

WeatherWallpaper

Janis Fix, Leon Gieringer

TINF18B3

Advanced Software Engineering

23. Mai 2021

Inhaltsverzeichnis

1	Einleitung	1
2	Clean Architecture	2
2.1	Vorher	2
2.2	Nachher	2
3	Entwurfsmuster	3
4	Programming Principles	4
4.1	SOLID	4
4.1.1	Single Responsibility Principle	4
4.1.2	Open/Closed Principle	4
4.1.3	Liskov Substitution Principle	4
4.1.4	Interface Segregation Principle	4
4.1.5	Dependency Inversion Principle	4
4.2	GRASP	4
4.2.1	Low Coupling	4
4.2.2	High Cohesion	4
4.2.3	Indirection	4
4.2.4	Polymorphism	4
4.2.5	Pure Fabrication	4
4.2.6	Protected Variations	4
4.3	DRY	5
4.4	YAGNI	5
5	Refactoring	6
5.1	Code Smells	6
6	Unit Tests	7
6.1	ATRIP	7
6.2	Beispiel für Unit-Tests und Mocks	7
6.3	Code Coverage	7

Listings

6.1	Unit-Test für den ConfigValidator	8
6.2	Unit-Test für den ImageHandler mit Mock	9

1 Einleitung

Hier steht meine Einleitung

2 Clean Architecture

2.1 Vorher

2.2 Nachher

- Schichtarchitektur planen und begründen
- ≥ 2 Schichten umsetzen

3 Entwurfsmuster

- ≥ 1 Entwurfsmuster einsetzen und begründen
- UML-Diagramm vorher und nachher

4 Programming Principles

4.1 SOLID

4.1.1 Single Responsibility Principle

4.1.2 Open/Closed Principle

4.1.3 Liskov Substitution Principle

Das Liskov Substitution Principle ist erfüllt, da abgesehen von den verwendeten Interfaces keine Vererbung verwendet wird.

4.1.4 Interface Segregation Principle

4.1.5 Dependency Inversion Principle

4.2 GRASP

4.2.1 Low Coupling

- ImageHandler
- WeatherHandler

4.2.2 High Cohesion

- ImageHandler
- WeatherHandler

4.2.3 Indirection

- Refresher

4.2.4 Polymorphism

4.2.5 Pure Fabrication

- ConfigValidator ist Pure fabrication und einzelne IValidationAspects sind domain Code
- StartUpHelper
 - ImageHandler
 - WeatherHandler
 - ConfigHandler

4.2.6 Protected Variations

- IBackgroundChanger Das Wechseln des Hintergrundbildes funktioniert auf verschiedenen Betriebssystemen (verschiedener Windowsversionen) unterschiedlich. Um dieser Änderungen standzuhalten bietet das Interface eine einheitliche Schnittstelle
- IImageWriter Selbes
- IFileAccessor Selbes
- IAPICaller Bei Änderung der Beschaffungsart der Wetter- bzw. Bild Daten

4.3 DRY

Don't Repeat Yourself

4.4 YAGNI

5 Refactoring

- Code Smells identifizieren
- ≥ 2 Refactoring anwenden und begründen

5.1 Code Smells

6 Unit Tests

Insgesamt wurden 29 Unit-Test geschrieben. Im Folgenden werden auf Einzelheiten zu den Unit-Tests eingegangen.

6.1 ATRIP

Die entwickelten Unit-Tests befolgen die ATRIP-Regeln. Das bedeutet also, dass sie...

- Automatic, also eigenständig ablaufen und ihre Ergebnisse selbst prüfen.
- Thorough, also gründlich (genug) sind und die wichtigsten Funktionalitäten prüfen. Dazu gehört bei unserem Use-Case:
 - Die Analyse der Wetterdaten,
 - Die Validierung der vom Nutzer eingegebenen Konfiguration,
 - Das Decodieren der Konfiguration,
 - Die Verarbeitung der Daten der APIs, sowie die Fehlerbehandlung der APIs
- Repeatable, also jederzeit (automatisch) ausführbar sind. Dabei wird beispielsweise im Falle des `WeatherInterpreterTest` darauf geachtet, dass der Test nicht von der aktuellen Systemzeit abhängig ist.
- Independent, also unabhängig voneinander in beliebiger Reihenfolge ausführbar sind. Kein Test ist auf das Ergebnis oder den Ablauf eines anderen Tests abhängig.
- Professional, also einfach verständlich sind.

Hier wahrscheinlich noch Screenshots für die einzelnen Unterpunkte

6.2 Beispiel für Unit-Tests und Mocks

Die Unit-Test wurden, sofern möglich, in der AAA-Normalform entwickelt. Bei Unit-Tests, die Exceptions erwarten musste der Act- und Assert-Schritt teilweise zusammengeführt werden. In Listing 6.1 wird einerseits gezeigt, wie der zu testende `ConfigValidator` im Konstruktor vor jedem Testdurchlauf neu initialisiert wird und die `ValidationAspects` registriert werden. Im Test selbst wird eine fehlerhafte Konfiguration erzeugt, da im Stadtnamen Zahlen vorhanden sind. Daraufhin wird überprüft, ob die Validierung die richtige `Exception` wirft und ob die `ExceptionMessage` richtig ist, also der Fehler korrekt erkannt wurde.

Als Beispiel für Mocks betrachten wir in Listing 6.2 ein Test für den `ImageHandler`. Dabei wird das Interface `IAPICaller` gemockt und die in Zeile x definierte `correctApiResponse` beim Aufruf der `Get`-Methode des API-Callers zurückgegeben. Durch den Einsatz des Mocks, lässt sich die Funktionalität des `ImageHandlers` testen ohne einen realen API-Caller zu verwenden. Am Schluss wird überprüft, ob das Ergebnis des Aufrufs mit dem erwarteten, eingegebenen Ergebnis übereinstimmt.

6.3 Code Coverage

Mithilfe der Visual Studio 2019 Enterprise Version lässt sich die Code Coverage für das Projekt ermitteln. Dabei erreicht WeatherWallpaper eine Code Coverage von knapp 42%. Dies ist in Abbildung 1 zu sehen. Des Weiteren bietet das Visual Studio Tool die Möglichkeit einzusehen, welche Zeilen von den Tests abgedeckt werden und welche nicht. Zeilen die nicht abgedeckt werden, werden rot hinterlegt und abgedeckte Zeilen blau, wie in Abbildung 2 zu sehen.

```

1 public ConfigValidatorTest()
2 {
3     // ConfigValidator is needed in every unit test, so we initialize it here
4     _configValidator = new ConfigValidator();
5     _configValidator.Register(new IsCorrectCity());
6     _configValidator.Register(new IsCorrectInterval());
7 }
8
9 [Fact]
10 public void ValidateInputsFalseCity()
11 {
12     // Arrange
13     const string city = "F4k3 ci7y";
14     const string country = "DE";
15     const int interval = 10;
16     Config conf = new Config()
17     {
18         Interval = interval,
19         Location = new Location()
20         {
21             City = city,
22             CountryAbrv = country
23         }
24     };
25     const string actualExceptionMessage
26         = "Stadtname beinhaltet unbekannte Zeichen.";
27     // Act, Assert
28     var ex = Assert.Throws<BadConfigException>(
29         () => _configValidator.ValidateInputs(conf));
30     Assert.Equal(actualExceptionMessage, ex.Message);
31 }

```

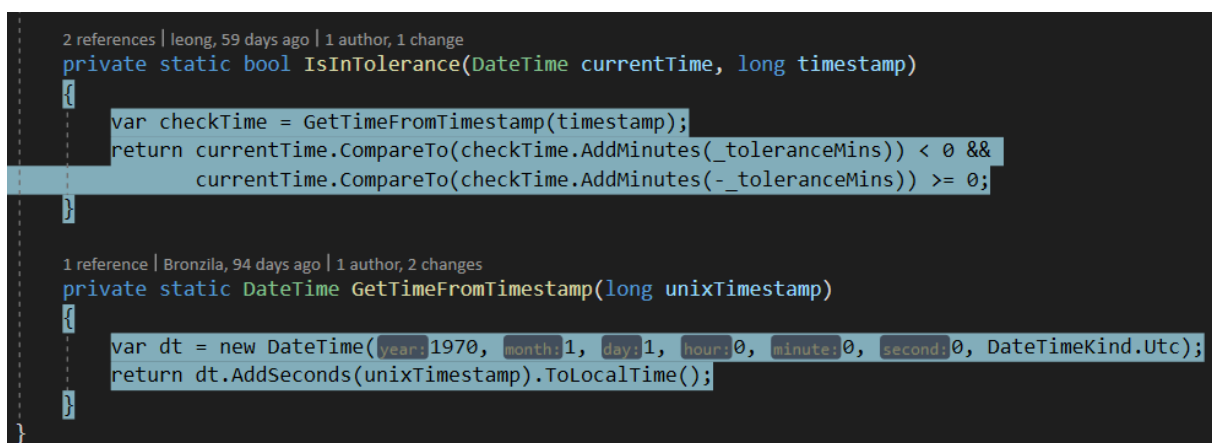
Listing 6.1: Unit-Test für den ConfigValidator

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Bloc...	Covered (% Blocks)
janis_DESKTOP-4OTT554 2021-05-18 19_15_41.coverage	355	27,54 %	934	72,46 %
weatherwallpapertest.dll	5	0,73 %	681	99,27 %
weatherwallpaper.dll	350	58,04 %	253	41,96 %
WeatherWallpaper.Classes.Helpers	115	47,33 %	128	52,67 %
WeatherWallpaper.Classes.Handler	13	16,88 %	64	83,12 %
WeatherWallpaper.Classes.API	0	0,00 %	43	100,00 %
WeatherWallpaper.Classes.Models	0	0,00 %	16	100,00 %
WeatherWallpaper.Classes.Exceptions	0	0,00 %	2	100,00 %
WeatherWallpaper	129	100,00 %	0	0,00 %
WeatherWallpaper.Classes.Background	39	100,00 %	0	0,00 %
WeatherWallpaper.Classes.Controllers	54	100,00 %	0	0,00 %

Abbildung 1: Code Coverage Ergebnisse

```
1 [Fact]
2 public void GetImageDataSuccessful()
3 {
4     //Arrange
5     var correctApiResponse = new ImageResponse()
6     {
7         Results = new List<Images>{
8             new Images { Links = new Links
9             {
10                 Download = "https://unsplash.com/photos/XxElwSAH0AA/download"
11             }}}
12 };
13 var responseJson = JObject.FromObject(correctApiResponse);
14 var api = new Mock<IAPICaller>();
15 api.Setup(caller => caller.Get(It.IsAny<string>()))
16     .Returns(Task.FromResult(responseJson));
17 var handler = new ImageHandler(api.Object);
18 string queryString = "?query=doesnt matter";
19
20 //Act
21 var result = handler.GetImageData(queryString);
22
23 //Assert
24 Assert.Equal(result.Results.First().Links.Download,
25     correctApiResponse.Results.First().Links.Download);
26 }
```

Listing 6.2: Unit-Test für den ImageHandler mit Mock



```
2 references | leong, 59 days ago | 1 author, 1 change
private static bool IsInTolerance(DateTime currentTime, long timestamp)
{
    var checkTime = GetTimeFromTimestamp(timestamp);
    return currentTime.CompareTo(checkTime.AddMinutes(_toleranceMins)) < 0 &&
        currentTime.CompareTo(checkTime.AddMinutes(-_toleranceMins)) >= 0;
}

1 reference | Bronzila, 94 days ago | 1 author, 2 changes
private static DateTime GetTimeFromTimestamp(long unixTimestamp)
{
    var dt = new DateTime(year:1970, month:1, day:1, hour:0, minute:0, second:0, DateTimeKind.Utc);
    return dt.AddSeconds(unixTimestamp).ToLocalTime();
}
```

Abbildung 2: Code Coverage Highlighting