Unit Testing

Interaktive Session am Teamtag „Anwendungsentwicklung“

Stefan Bente

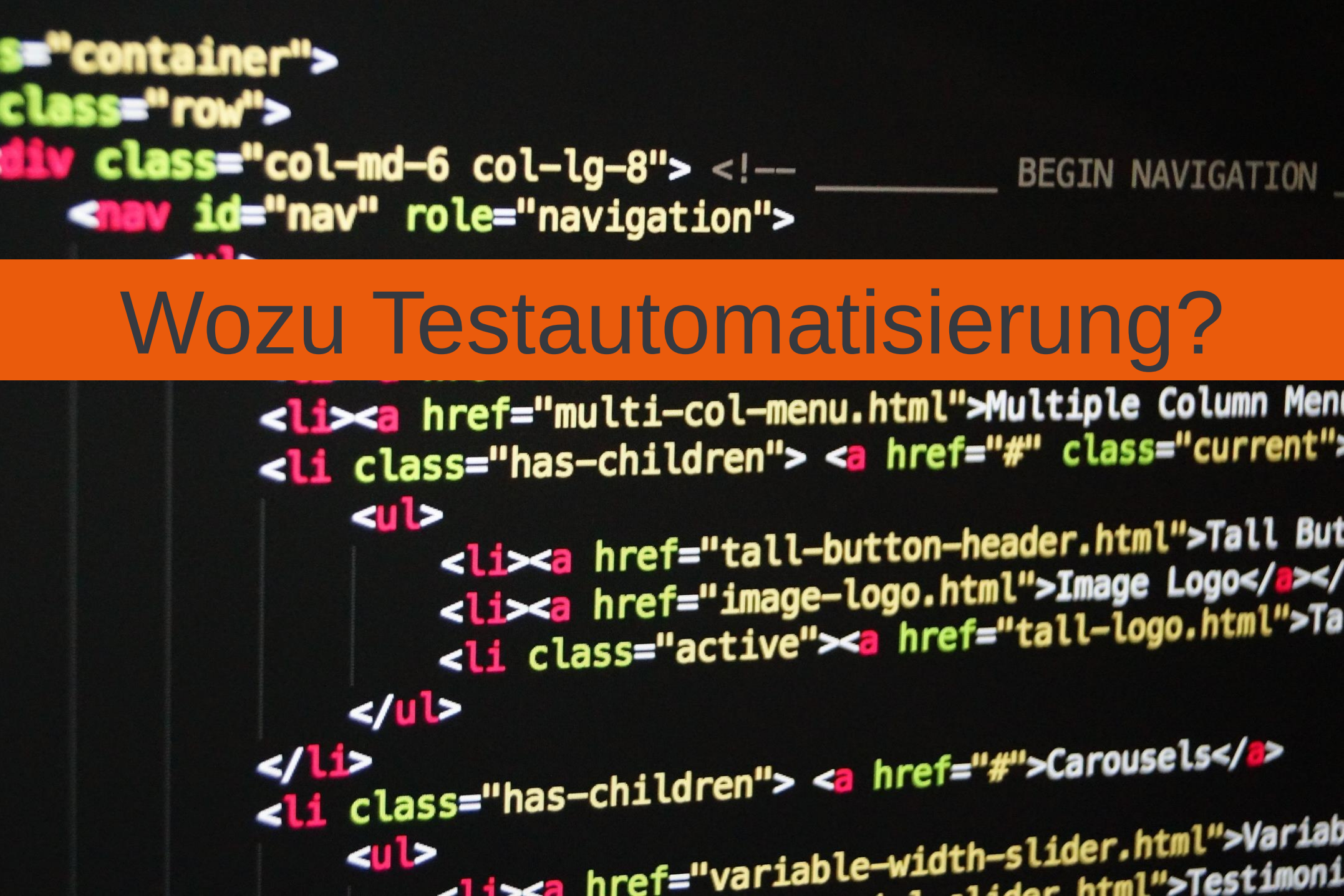
TH Köln
Cologne Institute for Digital Ecosystems (CIDE)
Software Architecture Lab (ArchiLab)

Technology
Arts Sciences
TH Köln

Was euch erwartet

1. Intro und Motivation
2. Unit Test Essentials
3. Refactoring
4. TDD und Testen neuer Features
5. Test-Typen, Mocks und Fakes
6. Anwendung in der „echten Welt“

www.archi-lab.io/unittest


s="container">
class="row">
div class="col-md-6 col-lg-8"> <!-- BEGIN NAVIGATION
<nav id="nav" role="navigation">
Wozu Testautomatisierung?
<a href="multi-col-menu.html">Multiple Column Menu
<li class="has-children"> <a href="#" class="current">

<a href="tall-button-header.html">Tall But
Image Logo</
<li class="active"><a href="tall-logo.html">Ta

<li class="has-children"> Carousels

<a href="variable-width-slider.html">Variab

Vorteile der Testautomatisierung

- Qualität der Software
- Schnelligkeit bei Deployment ➔
  Schnelligkeit bei neuen Features
- Sicheres Refactoring ➔
  Abbau technischer Schulden
- Tests als Helfer beim Design von Code

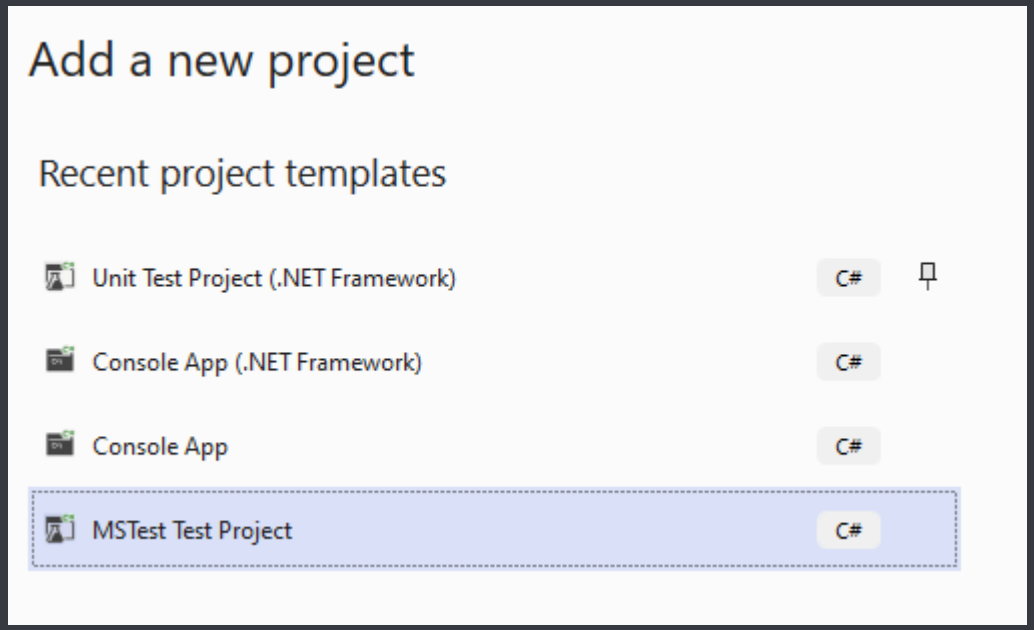
Unit Test Essentials

Bild: © https://www.sportsivia.com/de/products/alle-essentials-pakken

Unit Test Essentials

- Testprojekt in VS
- Testklassen
- AAA (Arrange, Act, Assert)
  - auch: given, when, then
- Assert-Typen
- Happy Path, Edge Case, Out of Bounds
- Pfadfinder-Regel

Testprojekt und Testklassen



- https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022#create-unit-test-projects-and-test-methods-c

Aufbau der Testklassen

- Tipp: Nutzt geschachelte Klassen (siehe Manuel & Georgs Anleitung)

```
[TestClass]
public class PersonServiceTest
{
    [TestClass]
    public class IsValidAccountName
    {
        [TestMethod]
        public void ValidAccountNameWithUpperAndLowerCase()...

        [TestMethod]
        public void ValidAccountName()...

        [TestMethod]
        public void InvalidAccountName()...

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void ShouldThrowArgumentNullExceptionCaseEmptyValue()...

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void ShouldThrowArgumentNullExceptionCaseNullValue()...
    }
```

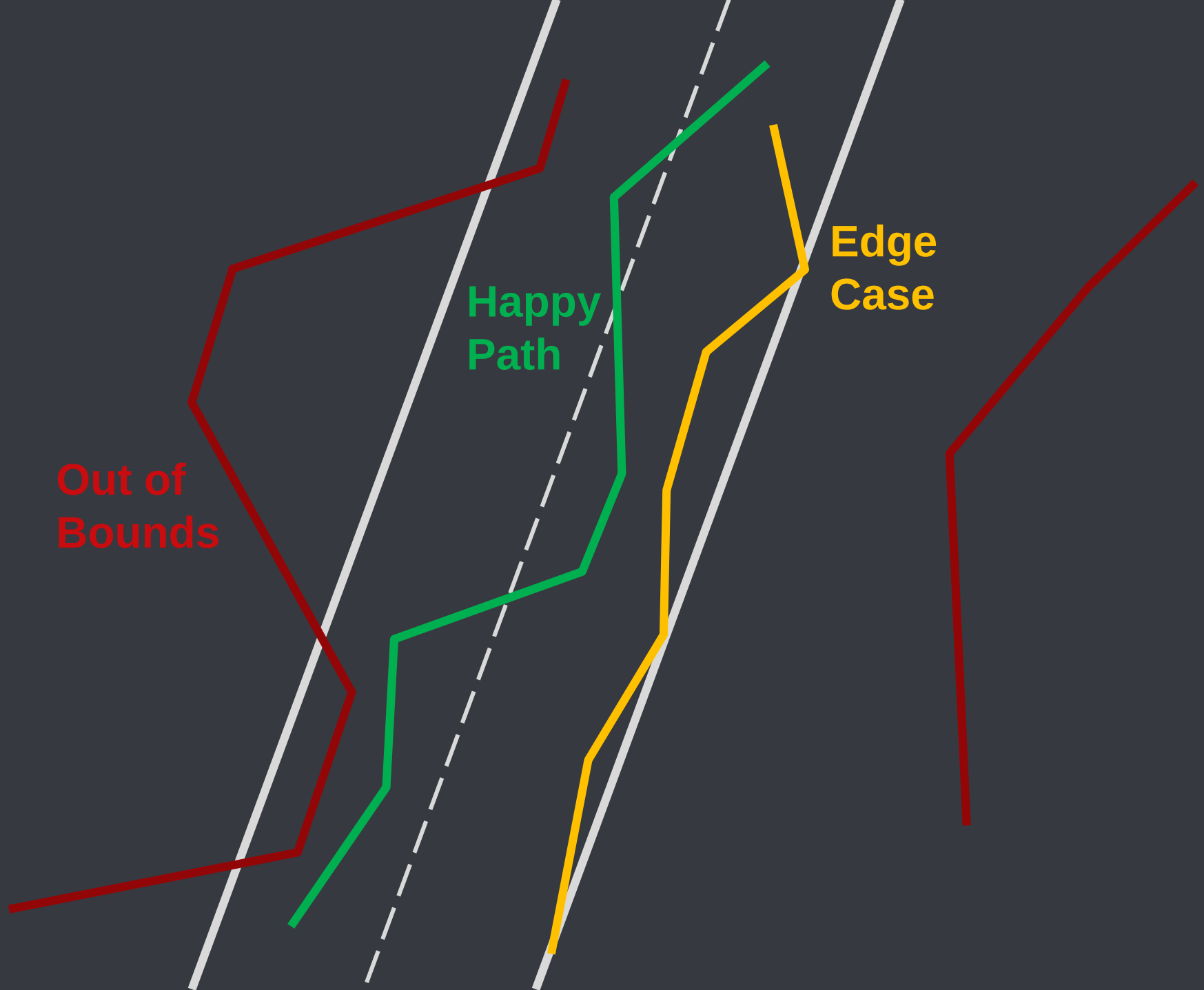
- `// arrange` `// given`
  - Initialisierung, Werte setzen, “Test vorbereiten”
- `// act` `// when`
  - Zu testende Funktionalität ausführen
- `// assert` `// then`
  - Nachbedingungen testen

```
[TestMethod]
public void Withdraw_ValidAmount_ChangesBalance()
{
    // arrange
    double currentBalance = 10.0;
    double withdrawal = 1.0;
    double expected = 9.0;
    var account = new CheckingAccount("JohnDoe", currentBalance);

    // act
    account.Withdraw(withdrawal);

    // assert
    Assert.AreEqual(expected, account.Balance);
}
```

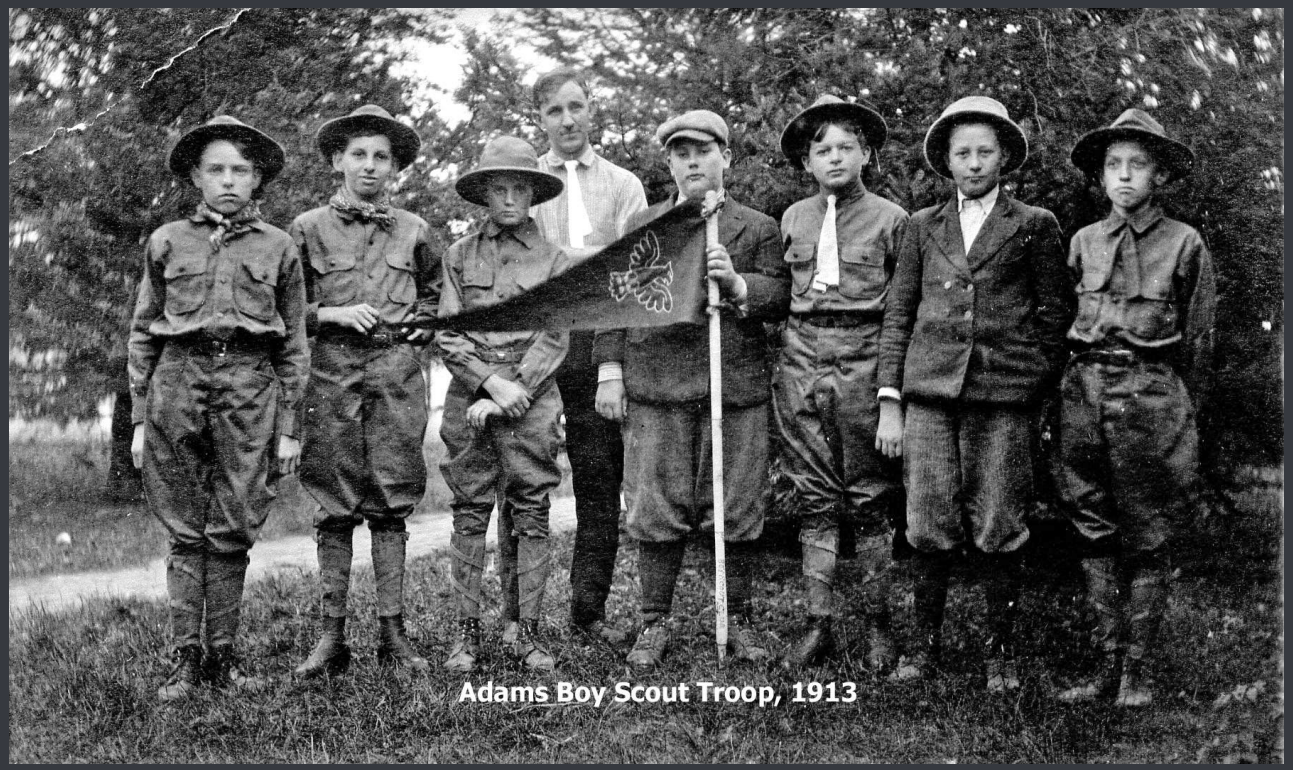
https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022

```
[TestMethod]
public void Withdraw_AmountMoreThanBalance_Throws()
{
    // arrange
    var account = new CheckingAccount("John Doe", 10.0);

    // act and assert
    Assert.ThrowsException<System.ArgumentException>(() => account.Withdraw(20.0));
}
```

https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022


Out of
Bounds
Happy
Path
Edge
Case

Pfadfinder-Regel



„Verlasst den Zeltplatz immer ein bisschen sauberer, als ihr ihn vorfunden habt“

Refactoring

Refactoring = Abbau technischer Schuld

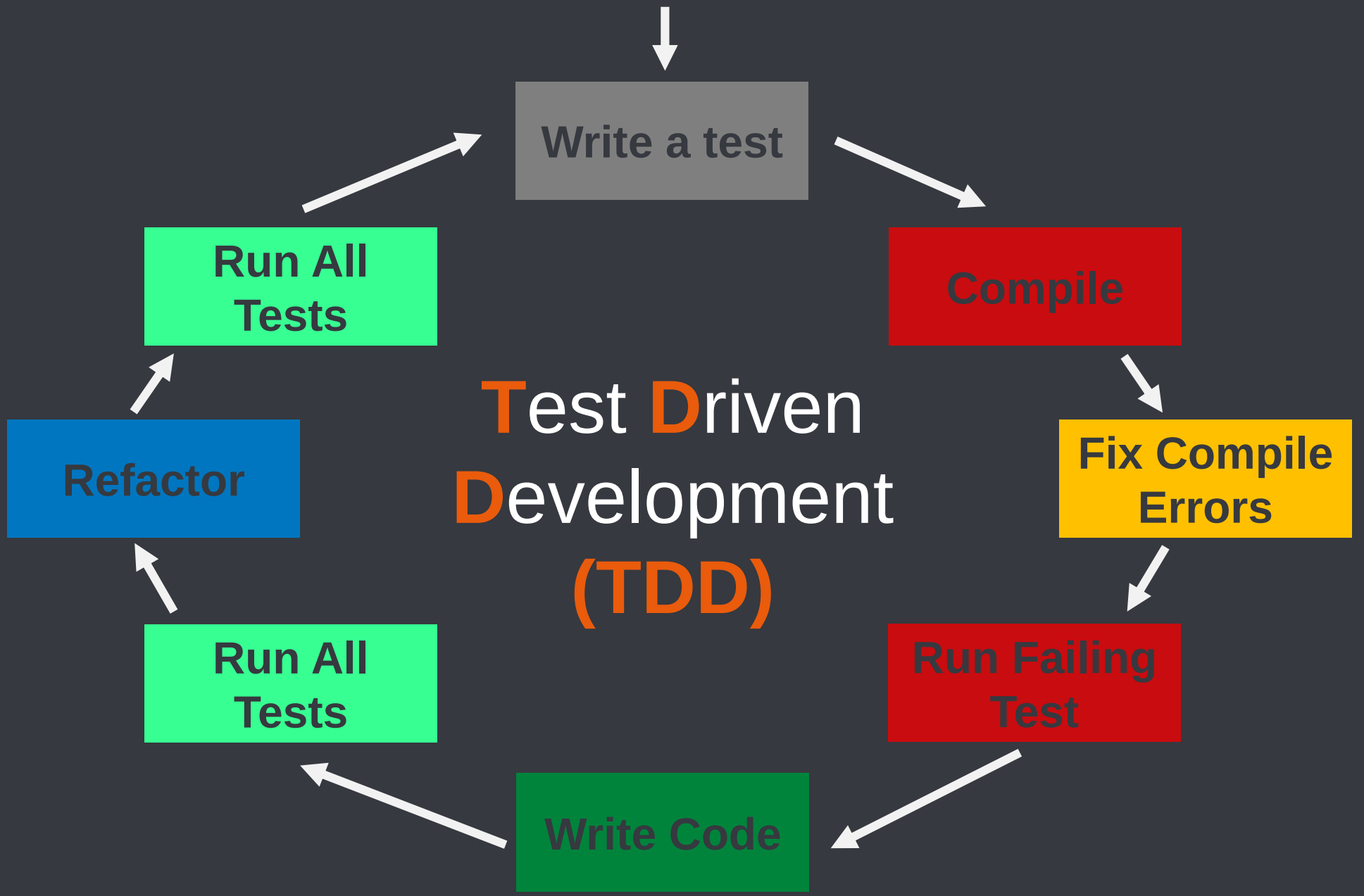
- Verstöße gegen Clean Code
- Verstöße gegen SOLID-Prinzipien
- Extraktion von Abstraktionen für Reuse
- Verstreute Funktionalität
- …

Vorgehen beim Refactoring

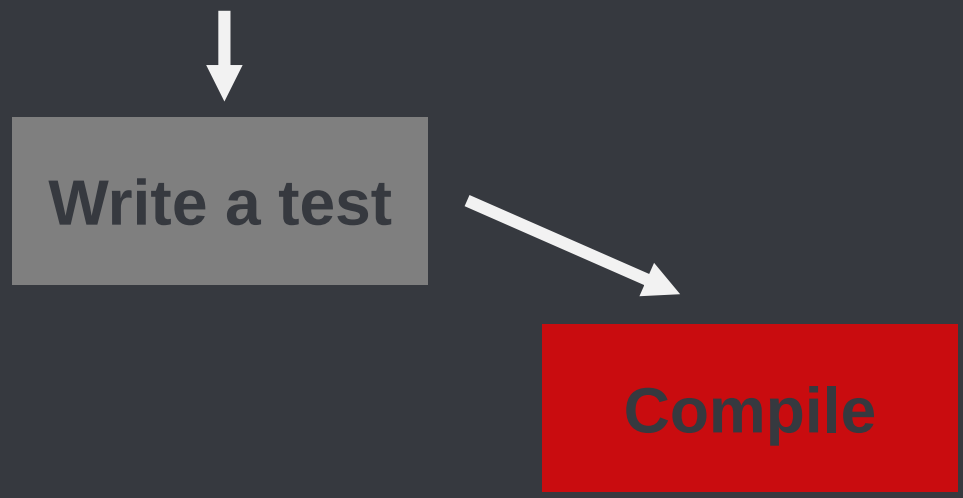
1. Vorhaben identifizieren
2. Funktionalität durch Unit Tests absichern
3. Refactoring durchführen
4. Sicherstellen, dass Tests (wieder) grün sind

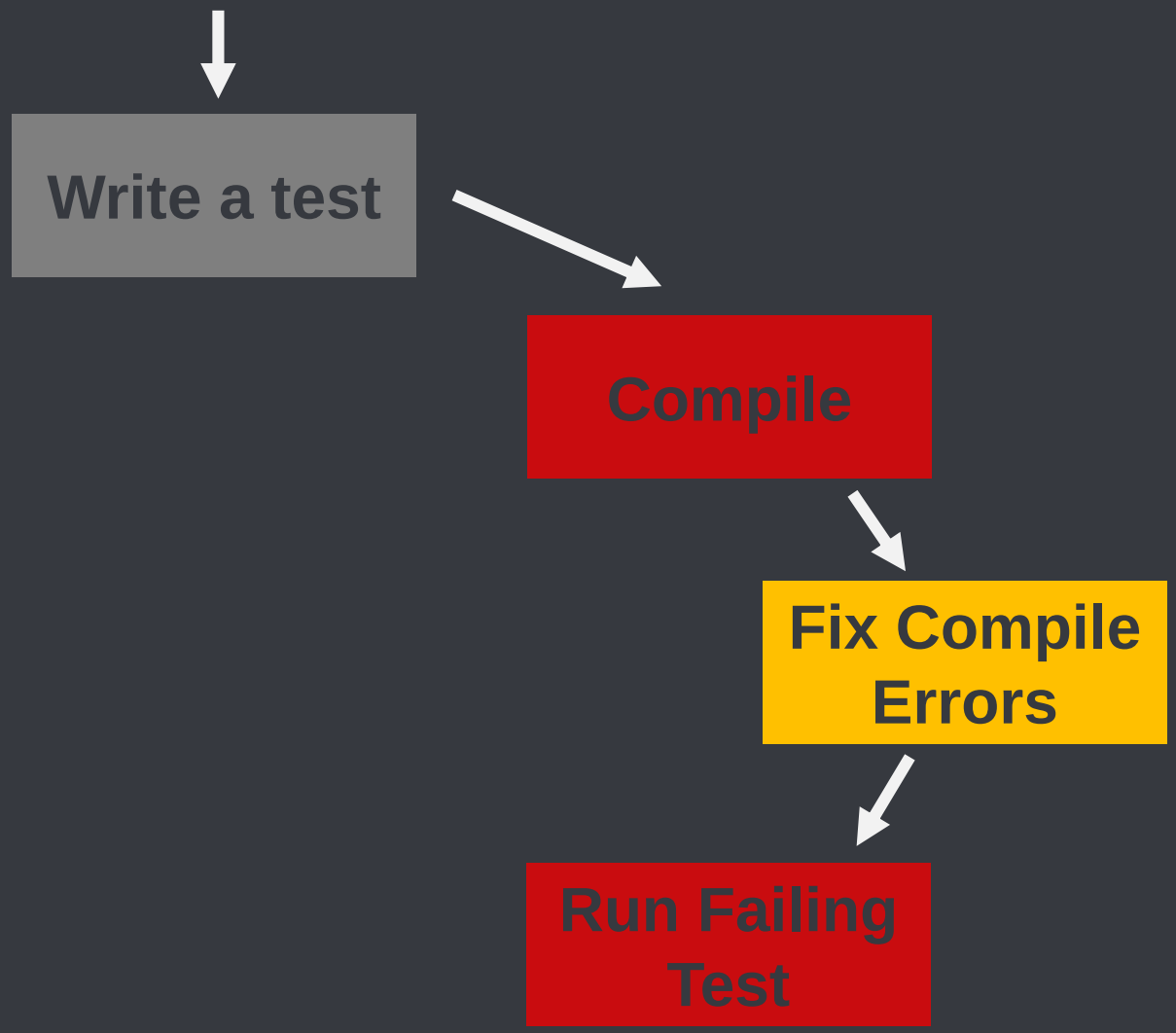
Test-Driven Development (TDD)

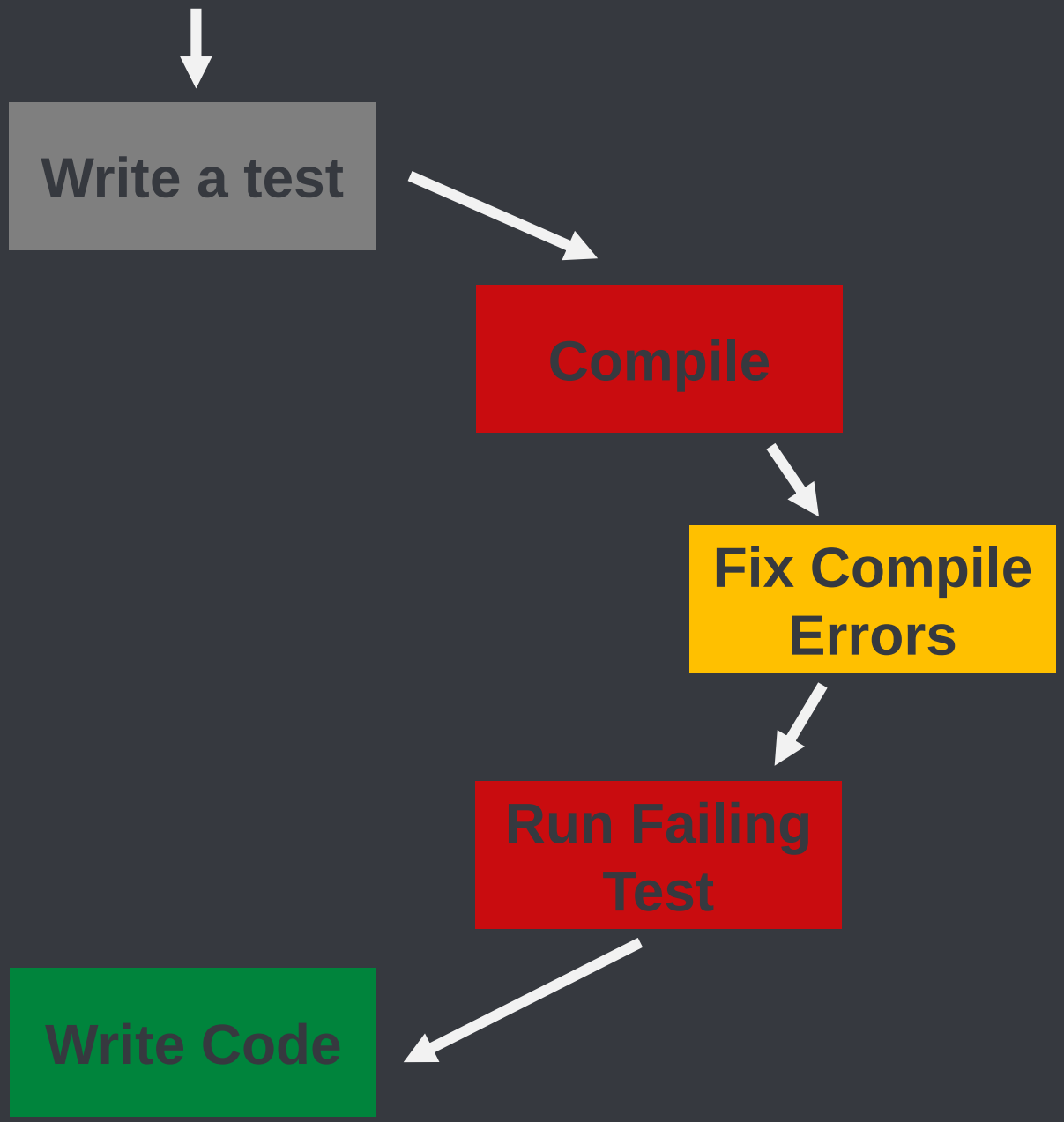
By Rygle, based on Purzen - https://openclipart.org/detail/261323/cartoon-moon-rocket-remix-2, CC0, https://commons.wikimedia.org/w/index.php?curid=76766171

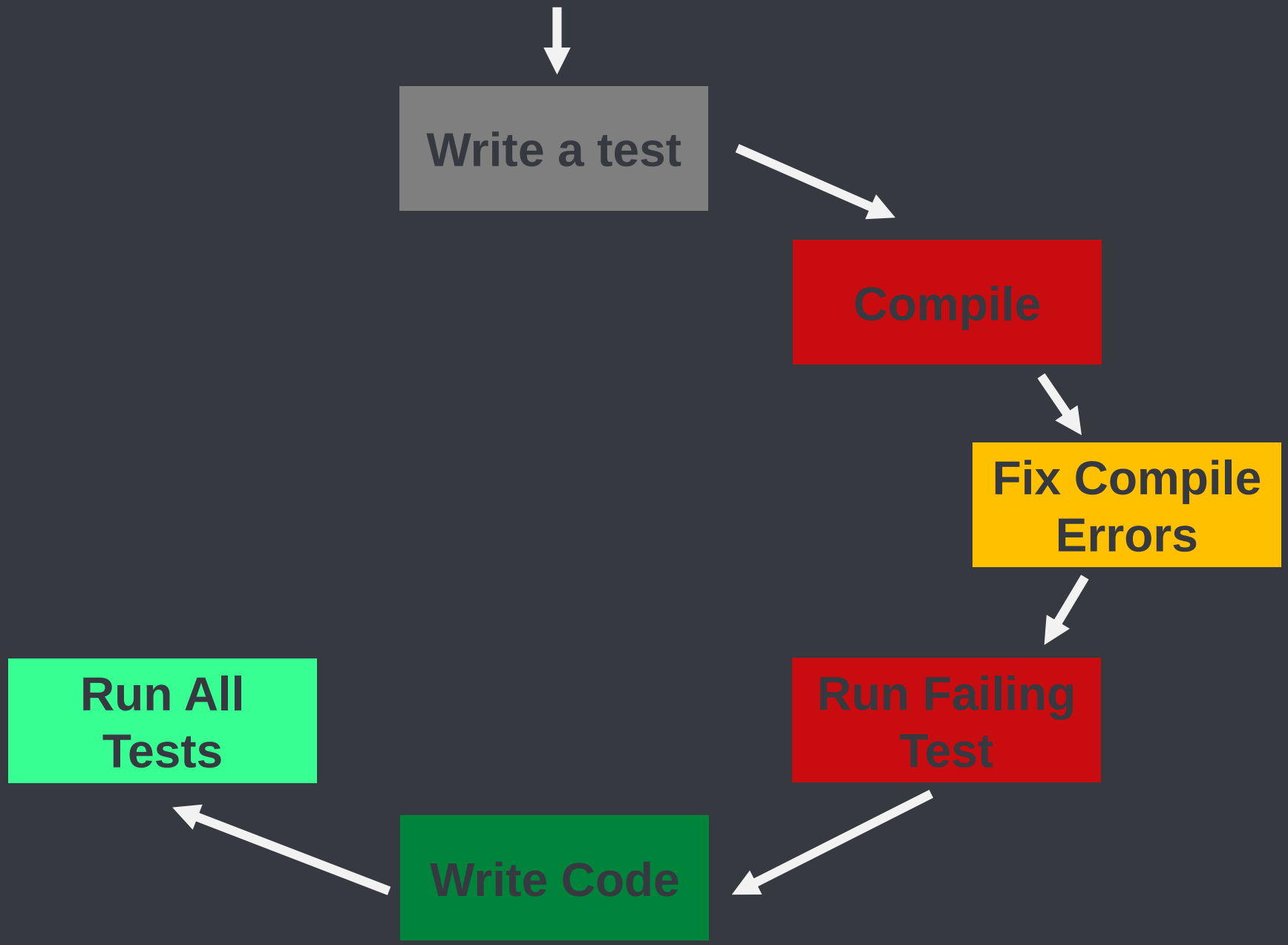


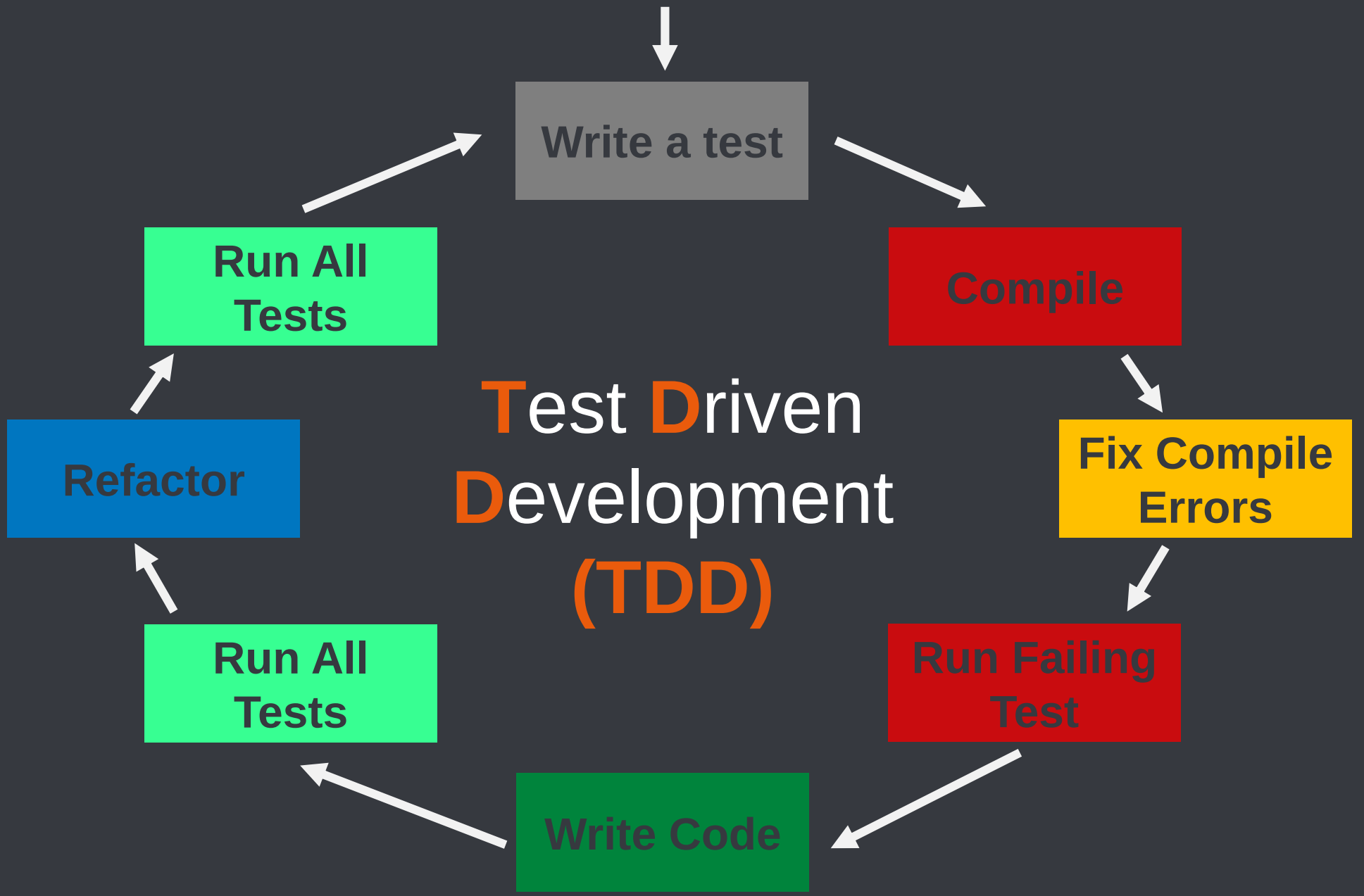
Angelehnt an https://github.com/ThoughtWorksInc/TWU101-TDDIntro


Write a test
Compile


Write a test
Compile
Fix Compile Errors
Run Failing Test


Write a test
Compile
Fix Compile Errors
Run Failing Test
Write Code


Write a test
Compile
Fix Compile Errors
Run Failing Test
Write Code
Run All Tests



Angelehnt an https://github.com/ThoughtWorksInc/TWU101-TDDIntro

Test-Typen, Mocks und Fakes

Unit Tests

- testen isolierte Code-Stücke
- typischerweise für Business-Logik
  - Domain Layer
- während Entwicklung häufig ausgeführt
  - ➔ sehr schnell, z.B. 200 Tests in <5 sec
- i.d.R. ohne technische Abhängigkeiten
  - laufen im Dev Env und CI-Pipeline
  - am besten auch ganz ohne Frameworks

Integration Tests

- testen Zusammenspiel verschiedener Komponenten
- Nutzen Mocks und Fakes
  - ➔ siehe gleich
- Müssen auch im Dev Env laufen
  - … aber können länger dauern

Weitere Typen von Tests (nicht vollständig)

- Smoke Tests
- E2E-Tests
- Systemtests / Acceptance Tests
- Performance / Load / Stress Tests
- Security Tests
- Compatibility Tests

Fake

- isoliert Abhängkeiten
- einfach
- i.d.R. selbst gecoded
- Beispiel:
- Fake für den Web-Service-Call

Mock

- isoliert Abhängkeiten
- komplex
- i.d.R. Framework
- Beispiele:
- ASP.NET Core Test Host
- InMemoryDB

www.archi-lab.io