->	Methode
Methode	<->	Methoden
Methode(n)	<->	Klasse
Klasse	<->	Klassen
Klasse(n)	<->	Interface
Interface	<->	Interfaces
Klasse(n)	<->	Namspace(s)
Namspace(s)	<->	Assembly/ies

X-Schichten Modell



DEMO

Abhängigkeiten durchbrechen

```
public class PersonSelectionViewModel
{
        List<Person> Persons { get; set; }
        public void Initialize()
        {
                var query = "SELECT * FROM PEOPLE";
                var connection = new DataBaseConnection();
                var command = connection.CreateCommand(query);
                var reader = command.Excute();

                 this.Persons = new List<Person>();
                while(reader.Next())
                {
                        var person = new Person();
                        person.Id = reader["ID"];
                        person.Name = reader["NAME"];

                         this.Persons.Add(person);
                }
                return;
        }
        . . .
```

Saxonia Systems
So geht Software.

Integrationstest

```
[TestMethod]
public void ViewModelMustShowAllPersons ()
{
    this.SetupDataBase();
    var person = this.AddPersonToDataBase();

    var sut = new PersonSelectionViewModel();
    sut.Initialize();

    this.AssertThatDataBaseContains(person);
}
```

Abhängigkeiten durchbrechen

```
public class PersonSelectionViewModel
{
        List<Person> Persons { get; set; }

        public void Initialize()
        {
            var repository = new PersonRepository();
            this.Persons = repository.GetAllPersons();
            return;
        }

        public void Save(Person person)
        {
            var repository = new PersonRepository();
            repository.Save(person);
            return;
        }

        . . .
}
```

Shims

```
[TestMethod]
public void ViewModelMustShowAllPersons()
{
        using(ShimsContext.Create())
        {
                var person = new Person() { Name = "Meyer" };
                ShimPersonRepository.AllInstances.GetPersons =
                        () => return new List<Person> {person};

                var sut = new PersonSelectionViewModel();
                sut.Initialize();

                Assert.IsTrue(sut.Persons.Contains(person));
        }
}
```

JustMock

```
[TestMethod]
public void ViewModelMustShowAllPersons ()
{
    var person = new Person() { Name = "Meyer" };

    var rep = Mock.Create<PersonRepository>();
    Mock.Arrange(() => rep.Persons).IgnoreInstance()
                                        .Returns(new List<Person> { person });

    var sut = new PersonSelectionViewModel();

    // Initializes the view model
    sut.Initialize();
    Assert.IsTrue(sut.Persons.Contains(person));
}
```

Abhängigkeiten durchbrechen

```
public class PersonSelectionViewModel
{
        List<Person> Persons { get; set; }
        private IPersonRepository repository;

        private IPersonRepository Repository
        {
                get
                {
                        return this.repository ?? (this.repository = new PersonRepository());
                }
        }

        public void Initialize()
        {
                this.Persons = this.repository.GetAllPersons();
                return;
        }

        . . .
}
```

Saxonia Systems
So geht Software.

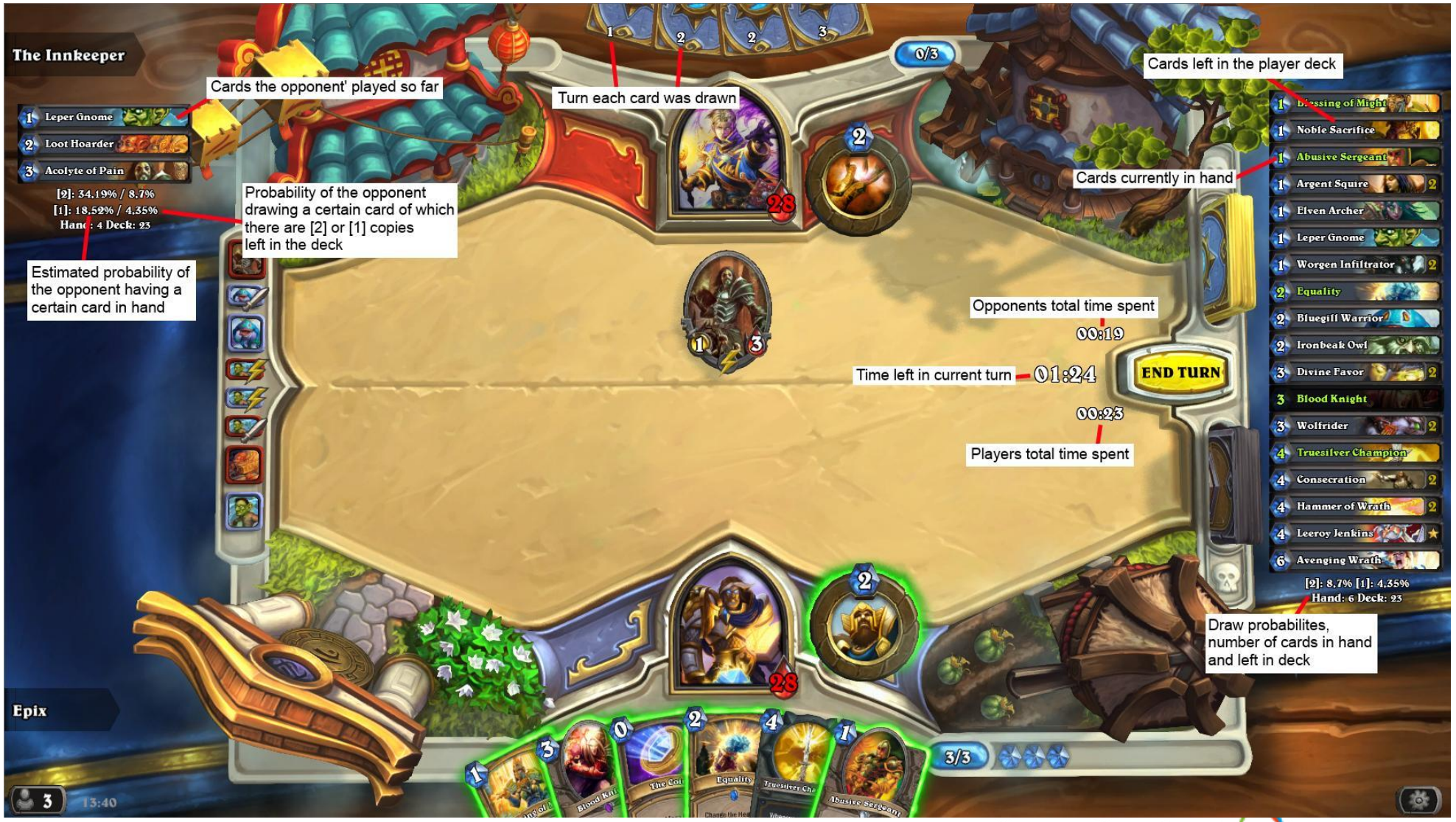
Private Object & FakeItEasy

```
[TestMethod]
public void ViewModelMustShowAllPersons()
{
        var person = new Person() { Name = "Meyer" };

        var repository = A.Fake<IPersonRepository>();
        A.CallTo(repository.GetPersons()).Returns(new List<Person> {person});

        var sut = new PersonSelectionViewModel();

        var accessor = new PrivateObject(sut);
        accessor.SetFieldOrProperty("repository", repository);

        sut.Initialize();

        Assert.IsTrue(sut.Persons.Contains(person));
}
```

Refactoring

REFAKTORISIERUNGSPOTENTIALE AUFDECKEN

Ein Beispiel



Saxonia Systems
So geht Software.

Ein Beispiel

HearthSim / **Hearthstone-Deck-Tracker**

Watch 206 | Star 1,973 | Fork 659

Code | Issues 172 | Pull requests 1 | Wiki | Pulse | Graphs

HDT is an automatic deck tracker and manager for Hearthstone. https://hsdecktracker.net/

2,334 commits | **9** branches | **149** releases | **57** contributors

Branch: **master** | New pull request | New file | Upload files | Find file | HTTPS | https://github.com/Hearth | Download ZIP

Epix37 ArenaRewardsSummary: add WotOG packs — Latest commit 451f8bc an hour ago

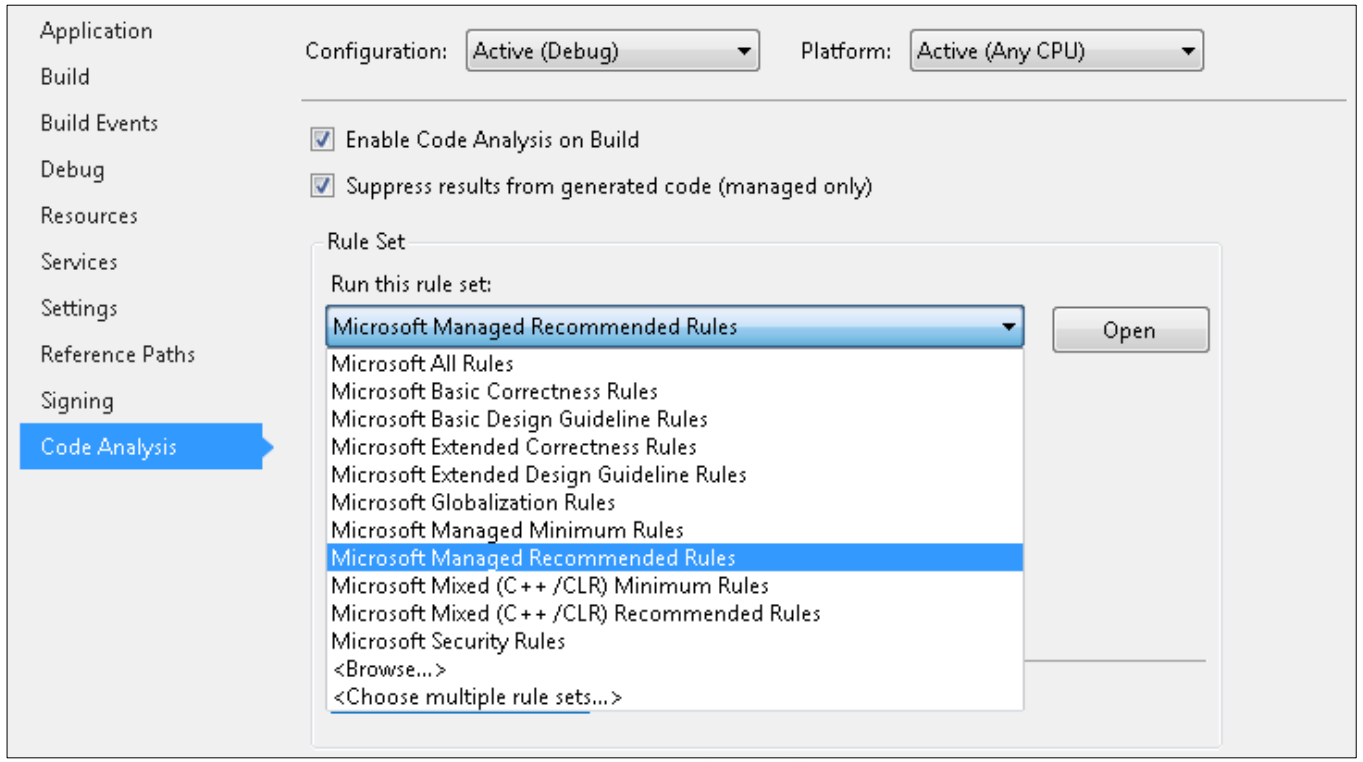
HDTHelper	Explicitly set to C#5 for VS2015 compatibility	6 months ago
HDTTests	update everthing to use HearthDb.Enums	22 hours ago
HDTUninstaller	Explicitly set to C#5 for VS2015 compatibility	6 months ago
HDTUpdate	update Updater to use GitHub API	9 days ago
Hearthstone Deck Tracker	ArenaRewardsSummary: add WotOG packs	an hour ago
lib	update to HearthDb 1.1	3 days ago
licenses	add C'Thun and Yogg-Saron counters	15 hours ago
packages	remove binaries, add 'packages' to .gitignore	2 months ago
raw-assets	update readme screenshots for card themes	23 days ago
.gitignore	remove binaries, add 'packages' to .gitignore	2 months ago
CONTRIBUTING.md	Update CONTRIBUTING.md	a month ago
Hearthstone Deck Tracker.sln	refactoring: extract CardImageBuilder from Card.Background	4 months ago
Hearthstone Deck Tracker.sln.DotSettings	Created V2. Pulled some LogReader bulk into Handlers	8 months ago
README.md	update repo links in readme	2 days ago

FX Cop

Statische Analyse von kompiliertem Code mit Einbindung in den Build

Adressiert:

- Performance
- Sicherheit
- Namensgebung
- Interoperabilität
- …



Saxonia Systems
So geht Software.

Code Metriken ermitteln

Code Metrics Results

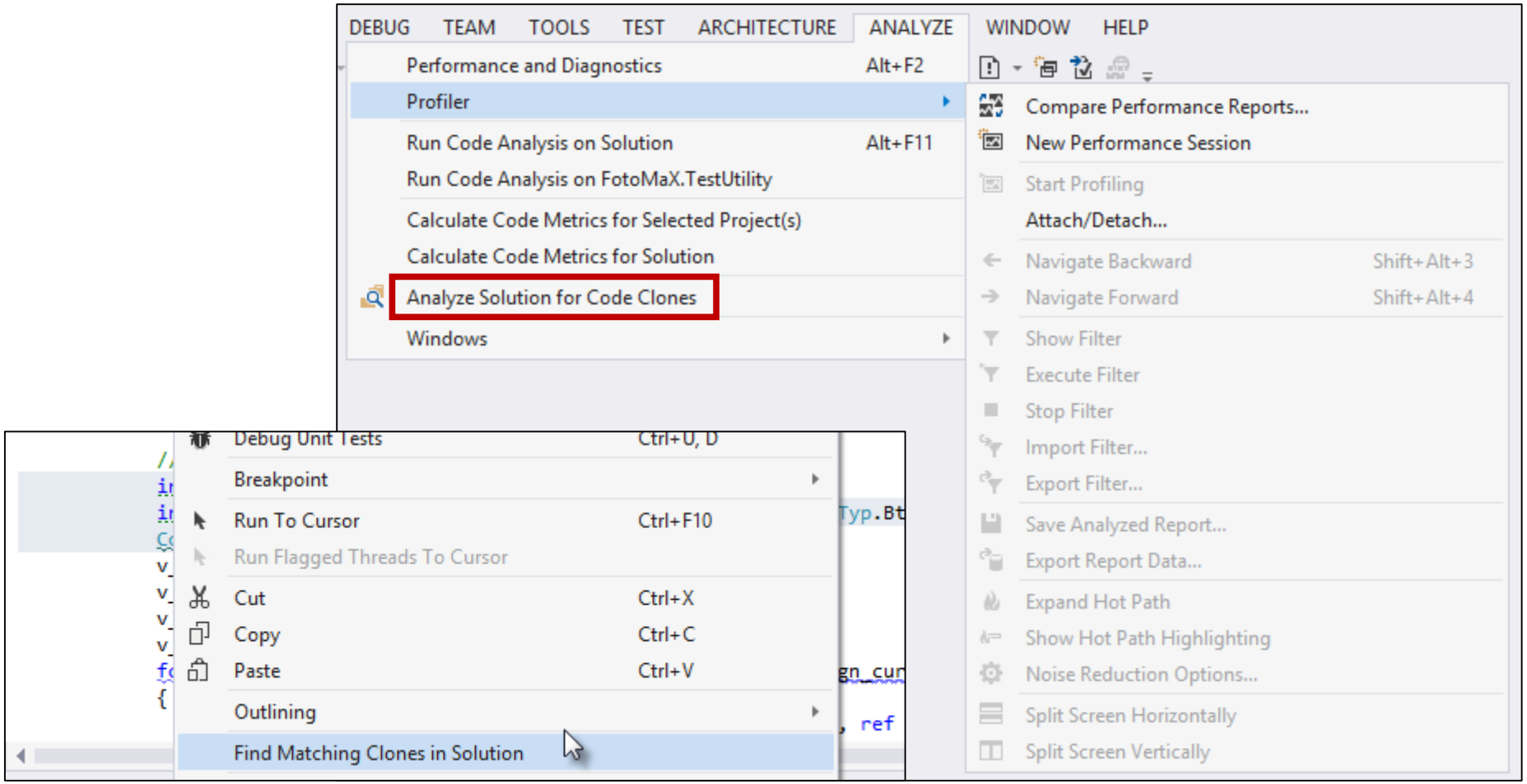
Filter: Cyclomatic Complexity | Min: 0 | Max: <Maximum Value>

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
Tests\FotoMaX.TestUtility (Debug)	90	3	4	5	3
Tests\FotoMaX.OrderProcessing.Tests (Debug)	70	3	1	19	15
Tests\FotoMaX.Infrastructure.Tests (Debug)	71	51	1	97	289
Tests\FotoMaX.ImageSource.Tests (Debug)	72	6	1	30	23
Specification\FotoMaX.Specification (Debug)	92	27	2	14	35
Implementation\OrderProcessing\FotoMaX.OrderProces	94	4	9	6	6
Implementation\Infrastructure\FotoMaX.Start (Debug)	86	9	9	20	22
{ } FotoMaX.Start	86	9	9	20	22
Shell	91	1	9	2	2
Bootst	82	6	3	17	16
App	85	2	3	4	4
Implementati	98	13	3	6	4
Implementati	87	108	9	75	178
Implementati	85	10	9	21	19

Go to Source Code
Copy
Open Selection in Microsoft Excel
Create Work Item
Add/Remove Columns...

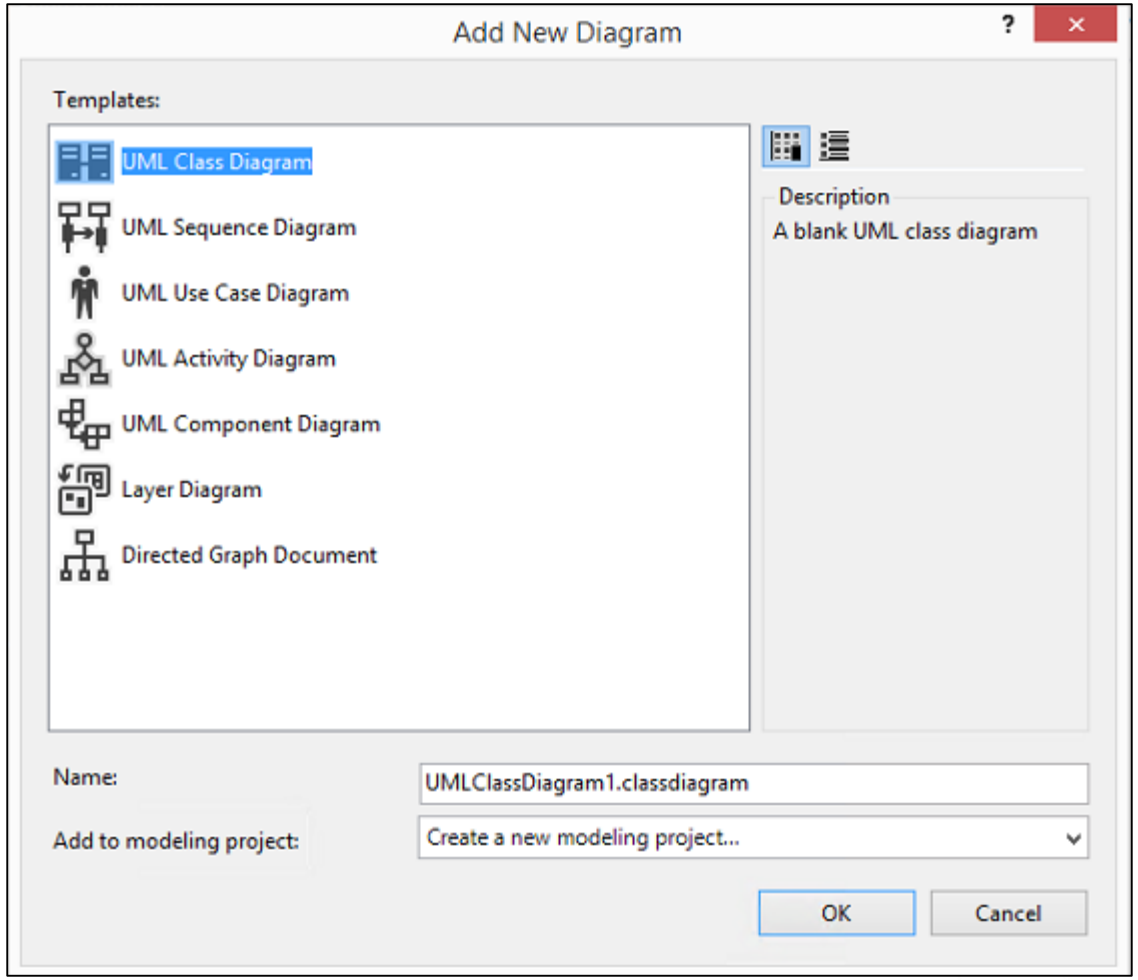
Saxonia Systems
So geht Software.

Code Clones

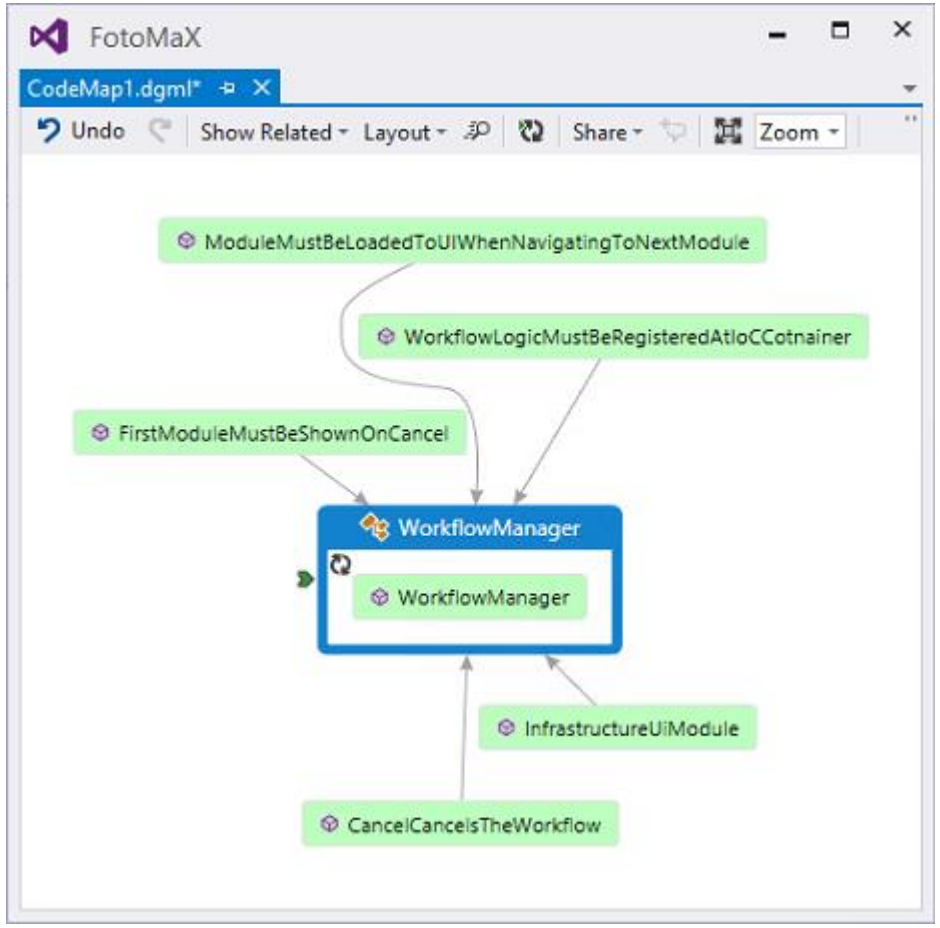


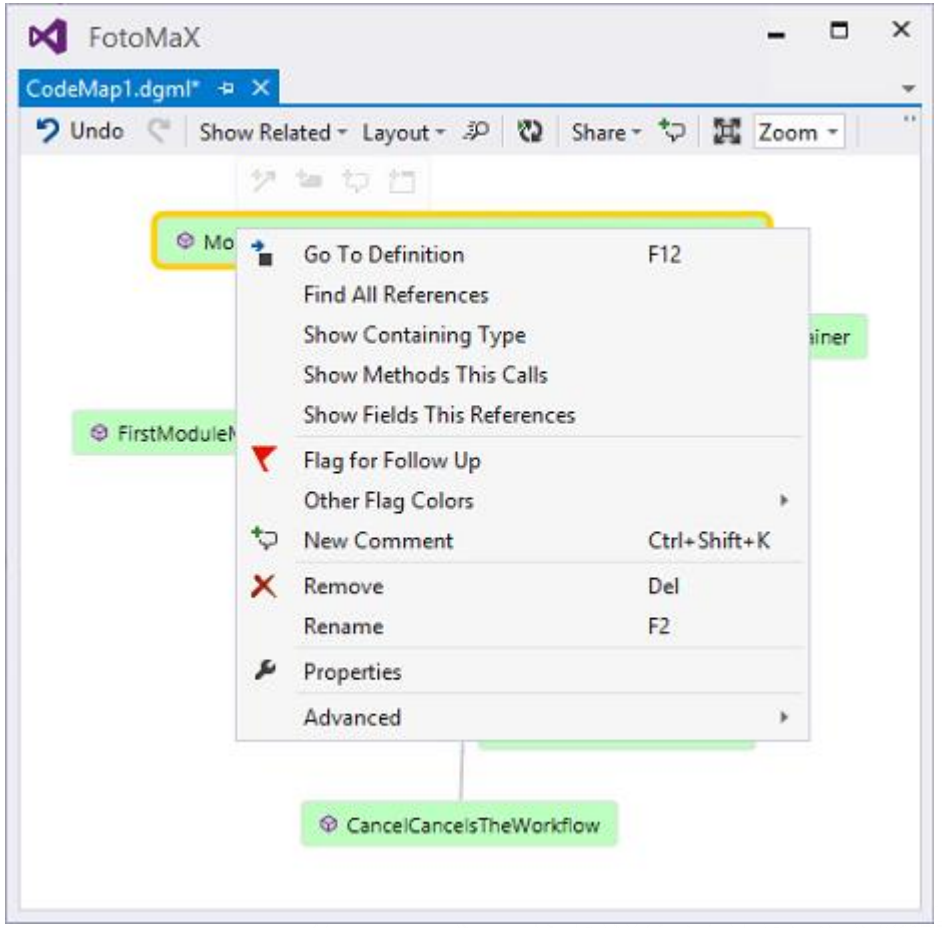
Saxonia Systems
So geht Software.

Architektur



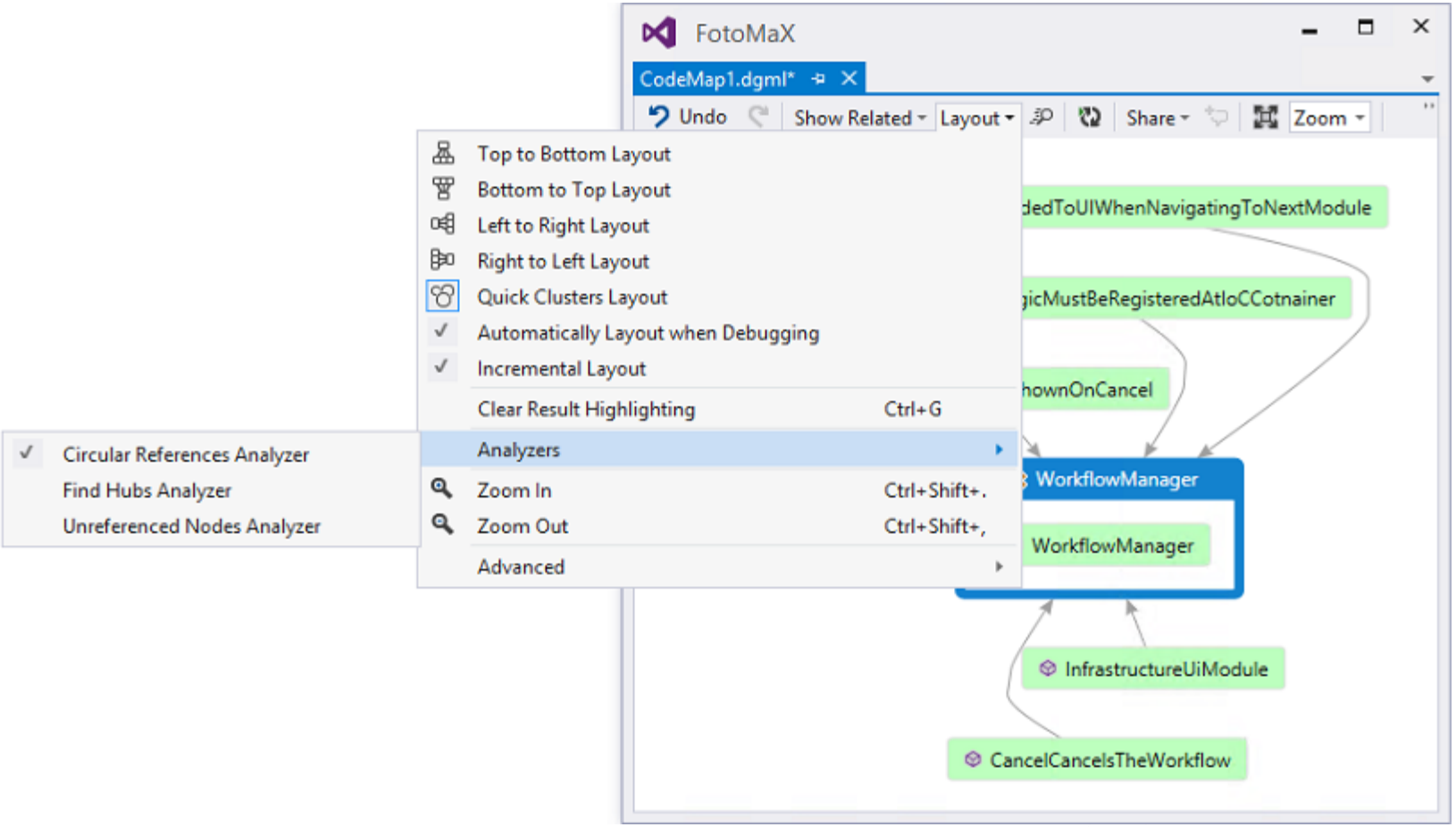
Code Map





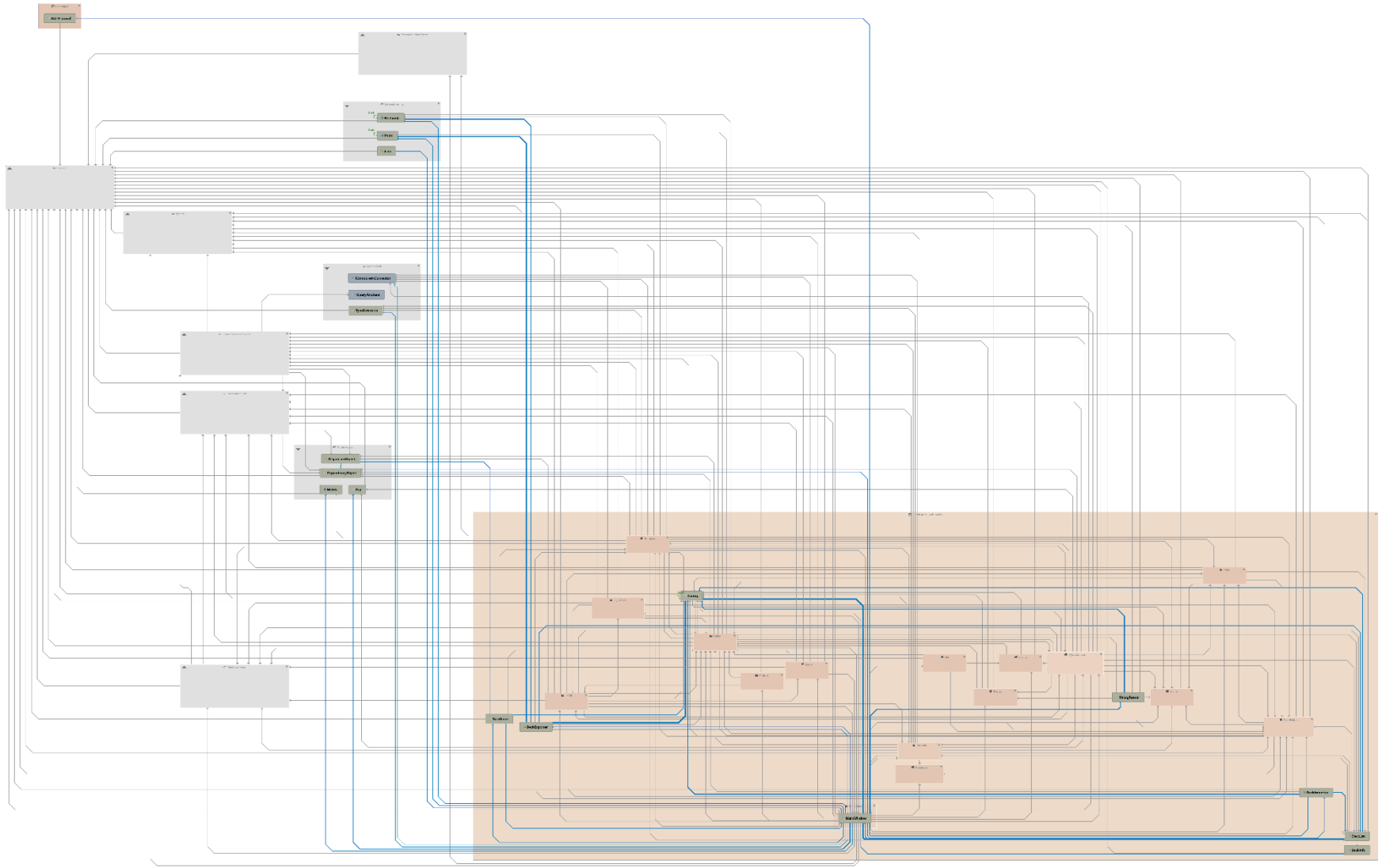
Saxonia Systems
So geht Software.

Code Map



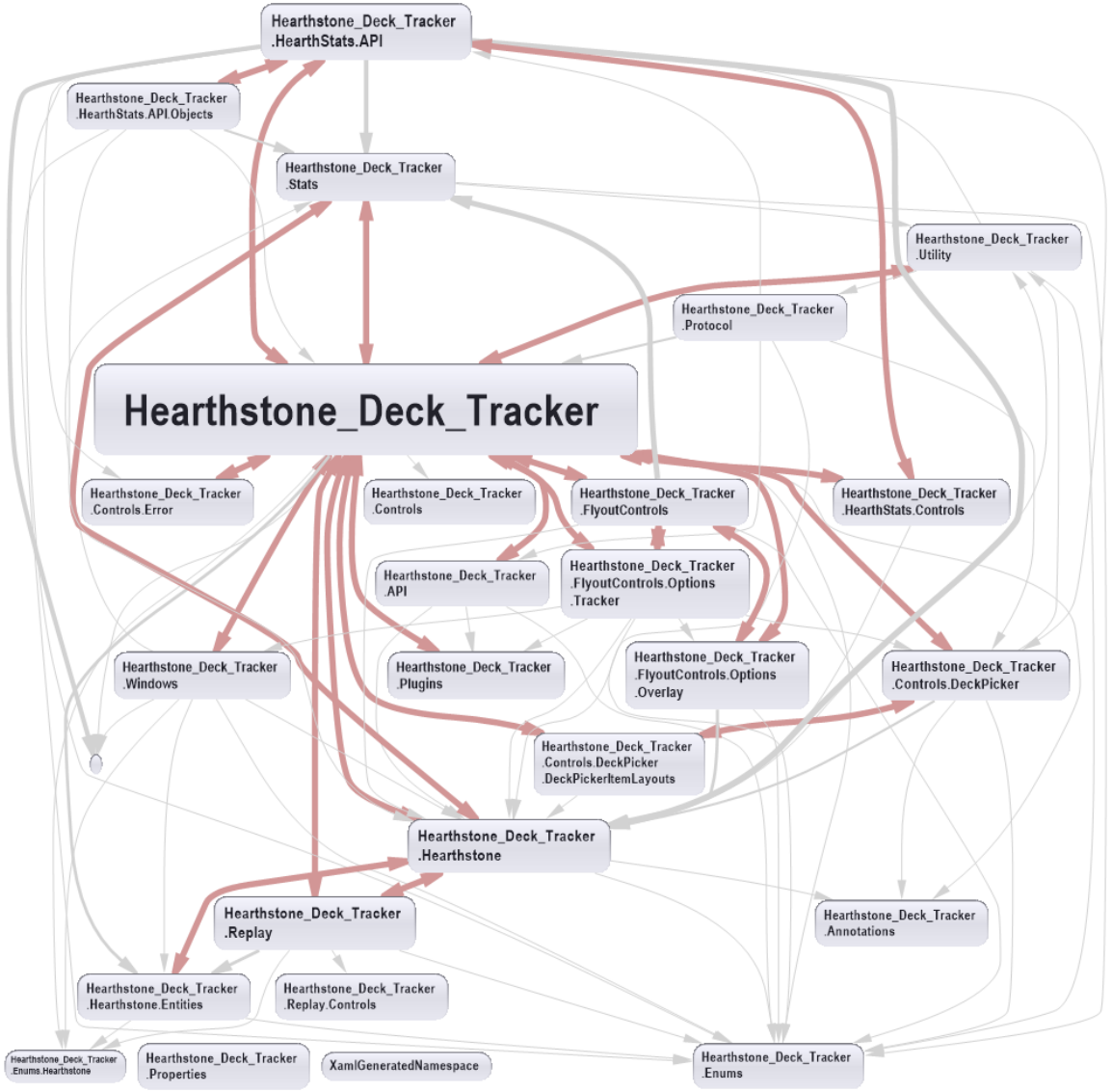
Saxonia Systems
So geht Software.

Abhängigkeitsgraph

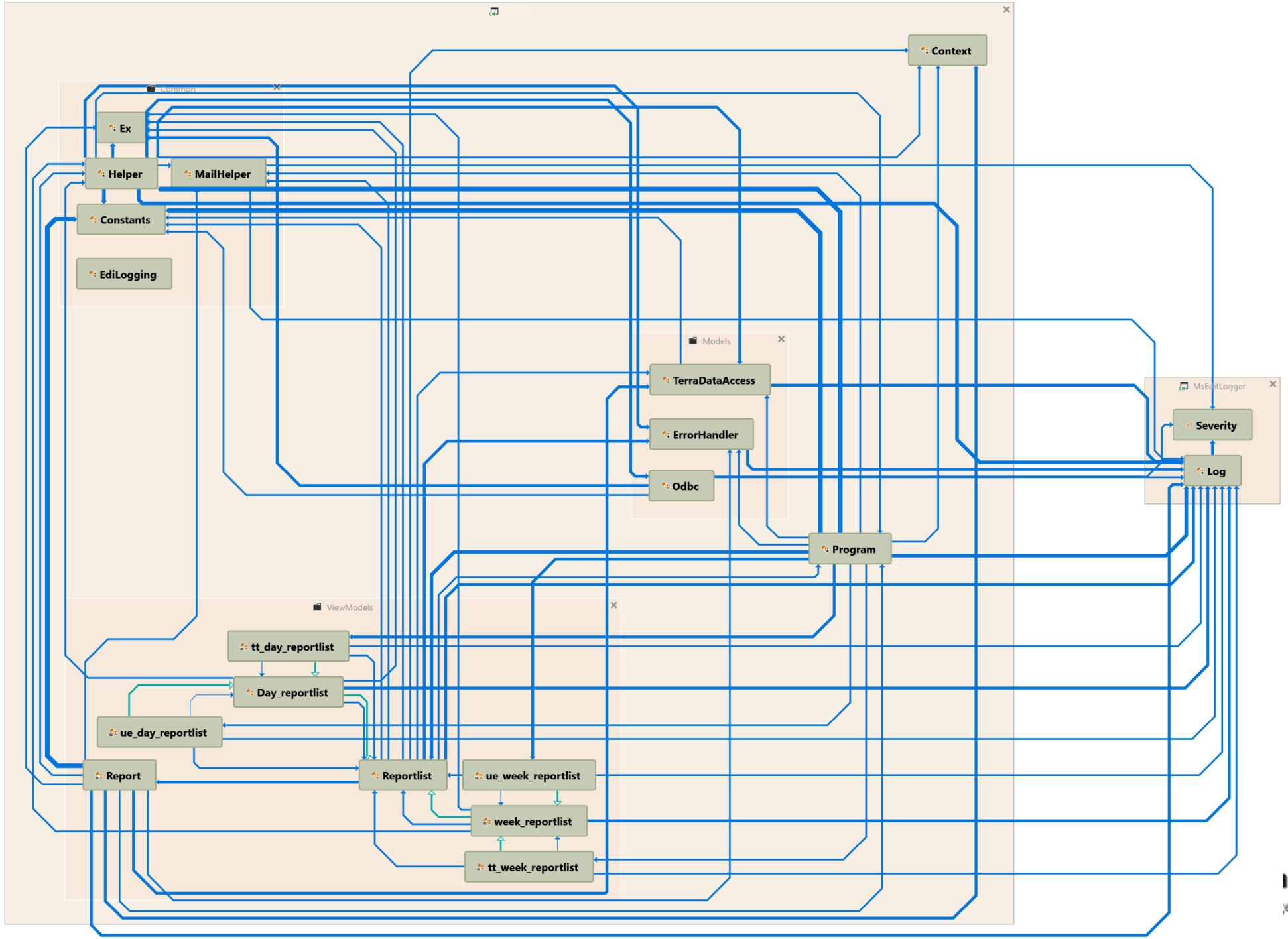


Saxonia Systems
So geht Software.

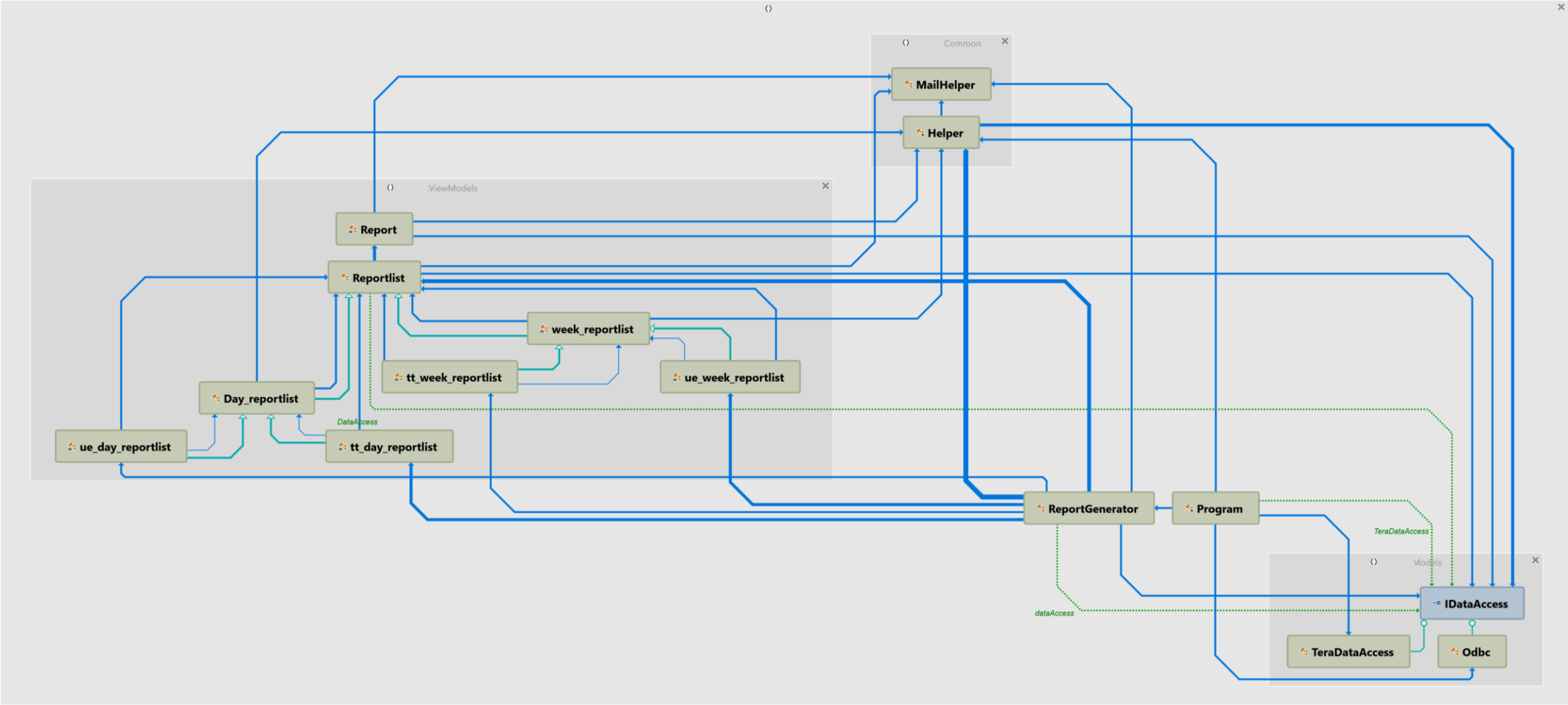
Abhängigkeitsgraph



Saxonia Systems
So geht Software.



Abhängigkeitsgraph



Saxonia Systems
So geht Software.

NDepend

NDepend Project Properties | Dashboard | WorkflowConfigReader.cs | InfrastructureUiModule.cs

Create Trend Chart Charts Range: 1w 2w 1m 2m 3m 6m 1y 2y 5y max

Project Name: **FotoMaX** v1.0.0.0
Project File: C:\Users\hendrik.loesch\Source\Repos\FotoMaX\FotoMaX.ndproj
Analysis Date: **Today 11:06** most recent *Analyzed by NDepend v5.2.1.8320*
Define a Baseline for Comparison

 # Lines of Code
629
285 (NotMyCode)

 # Types
115
10 Assemblies
28 Namespaces
372 Methods
99 Fields
97 Source Files

Comment
54.02%
739 Lines of Comment

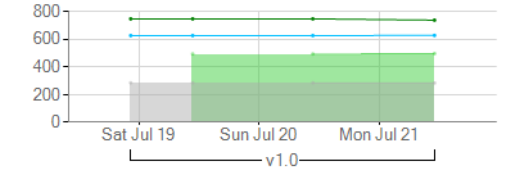
Method Complexity
5 Max
1.2 Average

Code Coverage by Tests
79.17%
498 Lines of Code Covered
131 Lines of Code Not Covered

Third-Party Usage
23 Assemblies used
54 Namespaces used
179 Types used
154 Methods used
4 Fields used

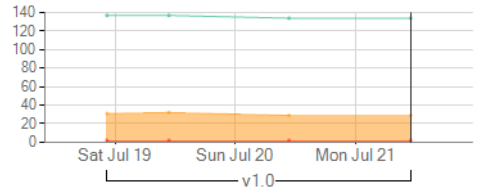
Code Rules
2 Critical Rules Violated
3 Critical Violations
29 Rules Violated (critical or not)
487 Violations
Recent Violations Only
107 Rules Ok
52 Queries
47 Trend Metric Queries
4 Queries Defining JustMyCode
0 Syntax or Execution Error
33 Inactive Rules and Queries

Lines of Code



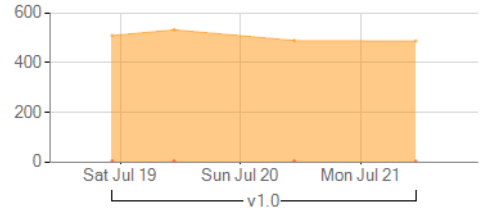
Metric	Value
# Lines of Code	
# Lines of Code Covered	
# Lines of Code (NotMyCo...	
# Lines of Comments	

Rules Violated



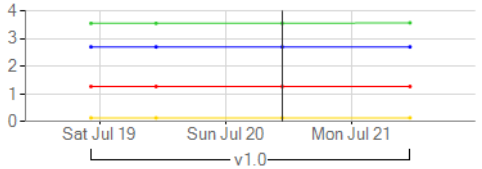
Metric	Value at July 21th 2014 11:06 (v1.0)
# Rules	134 Rules
# Rules Violated	29 Rules
# Critical Rules Violated	2 Rules

Rules Violations



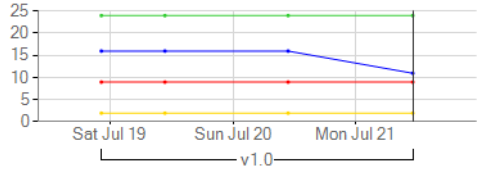
Metric	Value
# Rules Violations	

Average



Metric	Value at July 20th 2014 10:45 (v1.0)
Average IL Cyclomatic Co...	1.28 Paths
Average # Lines of Code f...	2.71 LoC
Average # Methods for Ty...	2.56 Methods

Max



Metric	Value at July 21th 2014 11:06 (v1.0)
Max IL Cyclomatic Comple...	9 Paths
Max # Lines of Code for M...	11 LoC
Max # of Methods for Types	24 Methods

Saxonia Systems
So geht Software.

Cyclomatic Complexity

```
public void ShowModule(IWorkflowModule nextModule)
{
    if (nextModule == null)
    {
        throw new ArgumentNullException("nextModule");
    }

    regionManager.RequestNavigate(Regions.MainRegion, nextModule.Uri);
}
```

Komplexität == 2

Saxonia Systems
So geht Software.

Code Coverage aka Test Coverage

```
private static LifetimeManager CreateLifetimeManager(InstanceConfiguration instanceConfiguration)
{
    LifetimeManager lifetimeManager = null;

    switch (instanceConfiguration)
    {
        case InstanceConfiguration.SingleInstance:
            lifetimeManager = new ContainerControlledLifetimeManager();
            break;
        case InstanceConfiguration.MultipleInstance:
            lifetimeManager = new PerResolveLifetimeManager();
            break;
        default:
            throw new ArgumentOutOfRangeException("instanceConfiguration");
    }

    return lifetimeManager;
}
```

Komplexität == 3

Saxonia Systems
So geht Software.

C.R.A.P.

$$C.R.A.P.(m) = CC(m)^2 * (1 - Coverage(m)/100)^3 + CC(m)$$

"The C.R.A.P. (Change Risk Analysis and Predictions) index is designed to analyze and predict the amount of effort, pain, and time required to maintain an existing body of code."

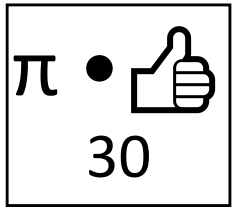
Alberto Savoia

Saxonia Systems
So geht Software.

Quelle: http://www.artima.com/weblogs/viewpost.jsp?thread=210575

C.R.A.P.

$$C.R.A.P.(m) = CC(m)^2 * (1 - Coverage(m)/100)^3 + CC(m)$$



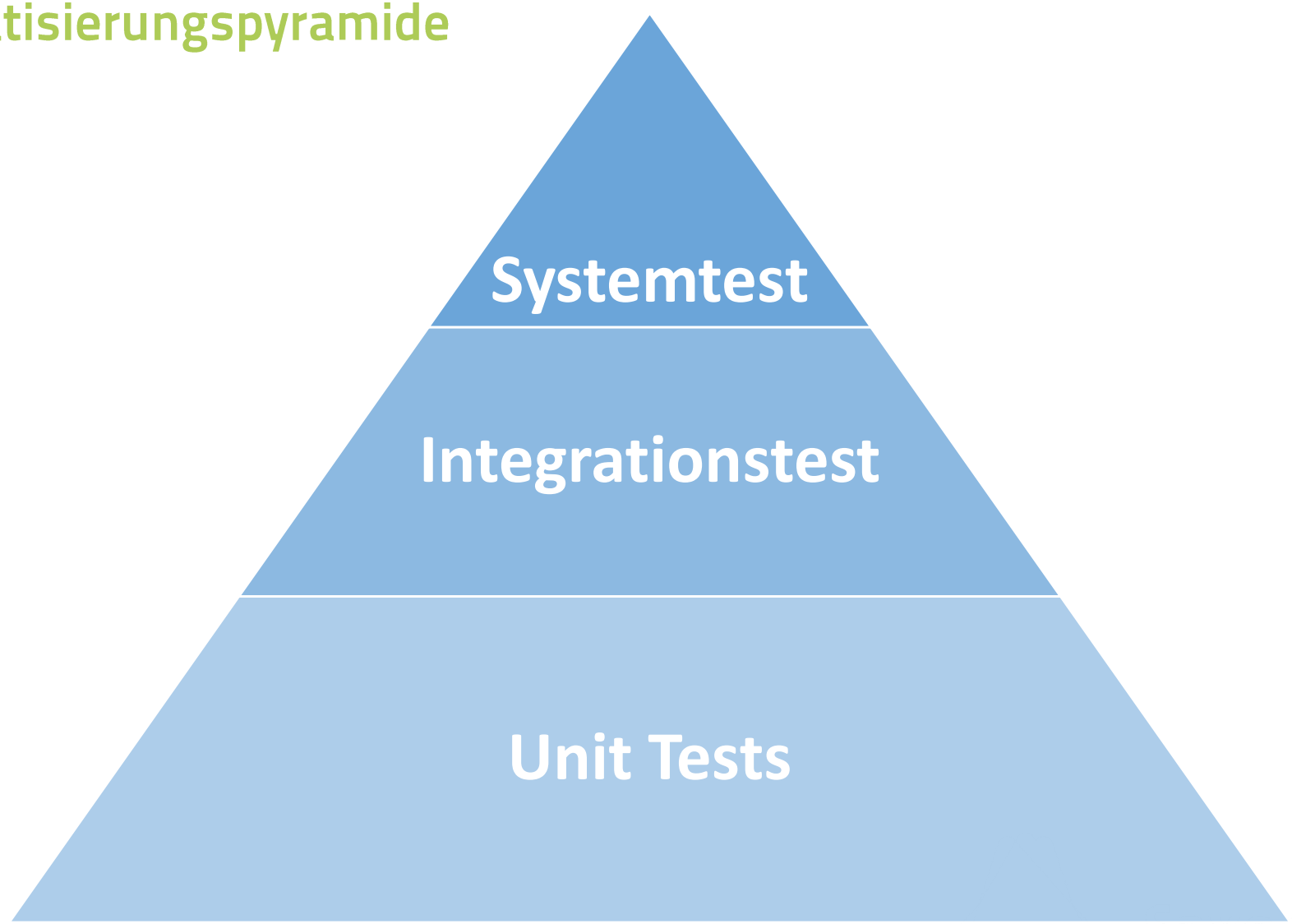
Method's CC	% of coverage required to be below CRAPpy threshold
0 – 5	0%
10	42%
15	57%
20	71%
25	80%
30	100%
31+	No amount of testing will keep methods this complex out of CRAP territory.

Saxonia Systems
So geht Software.

Quelle: https://www.artima.com/weblogs/viewpost.jsp?thread=215899

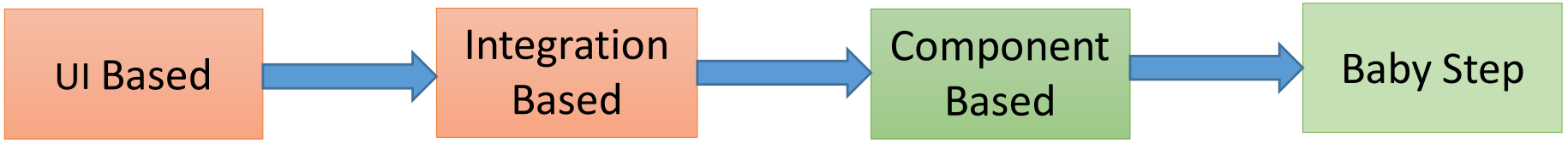
FAZIT

Testautomatisierungspyramide

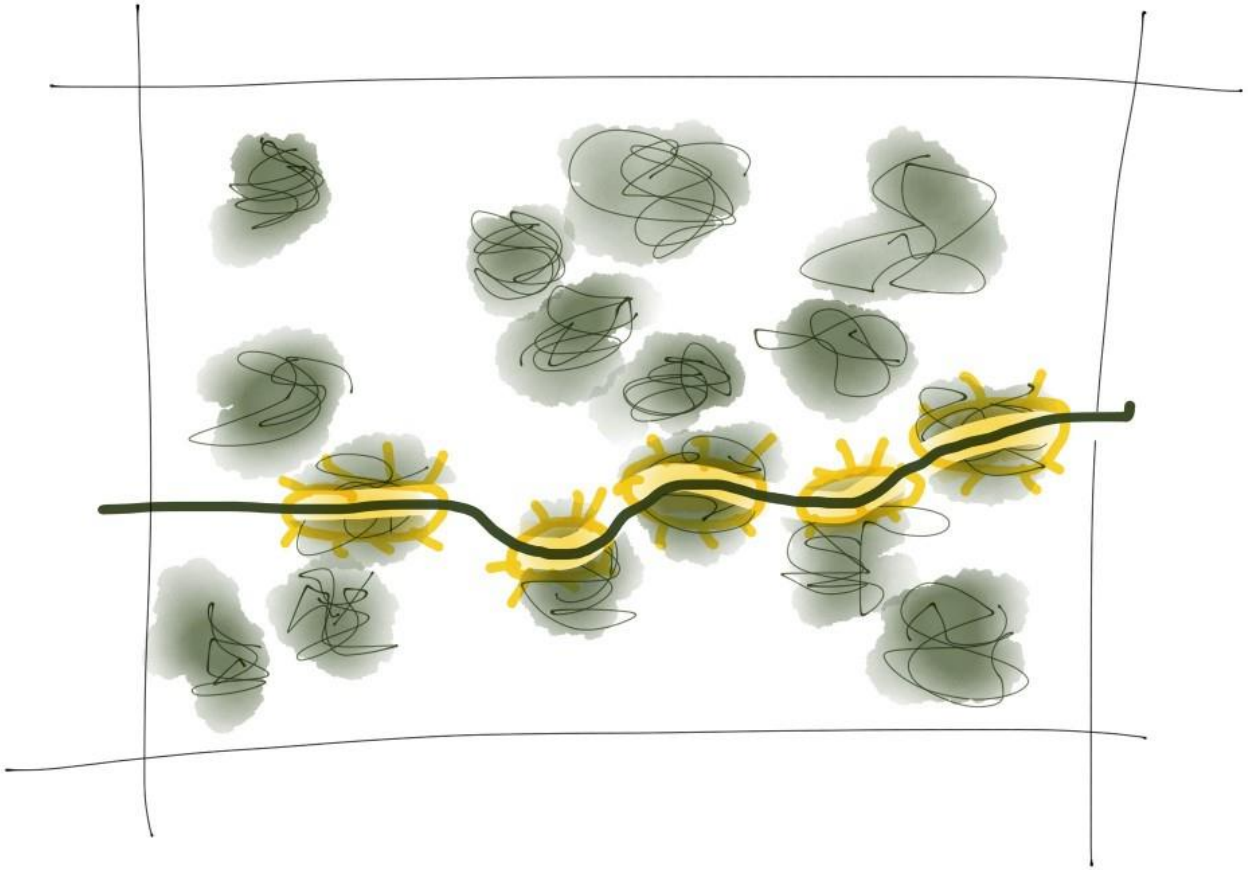


Saxonia Systems
So geht Software.

Testvorgehen



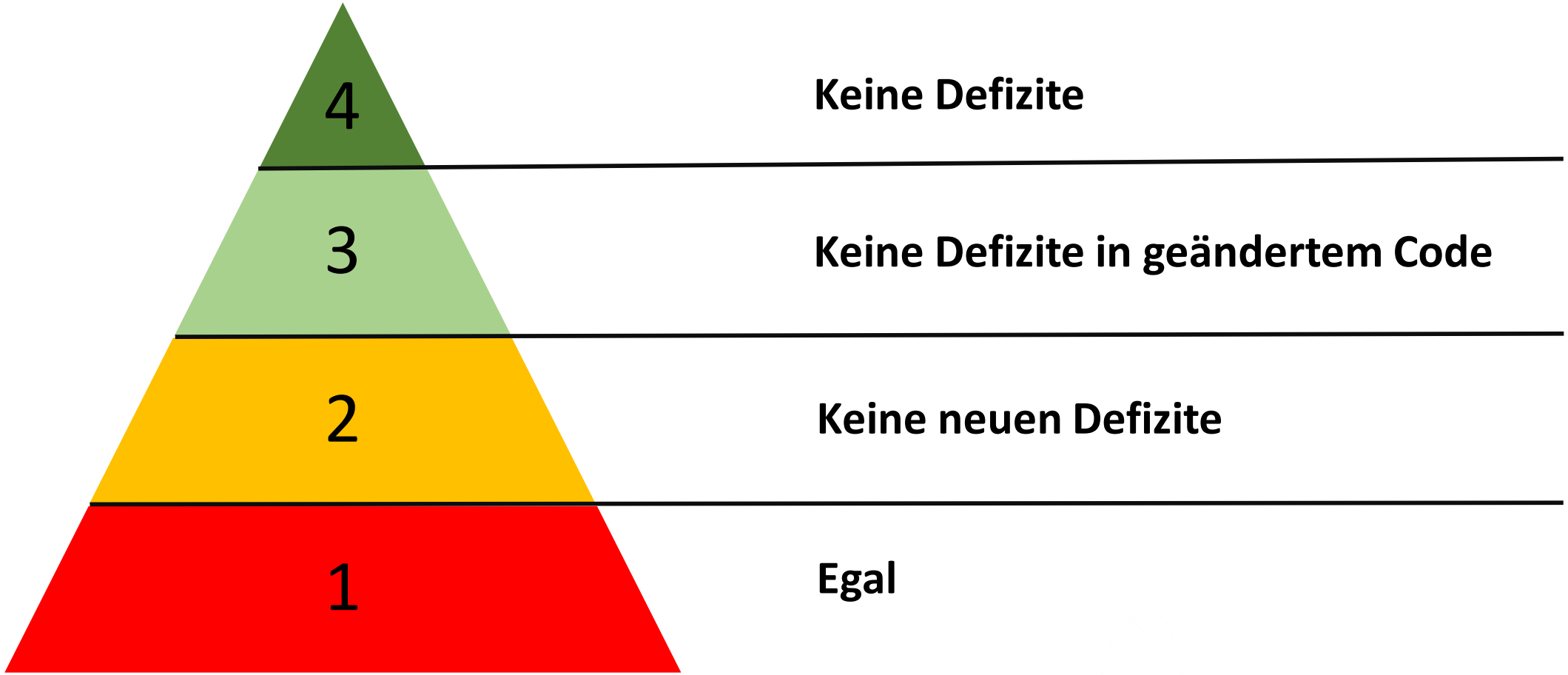
Refactoring



Quelle: Ron Jeffries http://xprogramming.com/articles/refactoring-not-on-the-backlog

Saxonia Systems
So geht Software.

Wieviel ist gut für mich?



Quelle: Dr. Elmar Jürgens, CQSE

Saxonia Systems
So geht Software.

Pragmatismus vs. Overengineering

Pragmatismus

Overengineering

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.