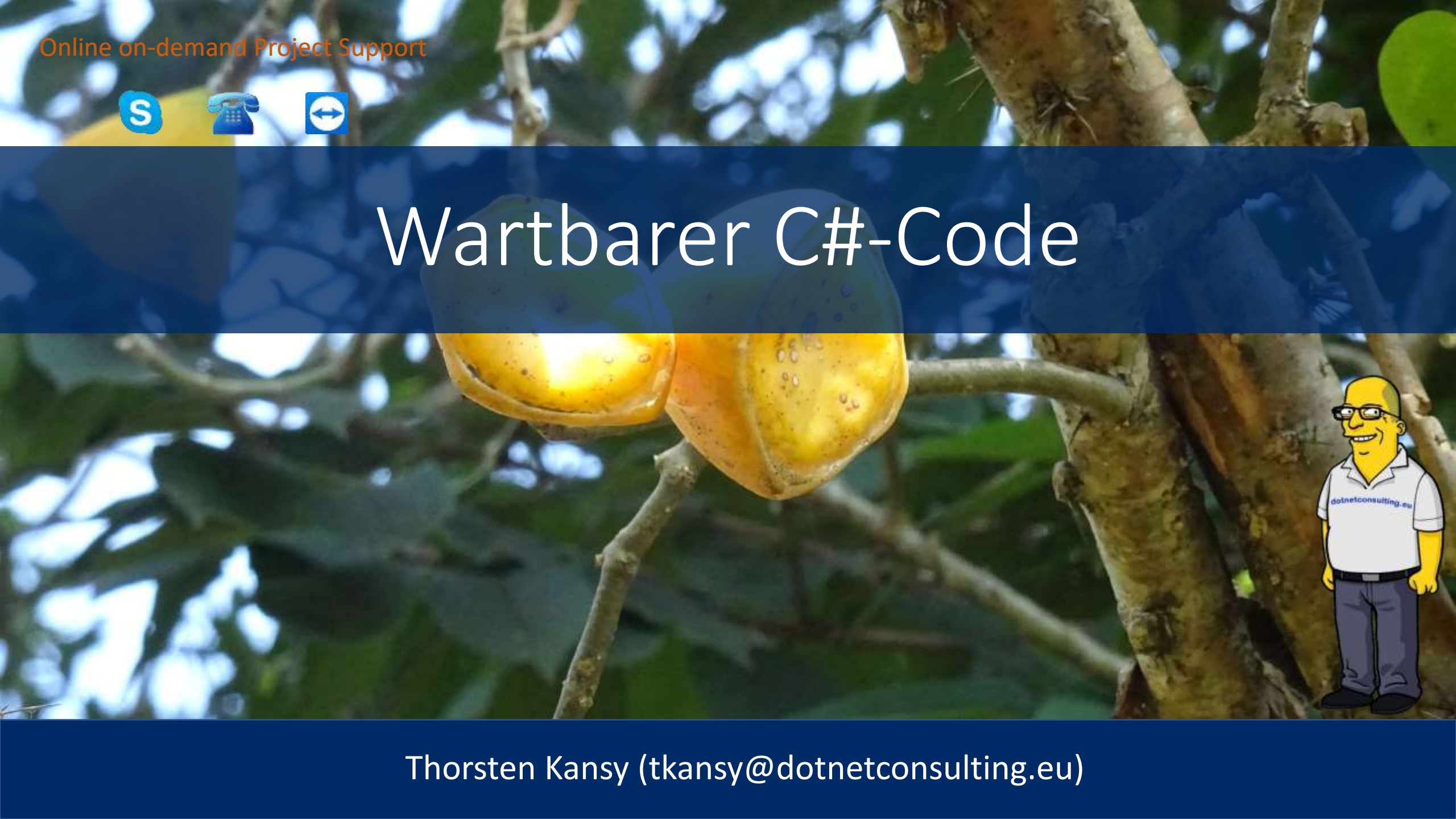



# Wartbarer C#-Code

Thorsten Kansy (tkansy@dotnetconsulting.eu)

# Thorsten Kansy

Freier Consultant, Software Architekt,
Entwickler, Trainer & Fachautor

DOTNETCONSULTING
by Thorsten Kansy

# Agenda

- Warum (gut) wartbaren Code?
- Code Metrics
- Code-Analyse
- SOLID-Prinzipien
- DRY-Prinzip
- KISS-Prinzip
- Naming-Conventions
- Refactoring

- Sonstiges

# Warum (gut) wartbaren Code?

# Warum ist wartbarer Code wichtig?

- Schlecht wartbarer Code lenkt ab
- Verursacht höhere Kosten
  - Fehlersuche komplexer
  - Einarbeitung aufwendiger
  - Anpassung aufwendiger
  - …
- Lohnt sich also eine Quick & Dirty-Lösung?

# Was ist eigentlich wartbarer Code?

- Anpassungen sind mit vertretbarem Auswand möglich
- Fehler können (leichter) gefunden und behoben werden

- Verständnis des Codes
- Einarbeitung neuer Mitarbeiter

- Refactoring freundlich

# Was ist schlechter/ nicht wartbarer Code?

- Das Gegenteil von wartbarem Code
  - Over engineered
  - Under engineered

„Funktioniert doch“ ist kein Argument, sondern oftmals eine Ausrede!

# Grundlegendes

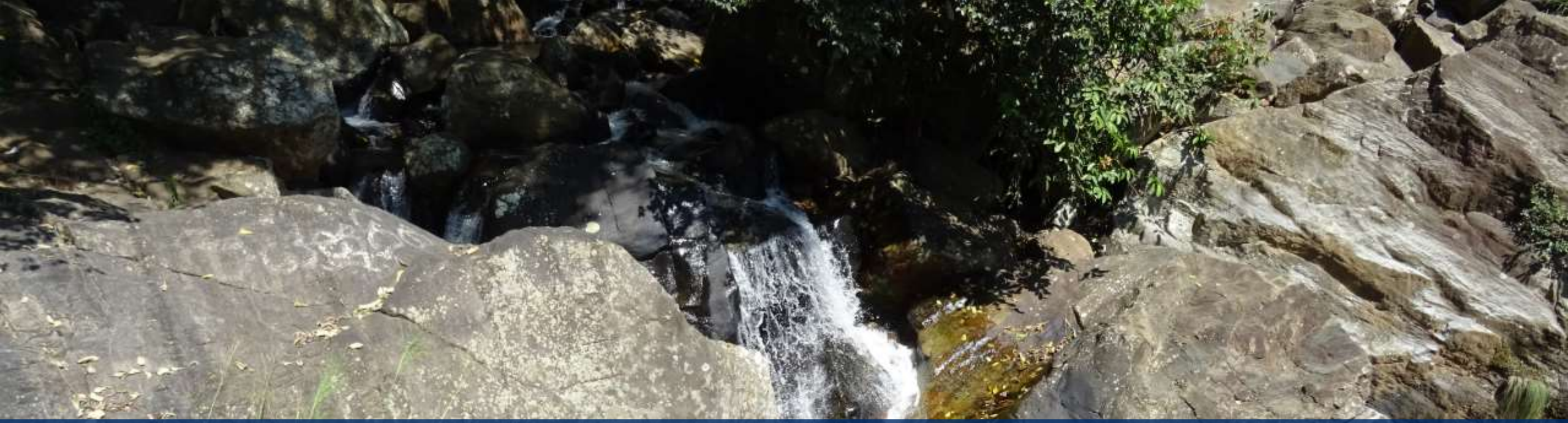
- Wartbarer Code
  - ist nicht binär
  - Entsteht in einem fortwährenden Prozess
    - Erstellung
    - Erhalt

- Sinnvolle(!) Architektur nach Design-Prinzipien hilft
  - SOLID-Prinzipien
  - DRY-Prinzip
  - KISS-Prinzip

# Meßbarkeit?

- Code Metrics

- Compiler Warnungen existieren nicht ohne Grund
  - Beheben!

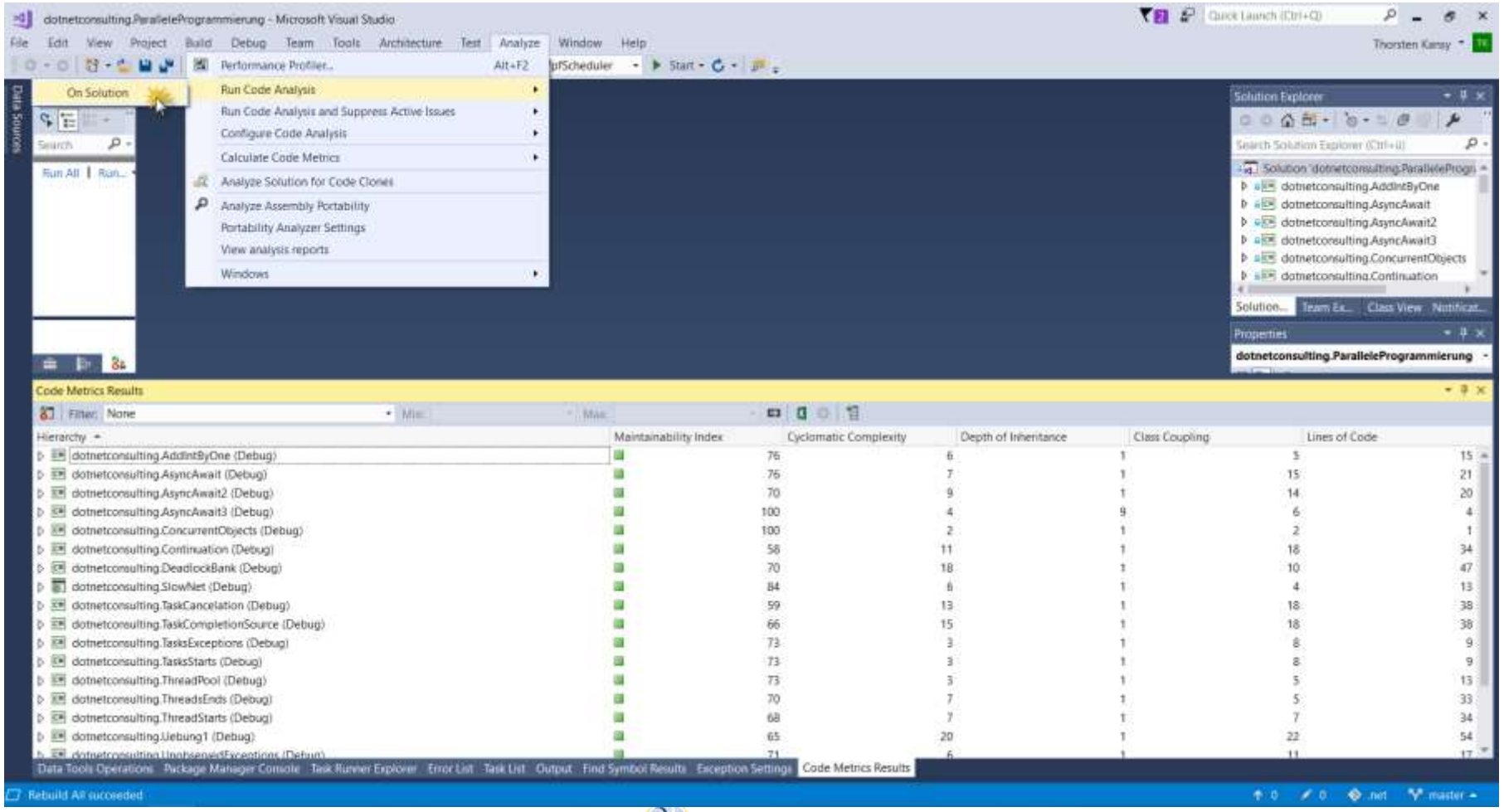


# Code Metrics

# Code Metrics

- Maintainability Index (>)
  - Gibt einen Wert an, wie gut der Code wartbar ist
- Cyclomatic Complexity (<)
  - Anzahl der Code paths
- Depth of Inheritance (<)
  - Tiefe der Vererbung
- Class Coupling (<)
  - Verflechtung von Typen/ Klassen
- Lines of Code (<)
  - Codezeilen (IL-Code)

# Code Metrics im Visual Studio



# Demo

# Wie die Wartbarkeit erhöhen?

- Was also tun, wenn die Wartbarkeit (zu) niedrig ist?

- In leichten-normalen Fällen
  - Pfadfinder Prinzip

- In schwereren Fällen
  - Umfängliches Redesign
  - Technische Schuld berücksichtigen

# Code Analyse

# Code Analyse

- Visual Studio schlägt Verbesserungen am Code vor
  - IDExxxx als Meldung

```
static private void IDE0028()
{
    IList<int> numbers = new List<int>();
    numbers.Add(createNumber());

    static int createNumber()...
}
```

```
static private void IDE0028()
{
    IList<int> numbers = new List<int>
    {
        createNumber()
    };

    static int createNumber()...
}
```

Nur bedingt ungesehen eine gute Idee!

# SOLID-Prinzipien

# SOLID-Prinzipien

- **S**ingle responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

# Single responsibility principle

„Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern.“
(Robert C. Martin)

Eine funktionale Einheit (Klasse, Funktion, Variable, etc.) darf nur für eine Aufgabe verantwortlich sein.

R.I.P Eierlegende Wollmilchsau

# Open–closed principle

„Module sollten sowohl offen (für Erweiterungen) als auch verschlossen (für Modifikationen) sein.“

(Bertrand Meyer)

Keinen (externen) Zugriff auf Interna einer Funktionalität

Schlüsselwort `sealed` selten verwenden.

# Liskov substitution principle

- Ein Funktion, die mit einem Type arbeitet, sollte auch mit einem Untertyp arbeiten.

- In C# (automatisch) eingebaut durch Vererbung und Polymorphismus
- Eine Aushebelung via Code ist aber denkbar, aber keine gute Idee

# Interface segregation principle

„Schnittstellen sollte nur so umfangreich sein, wie nötig.“

- C#-Interfaces
- C#-Klassen

Notfalls aufteilen

# Dependency inversion principle

„Module höherer Ebenen sollten nicht von Modulen niedrigerer Ebenen abhängen.“ (Wikipedia)

Abstraktionen durch

- Schnittstellen
- Abstrakte (Basis-)Klassen

# DRY-Prinzip

# DRY (Don‘t repeat yourself)

- Umfangreichere Funktionalitäten
  - Nuget
- Einfache Funktionalitäten
  - Bibliothek
- Aber um jeden Preis?
  - Wo ist die (Komplexitäts-)Grenze?
  - Einzelne Komponenten müssen so konzipiert sein, das sie möglichst oft eingesetzt werden können
  - Dürfen aber wiederrum nicht zu generisch werden
- Keine vorhandenen Funktionen nachbauen

# KISS-Prinzip

# KISS-Prinzip

- KISS-Prinzip
  - Keep it simple, stupid

- Don't over-engineer
  - Wartbaren Code schreiben, nicht zeigen was für ein toller Entwickler man so ist
  - Inneren Schweinehund besiegen

# Naming-Conventions

# Helfen Naming-Konventionen?

- Vermutlich schon
  - Wenn es die richtigen sind ;-)

- Einheitliche Namensgebung
  - Create- erstellt (z.B. `CreateAddress()`)
  - Calculate- berechnet (z.B. `CalculateInvoice()`)
  - Refresh- aktualisiert (z.B. `RefreshToken()`)

# Was ist mit Denglish?

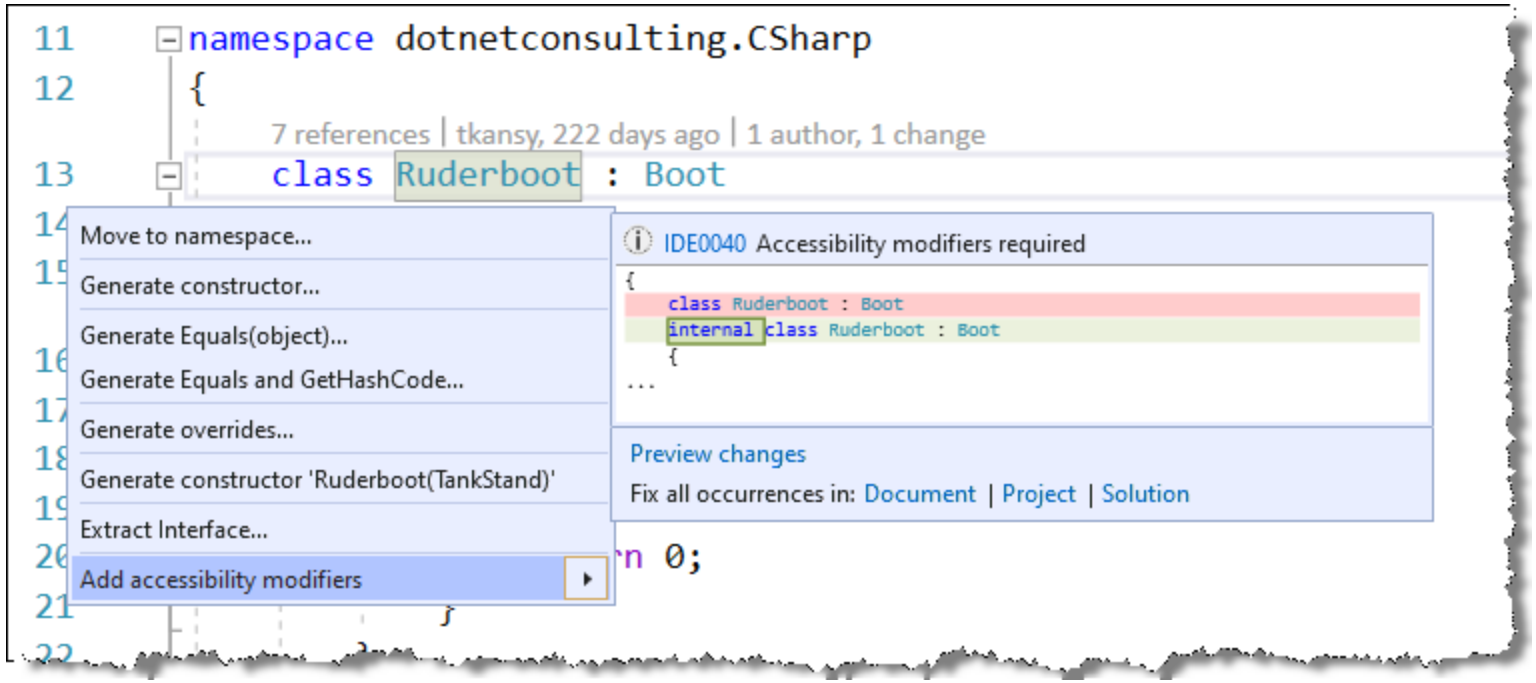
- `GetKunde()`
- `GetCustomer()`

aber

- `GetAbsetzbaresSondergut()`
- `GetDeductibleSeparateEstates()`

# Refactoring

# C# & Visual Studio Refactoring




- `nameof`-Operator
  - Umbenennung!

# Demo

# Sonstiges

# Sonstiges

- var vs explizite Datentypen
- No Magic Numbers
- (angemessene) Kommentare nicht vergessen
- Spaghetti Code vermeiden
- Jeder Code ohne Sinn ist überflüssig
  - Wirklich? Wirklich!
  - Code nach Abschluss aufräumen, unnötiges Entfernen

Es gibt Git, SVN & Co.

# Keine Kommentare nur weil Sie grün sind

```
//protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
//{
//    if (!optionsBuilder.IsConfigured)
//    {
//        // Konfiguration auf Azure Comos DB Emulator
//        string endPoint = @"https://localhost:8081";
//        string authKey = @"C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyP
//        string databaseName = @"EFDatabaseOrders";
//        optionsBuilder
//            .EnableSensitiveDataLogging(true)
//            .ConfigureWarnings(c =>
//            {
//                // c.Throw(???);
//            })
//            .UseCosmos(endPoint, authKey, databaseName);
//    }

//    base.OnConfiguring(optionsBuilder);
//}

//protected override void OnModelCreating(ModelBuilder mb)
//{
//    mb.Entity<Order>(e =>
//    {
//        e.ToContainer("Orders");
//        // e.Metadata.Cosmos().ContainerName = "Orders";
```

# No Magic Numbers

Verschleitert die Bedeutung

Besser Konstanten mit guten Namen

```
static void ProcessData(Data data)
{
    if (data.status == 451)
    {
        // ...
    }
}
```

# Fragen?

# Links

www.dotnetconsulting.eu

@Tkansy

tkansy@dotnetconsulting.eu

DOTNETCONSULTING
by Thorsten Kansy