DHBW

Online-Library Metrics

Version <1.0>

Online-Library	Version: <1.0>	
Metrics	Date: 07.05.2014	

Revision History

Date	Version	Description	Author
07.05.2014	<1.0>	Metrics	Torsten, Wanping

Online-Library	Version: <1.0>
Metrics	Date: 07.05.2014

Table of Contents

1.	Metri	ics	4
	1.1	Description	4
2.	Wart	barkeitsindex	4
	2.1	Description	4
	2.2	Examples	4
3.	Zyklo	omatische Komplexität	4
	3.1	Descripton	4
	3.2	Examples	4
4.	Scree	enshot	6

Online-Library	Version: <1.0>
Metrics	Date: 07.05.2014

Metrics

1. Metrics

1.1 Description

Metrik in der Software Entwicklung ist eine Funktion, die eine Eigenschaft von Software in einen Zahlenwert, auch Maßzahl genannt, abbildet. Hierdurch werden formale Vergleichs- und Bewertungsmöglichkeiten geschaffen.

2. Wartbarkeitsindex

2.1 Description

Berechnet einen relativen Indexwert zwischen 0 und 100, der angibt, wie einfach der Code zu verwalten ist. Ein hoher Wert steht für bessere Verwaltbarkeit. Mit farbcodierten Bewertungen können problematische Stellen im Code schnell ermittelt werden. Eine grüne Bewertung liegt zwischen 20 und 100 und gibt an, dass der Code über eine gute Wartbarkeit verfügt. Eine gelbe Bewertung liegt zwischen 10 und 19 und gibt an, dass der Code über eine mäßige Wartbarkeit verfügt. Eine rote Bewertung liegt zwischen 0 und 9 und gibt eine niedrige Wartbarkeit an.

2.2 Examples

Hoher Wartbarkeitsindex:

```
public class Genre
{
   public int GenreId { get; set; }
   public string Typ { get; set; }
   public string Name { get; set; }
   public virtual ICollection<Titel> Titels { get; set; }
}
```

Diese Klasse besteht nur aus getter- und setter-Methoden. Deshalb kann sie einfach gewartet werden.

```
Mittelwertiger Wartbarkeitsindex:
public void DeleteUser(string UserName)
{
    if (UserExists(UserName))
    {
        db.Users.Remove(db.Users.Where(u => u.UserName == UserName).Single());
    }
}
```

Diese Klasse hat nur eine mittelmäßige Wartbarkeit, da der if-Befehl aus einer Verkettung von mehreren Zugriffen besteht.

3. Zyklomatische Komplexität

3.1 Description

Misst die strukturelle Komplexität des Codes. Sie wird durch Berechnung der Anzahl unterschiedlicher Codepfade im Programmfluss erstellt. Für ein Programm mit komplexer Ablaufsteuerung sind mehr Komponententests erforderlich, wenn eine gute Codeabdeckung erzielt werden soll, zudem verschlechtert sich die Verwaltbarkeit.

3.2 Examples

Niedrige zyklomatische Komplexität:

Online-Library	Version: <1.0>
Metrics	Date: 07.05.2014

```
public void DeleteUser(string UserName)
{
    if (UserExists(UserName))
    {
        db.Users.Remove(db.Users.Where(u => u.UserName == UserName).Single());
    }
}
```

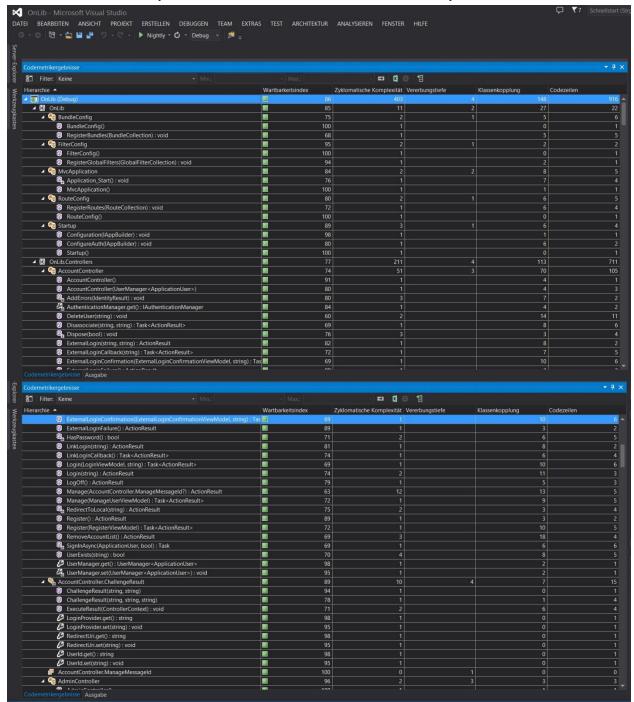
Diese Klasse hat eine niedrige Anzahl an Codepfade im Programmfluss, deshalb ist die zyklomatische Komplexität dieser Funktion sehr niedrig.

```
Relativ hohe zyklomatische Komplexität:
public bool Exists(string name, string vorname)
{
    foreach (var item in db.Autors)
    {
        if (item.Nachname == name)
        {
            if (String.IsNullOrEmpty(vorname))
            {
                return true;
        }
        else if (item.Vorname == vorname)
        {
                return true;
        }
    }
    return false;
}
```

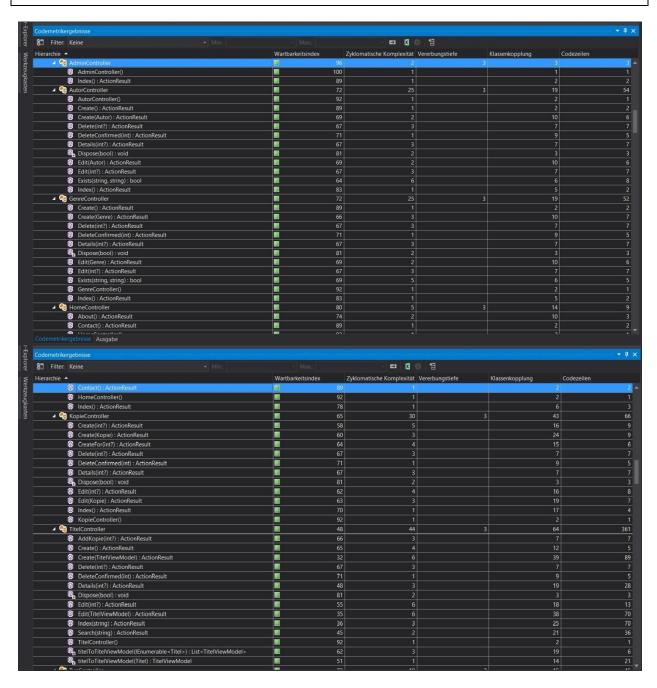
Diese Klasse hat eine relativ hohe Anzahl an Codepfade im Programmfluss, deshalb ist die zyklomatische Komplexität dieser Funktion etwas höher.

Online-Library	Version: <1.0>
Metrics	Date: 07.05.2014

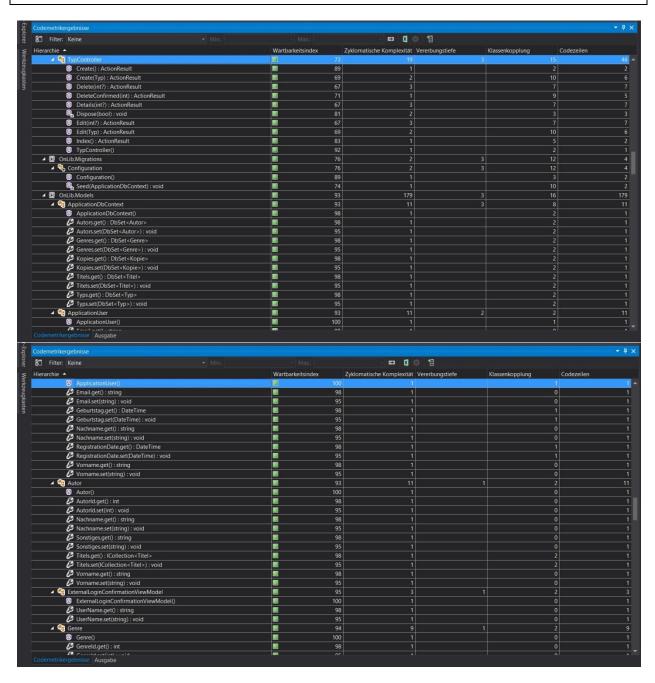
4. Screenshots (Visual Studio Code Metrics Powertool)



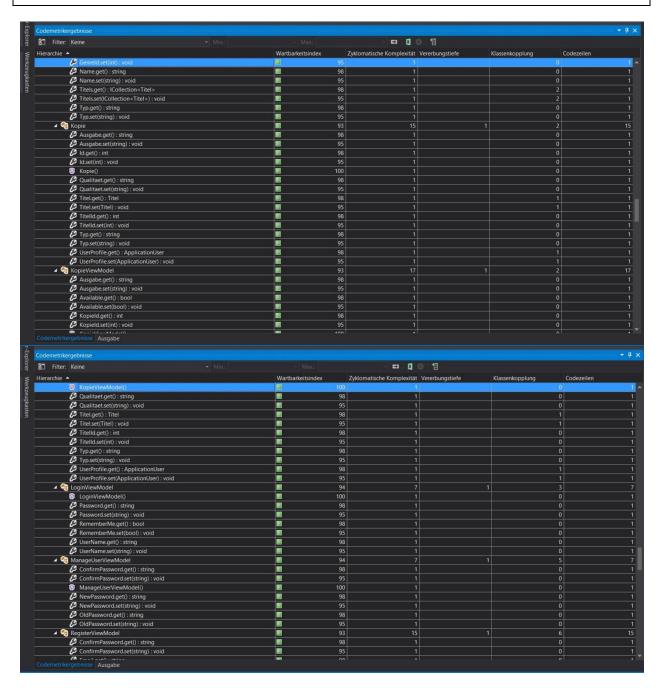
Online-Library	Version: <1.0>	
Metrics	Date: 07.05.2014	



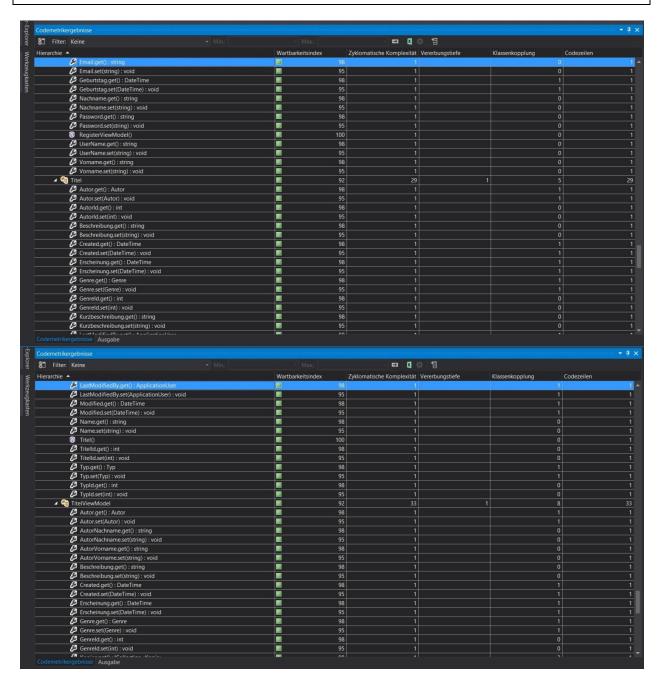
Online-Library	Version: <1.0>	
Metrics	Date: 07.05.2014	



Online-Library	Version: <1.0>	
Metrics	Date: 07.05.2014	



Online-Library	Version: <1.0>	
Metrics	Date: 07.05.2014	



Online-Library	Version: <1.0>
Metrics	Date: 07.05.2014

