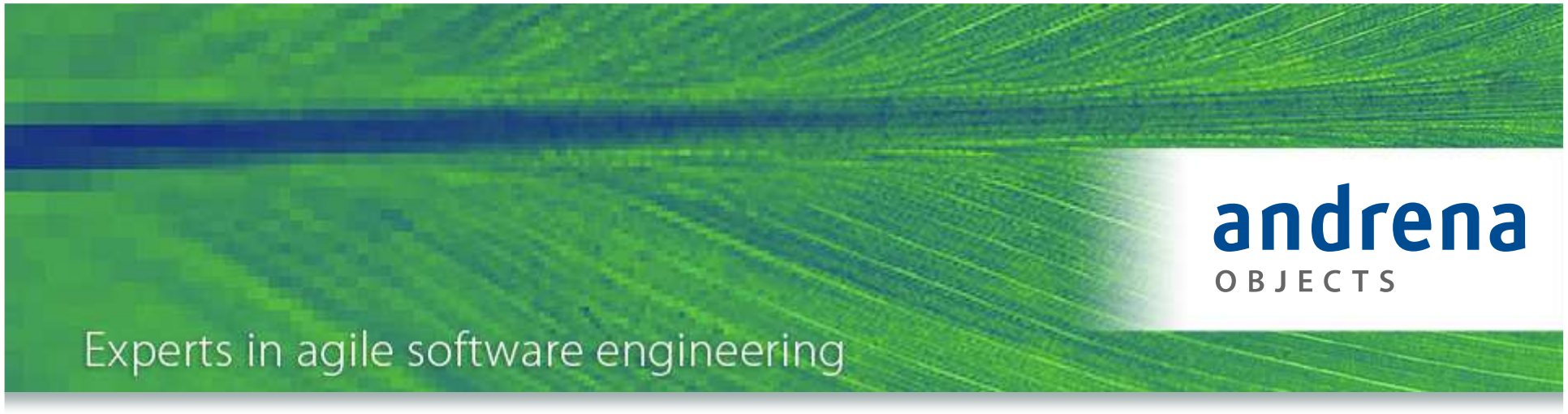
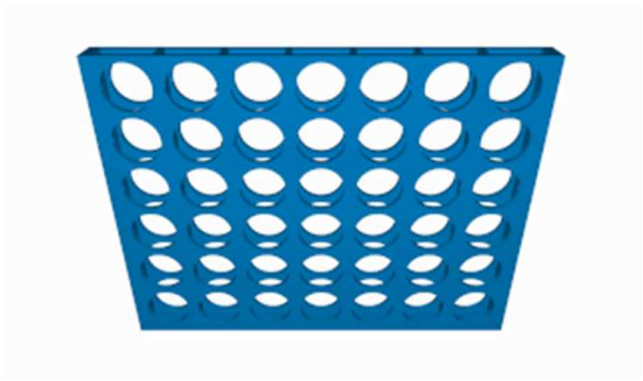


Clean – works for code, works for test

Stefan Mandel, Felix Schad

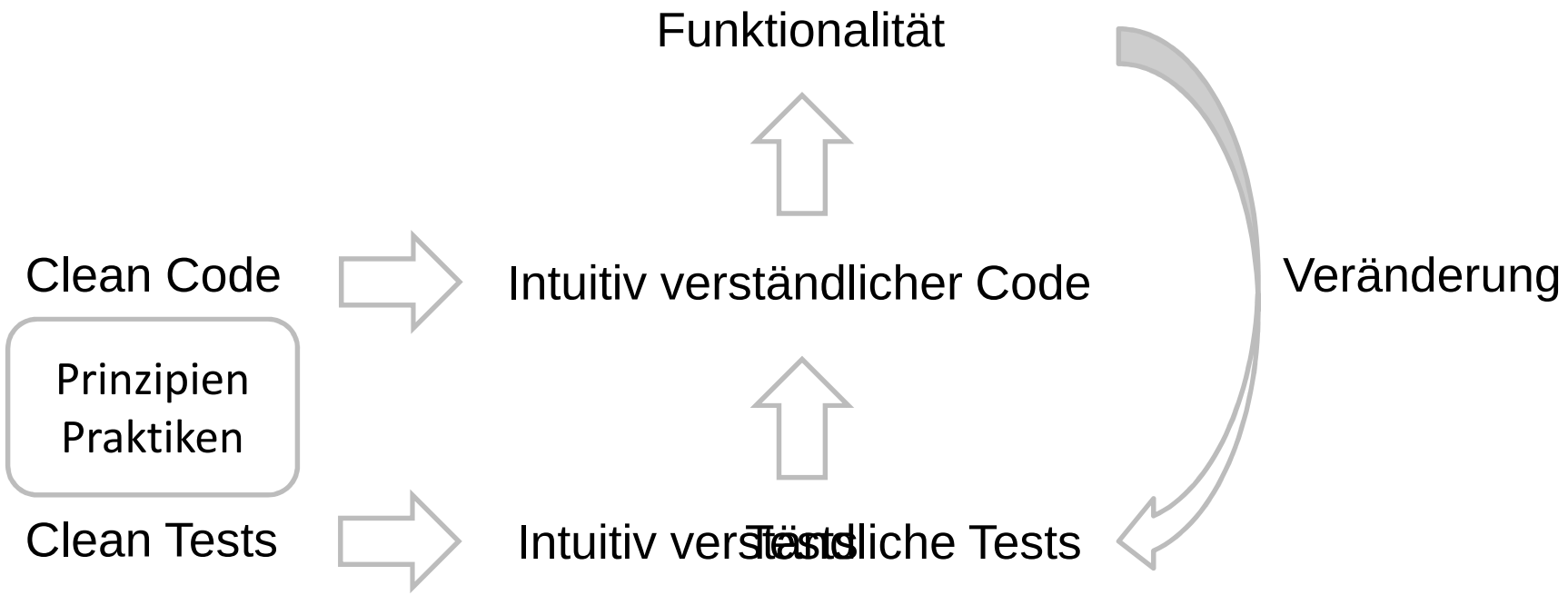
Beispiel –„4 Gewinnt“



```
@Test
public void testIsFinished() throws Exception {
   grid.drop(0, RED);
   grid.drop(4, RED);
   grid.put(4, 4, RED);
   grid.put(0, 4, RED);

   assertThat(grid.isC nComplete(0, RED), is(false));
   assertThat(grid.is Comp te(0, RED), is(false));
   assertThat(grid. iagonalComplete(0, RED), is(false));
   assertThat(grid alComplete(0, RED), is(false));

   grid.put(2, 2, RED);
   assertThatNothingCha );
   grid.put(1, 1, RED);
   assertThatNothingChanged();
   grid.put(3, 3, RED);
   assertThat(grid.isColumnCom ete(0, RE (false));
   assertThat(grid.isRowComplete(0, RE s(f e));
   assertThat(grid.isDifferenceDiag Co ete( , RED), is(true));
   assertThat(grid.isSumDiagonalCom ete(0, ED), is(false));
   grid.put(1, 1, null);
   assertThat(grid.isSumDiagonalComplete(4, RED), is(f se));
   grid.put(1, 3, RED);
   assertThat(grid.isSumDiagonalComplete(4, ED), is(false));
   grid.put(3, 1, RED);
   assertThat(grid.isSumDiagonalComplete(4, RED), is(true));
   for (int i = 0; i < 5; i++) {
      grid.put(i, 0, YELLOW);
   }
   for (int j = 0; j < 5; j++) {
      if (j == 0) {
         assertThat(grid.isRowComplete(j, YELLOW), is(true));
      } else {
         assertThat(grid.isRowComplete(j, YELLOW), is(false));
      }
   }
}
```

WTF?

Clean Code – Clean Tests

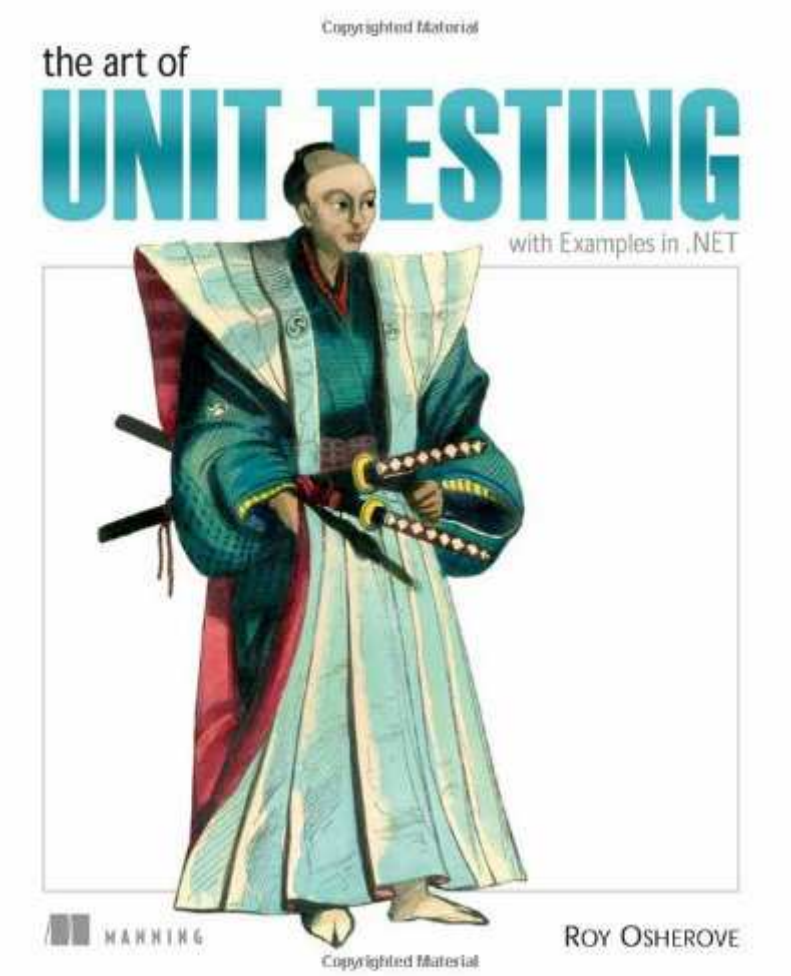


Principles (Roy Osherove)

Trustworthy

Maintainable

Readable



Principles (Object Mentor)

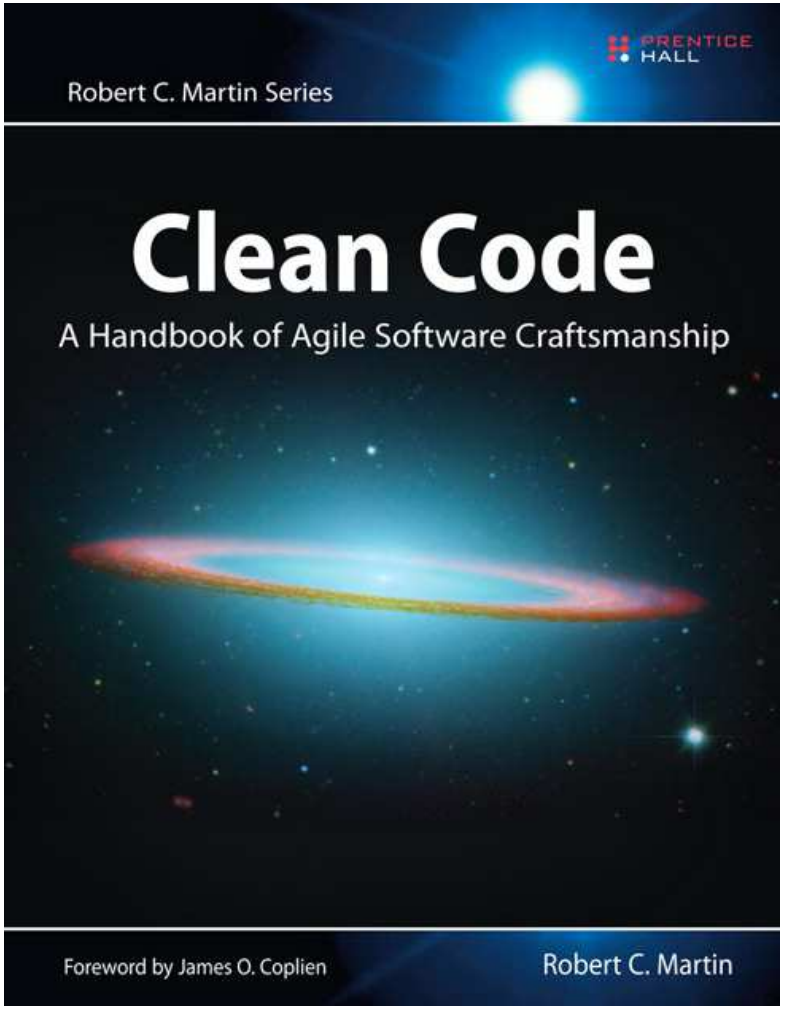
Fast

Independent

Repeatable

Self-validating

Timely



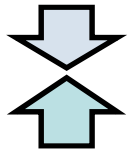
Concrete Concise Compact Consequent

C^4 PATTERN

Concrete

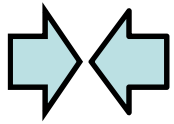
- Geht es konkreter?
  - Informationen hinzufügen?
  - Jeder Schritt sichtbar?
  - Jeder Zustand sinnvoll benannt?
  - Kontext-Information in den Namen enthalten?
  - Schleifen und Methodenaufrufe auflösen?
  - Bedingungen auf verschiedene Tests verteilen?

Concise



- Geht es kürzer/knapper?
  - Schritte weglassen?
  - Prüfungen weglassen?
  - Ohne den Zweck des Tests zu ändern

Compact



- Geht es kompakter/dichter?
  - Jeder Schritt so kurz wie möglich dargestellt?
  - Zeile Verkürzen, indem man Information daraus versteckt?
  - Wesentliche Informationen in einer Zeile hervorheben?

Consequent

- Geht es systematischer?
  - AAA Pattern
  - Nur Grenzfälle
  - Ein Assert
  - Naming Pattern
  - Fixture Systematik

Arrange Act Assert

AAA PATTERN

AAA-Pattern

- Arrange:

  Vorbereitungen, um das Szenario aufzusetzen (Fixture)

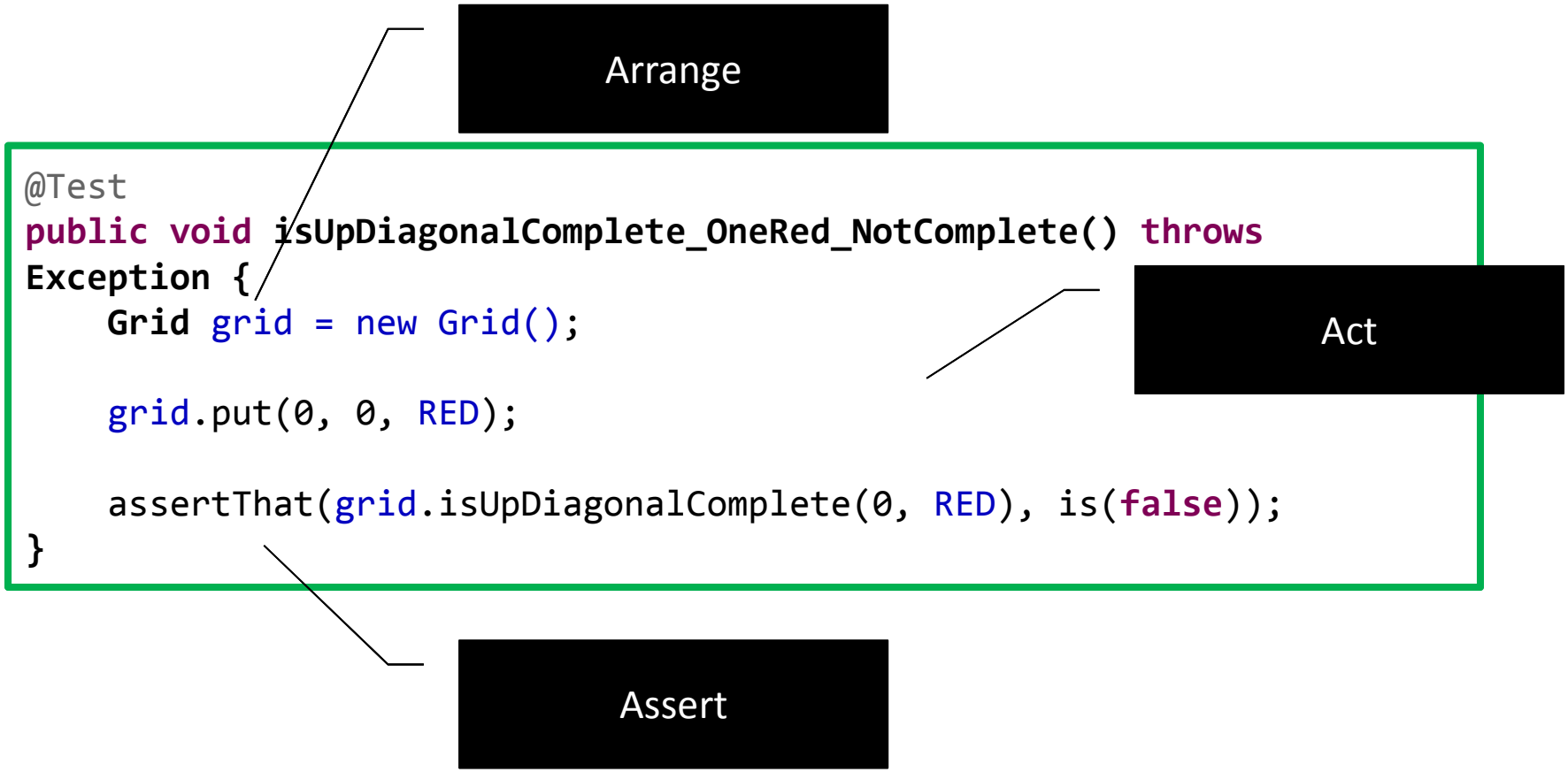
- Act:

  Ausführen der zu testenden Methode

- Assert:

  Prüfen des Zustands nach dem Testaufruf

AAA-Pattern

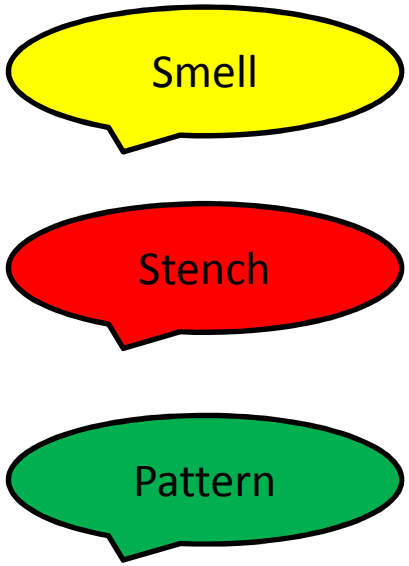

Arrange
Act
Assert
@Test
public void isUpDiagonalComplete_OneRed_NotComplete() throws
Exception {
Grid grid = new Grid();
grid.put(0, 0, RED);
assertThat(grid.isUpDiagonalComplete(0, RED), is(false));
}

andrena
OBJECTS

BEISPIELE

Die folgenden Beispiele enthalten

- Bad
  - Wie man sie in freier Wildbahn vorfindet
- Good
  - Wie man sie lieber vorfinden würde
- Smells
  - Was auf Probleme hindeutet
- Stenches
  - Was man lieber ganz vermeidet
- Patterns
  - Was man praktisch immer tun kann



Agenda

- Naming Variables
- Naming Methods
- Magic Values
- Multiple Aspects per Test
- Incremental Testing
- Superfluous Arrange
- Delegated Arrange-Act
- Excessive Stubbing
- Conditions and Loops
- Ignore Setup

> “There are only two hard things in Computer Science: cache invalidation and naming things.”
> – Phil Karlton

NAMING VARIABLES

Naming Variables

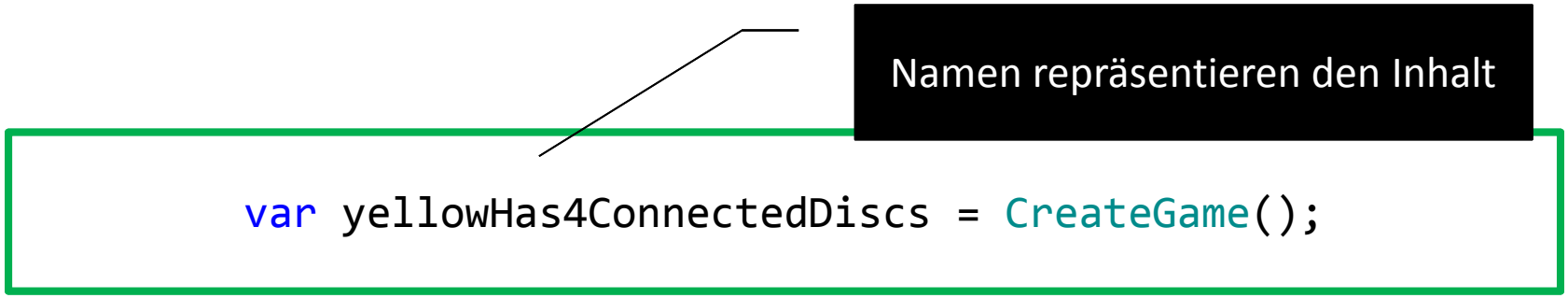
```
[Test]
public void IsFinishedTest1()
{
        var cut = CreateGame();
        Assert.That(cut.IsFinished(), Is.True);
}
```

Naming Variables

```
[Test]
public void IsFinishedTest1()
{
        var yellowHas4ConnectedDiscs = CreateGame();
        Assert.That(yellowHas4ConnectedDiscs.IsFinished(),
                        Is.True);
}
```

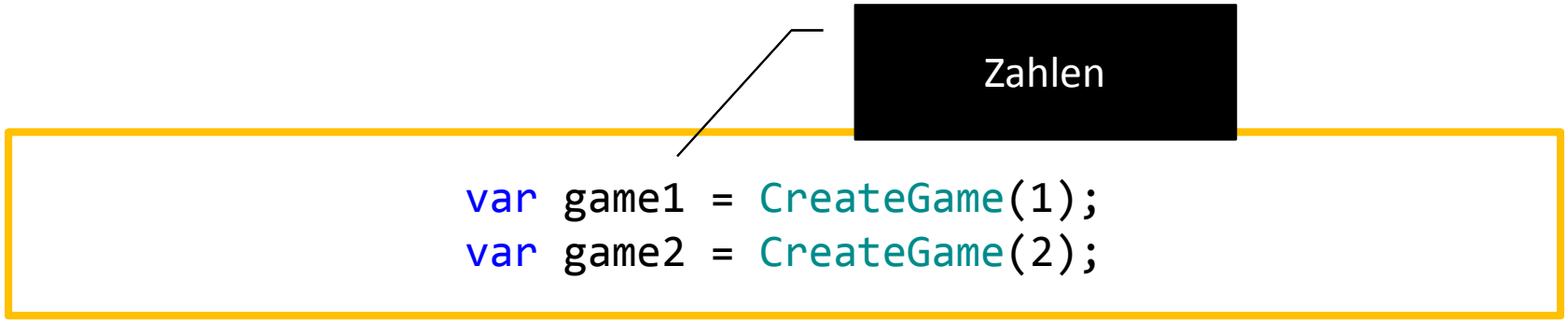
Naming Variables

Pattern



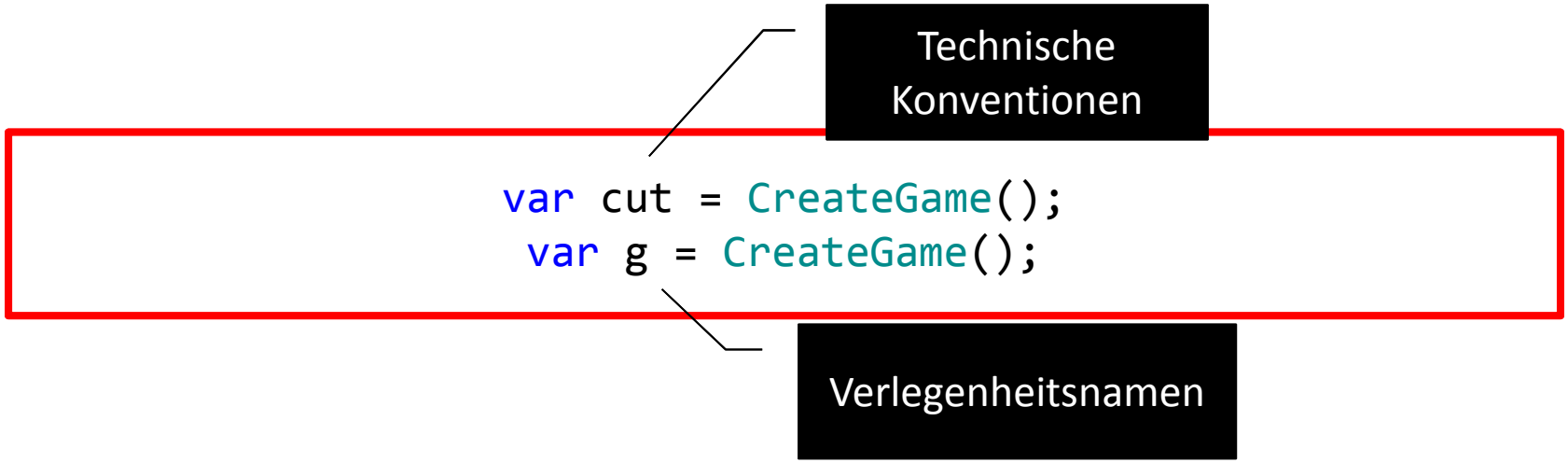
Naming Variables

Smell



Naming Variables

Stench



andrena
OBJECTS

NAMING METHODS

Naming Methods

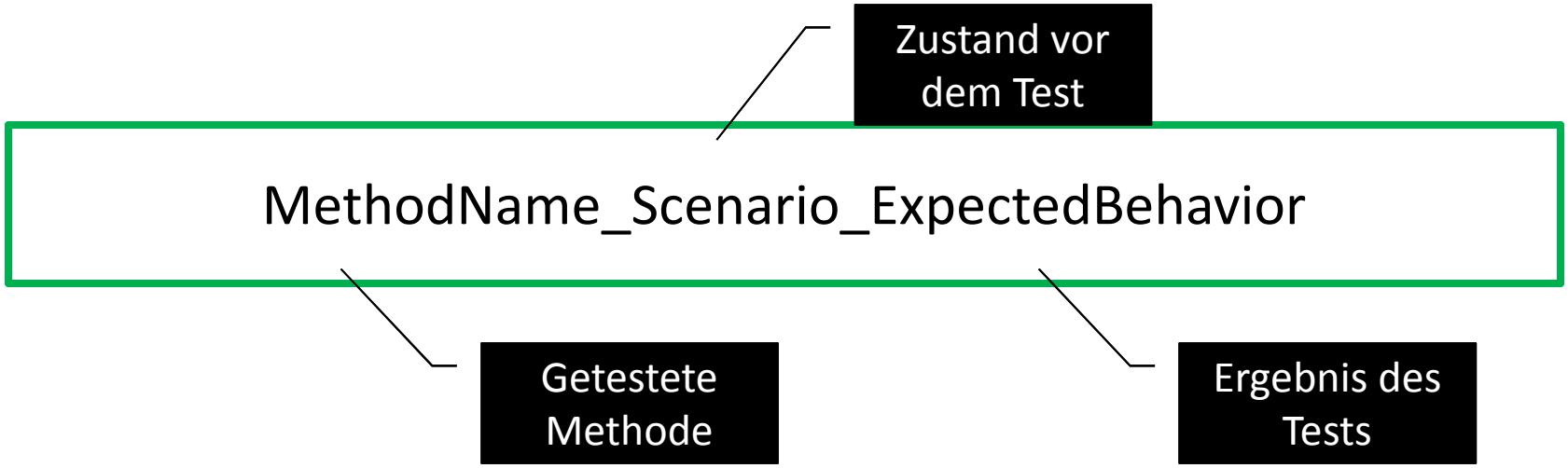
```
[Test]
public void IsFinishedTest1()
{
        var cut = CreateTestData();
        Assert.That(cut.IsFinished(), Is.True);
        Assert.True(cut.GetWinner() == Color.Yellow);
}
```

Naming Methods

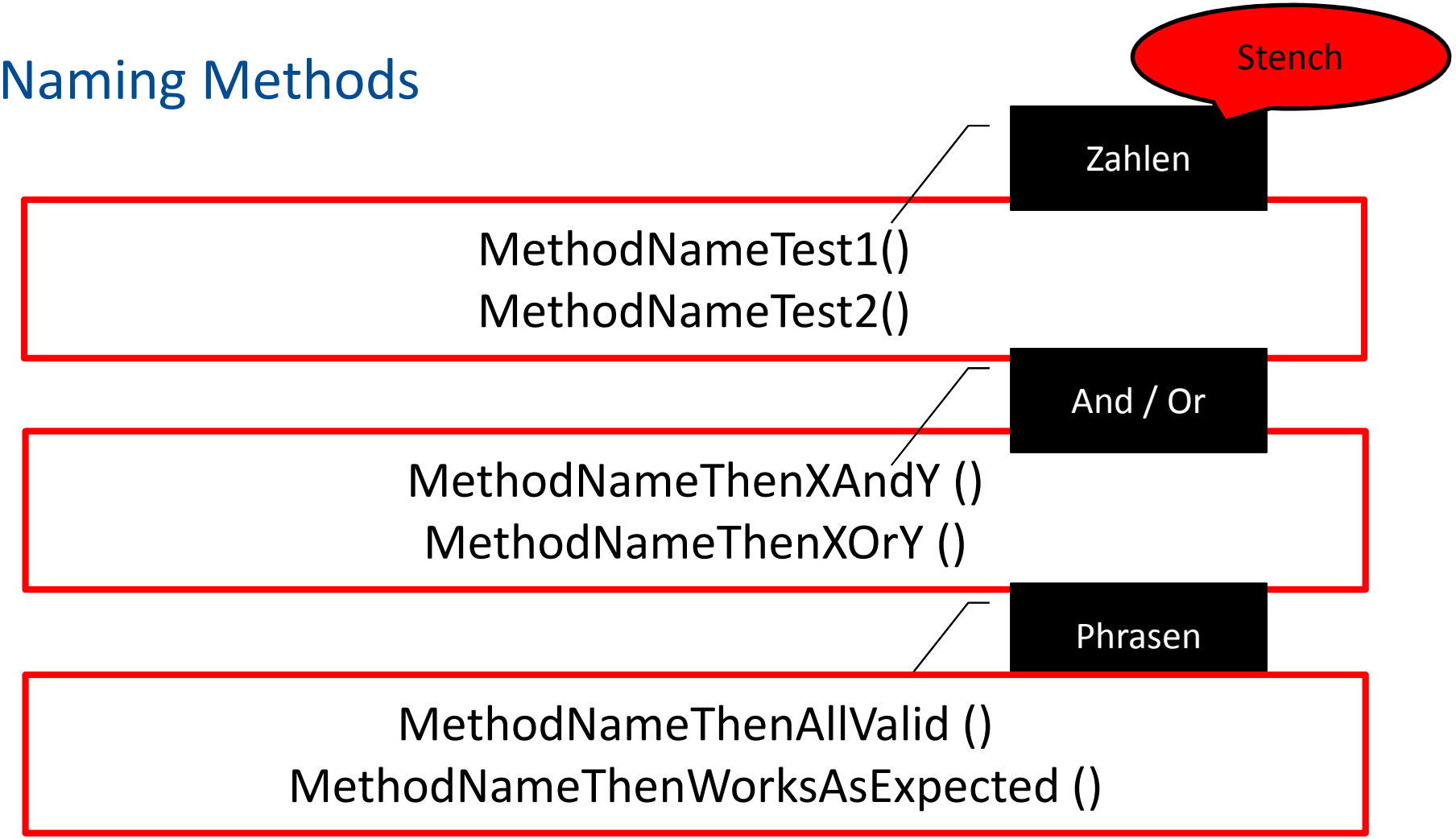
```
[Test]
public void IsFinished_4ConnectedYellowDiscs_GameFinished()
{
        var cut = CreateTestData();
        Assert.That(cut.IsFinished(), Is.True);
}

[Test]
public void WhenYellowHas4ConnectedDiscsThenGameFinished()
{
        var cut = CreateTestData();
        Assert.That(cut.IsFinished(), Is.True);
}
```

Naming Methods

Pattern



Naming Methods



andrena
OBJECTS

MAGIC VALUES

Magic Values

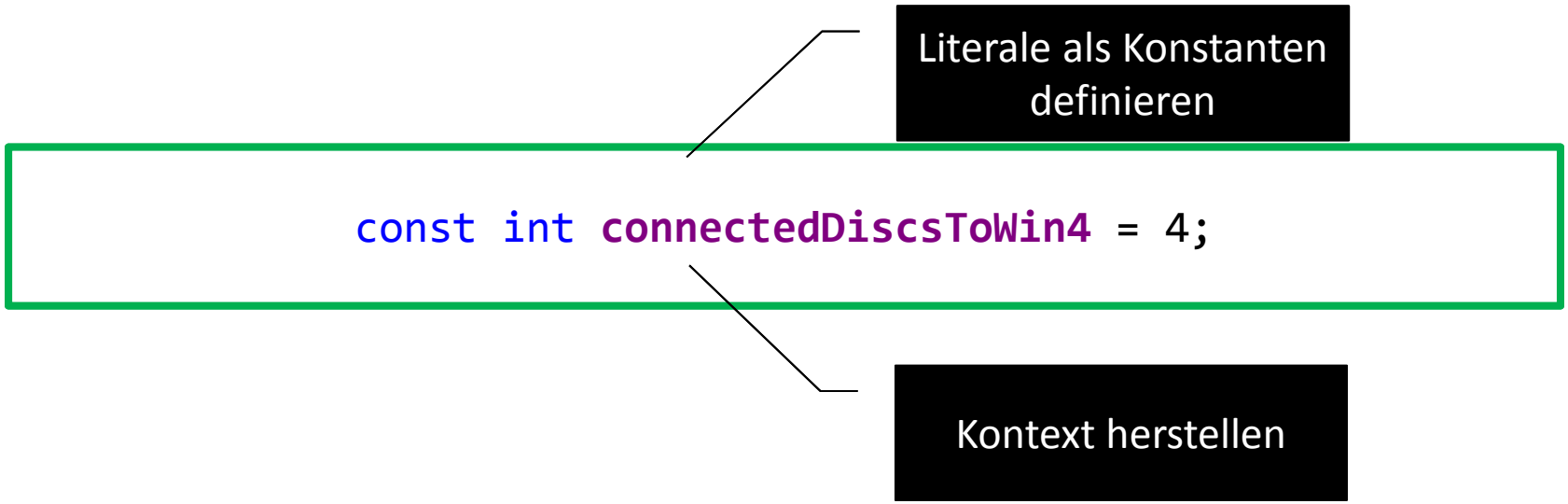
```
[SetUp]
public void SetUp()
{
        _game = new Game(7, 7, 4);
}
```

Magic Values

```
private const int ConnectedDiscsToWin4 = 4;
private const int Cols7 = 7;
private const int Rows7 = 7;

[SetUp]
public void SetUp()
{
      _game = new Game(Cols7, Rows7, ConnectedDiscsToWin4);
}
```

Magic Values

Pattern



andrena
OBJECTS

MULTIPLE ASPECTS PER TEST

Multiple Aspects per Test

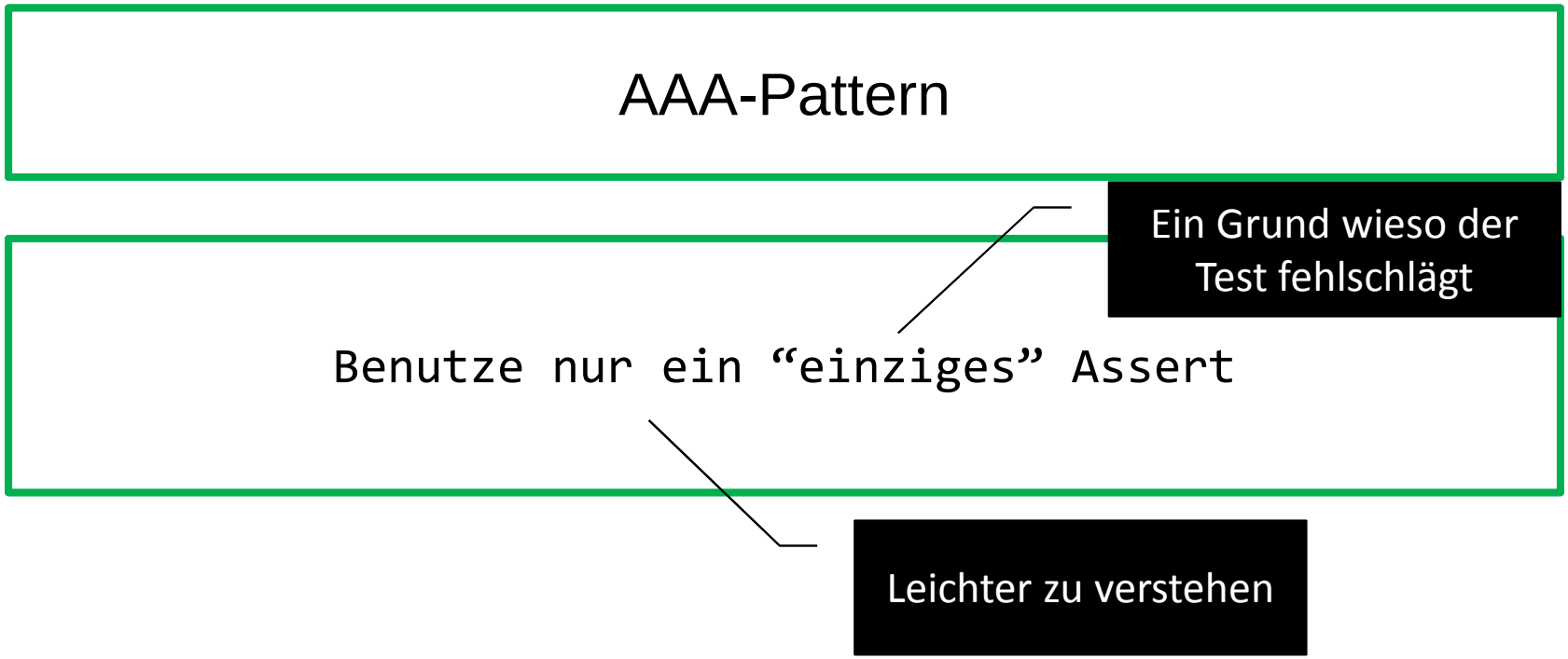
```
[Test]
public void Drop_Column1_NotFinished()
{
    _game.Drop(new Disc(Color.Red), 1);
    Assert.That(_game.IsFinished(), Is.False);
    Assert.That(_game.GetWinner(), Is.EqualTo(Color.Undefined));
}
```

Multiple Aspects per Test

```
[Test]
public void Drop_Column1_NotFinished()
{
    _game.Drop(YellowDisc, 1);

     Assert.That(_game.IsFinished, Is.False);
}

[Test]
public void Drop_Column1_UndefinedWinner()
{
    _game.Drop(YellowDisc, 1);

    Assert.That(_game.GetWinner(), Is.EqualTo(Color.Undefined));
}
```

Multiple Aspects per Test

Pattern



INCREMENTAL TESTING

Incremental Testing

```
@Test
public void testIsFinished() throws Exception {
    grid.put(0, 0, RED);
    grid.put(4, 4, RED);
    assertThat(grid.isUpDiagonalComplete(0, RED), is(false));
    grid.put(2, 2, RED);
    assertThat(grid.isUpDiagonalComplete(0, RED), is(false));
    grid.put(1, 1, RED);
    assertThat(grid.isUpDiagonalComplete(0, RED), is(false));
    grid.put(3, 3, RED);
    assertThat(grid.isUpDiagonalComplete(0, RED), is(true));
}
```

Incremental Testing

```
@Test
public void isUpDiagonalComplete_EmptyGrid_NotComplete() throws Exception {
    assertThat(grid.isUpDiagonalComplete(0, RED), is(false));
}

@Test
public void isUpDiagonalComplete_OneRed_NotComplete() throws Exception {
    grid.put(0, 0, RED);
    boolean upDiagonalComplete = grid.isUpDiagonalComplete(0, RED);
    assertThat(upDiagonalComplete, is(false));
}
```

Incremental Testing

```
@Test
public void isUpDiagonalComplete_OneLowerThanComplete_NotComplete() throws
Exception {
    grid.put(0, 0, RED);
    grid.put(1, 1, RED);
    grid.put(2, 2, RED);
    assertThat(grid.isUpDiagonalComplete(0, RED), is(false));
}
@Test
public void isUpDiagonalComplete_4ConnectedReds_Complete() throws Exception {
    grid.put(0, 0, RED);
    grid.put(1, 1, RED);
    grid.put(2, 2, RED);
    grid.put(3, 3, RED);
    assertThat(grid.isUpDiagonalComplete(0, RED), is(true));
}
```

Incremental Testing

Pattern

AAA-Pattern

Nicht: AAAAAAAAA

Incremental Testing

Pattern

Grenzen testen:

Leer	Gefüllt	Vor der Grenze	Nach der Grenze

"Numquam ponenda est pluralitas sine necessitate"
– William of Ockham

SUPERFLUOUS ARRANGE

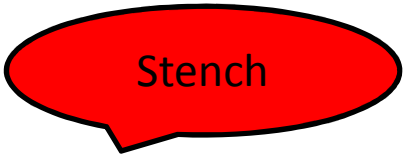
Superfluous Arrange

```
[Test]
public void DropTest1()
{
    var game = new Game();
    game.Drop(new Disc(Color.Yellow), 2);
    game.Drop(new Disc(Color.Red), 2);
    game.Drop(new Disc(Color.Yellow), 1);
    game.Drop(new Disc(Color.Red), 3);
    game.Drop(new Disc(Color.Yellow), 3);
    game.Drop(new Disc(Color.Red), 2);
    game.Drop(new Disc(Color.Yellow), 2);
    game.Drop(new Disc(Color.Red), 1);
    game.Drop(new Disc(Color.Yellow), 0);
    game.Drop(new Disc(Color.Red), 3);
    game.Drop(new Disc(Color.Yellow), 1);
    game.Drop(new Disc(Color.Red), 3);
    game.Drop(new Disc(Color.Yellow), 3);
    game.Drop(new Disc(Color.Red), 4);
    game.Drop(new Disc(Color.Yellow), 0);
    Assert.That(game.IsFinished(), Is.True);
}
```

Superfluous Arrange

```
[Test]
public void WhenDrop4InARowThenGameIsFinished()
{
    var game = new Game();
    game.Drop(new Disc(Color.Yellow), 0);
    game.Drop(new Disc(Color.Yellow), 1);
    game.Drop(new Disc(Color.Yellow), 2);
    game.Drop(new Disc(Color.Yellow), 3);

    Assert.That(game.IsFinished(), Is.True);
}
```

Superfluous Arrange



Was soll getestet werden?

```
public void DropTest1()
```

Was wird getestet?

```
game.Drop(new Disc(Color.Yellow), 2);
game.Drop(new Disc(Color.Red), 2);
game.Drop(new Disc(Color.Yellow), 1);
game.Drop(new Disc(Color.Red), 3);
…
```

Superfluous Arrange

Pattern

Was möchte ich testen

```
public void WhenDrop4InARowThenGameIsFinished()
```

Arrange enthält genau das – nicht mehr

```
game.Drop(new Disc(Color.Yellow), 0);
game.Drop(new Disc(Color.Yellow), 1);
game.Drop(new Disc(Color.Yellow), 2);
game.Drop(new Disc(Color.Yellow), 3);
```

andrena
OBJECTS

DELEGATED ARRANGE ACT

Delegated Arrange-Act

```
[Test]
public void DropTest1()
{
        var game = DataMock.CreateTestData();
        Assert.That(game.IsFinished(), Is.True);
}
```

Delegated Arrange-Act

```
[Test]
public void WhenDrop4InARowThenGameIsFinished()
{
        var game = new Game();
        game.Drop(new Disc(Color.Yellow), 0);
        game.Drop(new Disc(Color.Yellow), 1);
        game.Drop(new Disc(Color.Yellow), 2);

        game.Drop(new Disc(Color.Yellow), 3);

        Assert.That(game.IsFinished(), Is.True);
}
```

Delegated Arrange-Act

Pattern

AAA-Pattern

Minimales Arrange

```
var game = new Game();
game.Drop(new Disc(Color.Yellow), 0);
game.Drop(new Disc(Color.Yellow), 1);
game.Drop(new Disc(Color.Yellow), 2);

game.Drop(new Disc(Color.Yellow), 3);

Assert.That(game.IsFinished(), Is.True);
```

Verletzen von DRY möglich

Act: Idealerweise ein Methodenaufruf

andrena
OBJECTS

EXCESSIVE STUB SETUP

Excessive Stub Setup

```
@Test
public void testGetWinnerIfRowComplete() throws Exception {
  when(grid.getCols()).thenReturn(8);
  when(grid.getRows()).thenReturn(8);
  when(grid.isColumnComplete(anyInt(),
    any(Color.class))).thenReturn(false);
  when(grid.isRowComplete(anyInt(),
    any(Color.class))).thenAnswer(completeInRow(1, RED));
  when(grid.isUpDiagonalComplete(anyInt(),
    any(Color.class))).thenReturn(false);
  when(grid.isDownDiagonalComplete(anyInt(),
    any(Color.class))).thenReturn(false);
  assertThat(game.getWinner(), equalTo(RED));
}
```

Excessive Stub Setup

```
@Test
public void testGetWinnerIfRowComplete() throws Exception {
  prepare(grid)
    .withColumns(8)
    .withRows(8)
    .withColumnComplete(false)
    .withUpDiagonalComplete(false)
    .withDownDiagonalComplete(false)
    .withRowComplete(completeInRow(1, RED));
  assertThat(game.getWinner(), equalTo(RED));
}
```

Excessive StubSetup

```
private GridBuilder prepare(Grid grid) {
  return new GridBuilder(grid);
}

private static class GridBuilder {
  private Grid grid;

  public GridBuilder(Grid grid) {
    this.grid = grid;
  }

  public GridBuilder withRows(int rows) {
    when(grid.getRows()).thenReturn(rows);
    return this;
  }
  …
}
```

andrena OBJECTS

Excessive Stub Setup


```
when(grid.getCols()).thenReturn(8);
```

Prozeduraler Style

andrena OBJECTS

Excessive Stub Setup

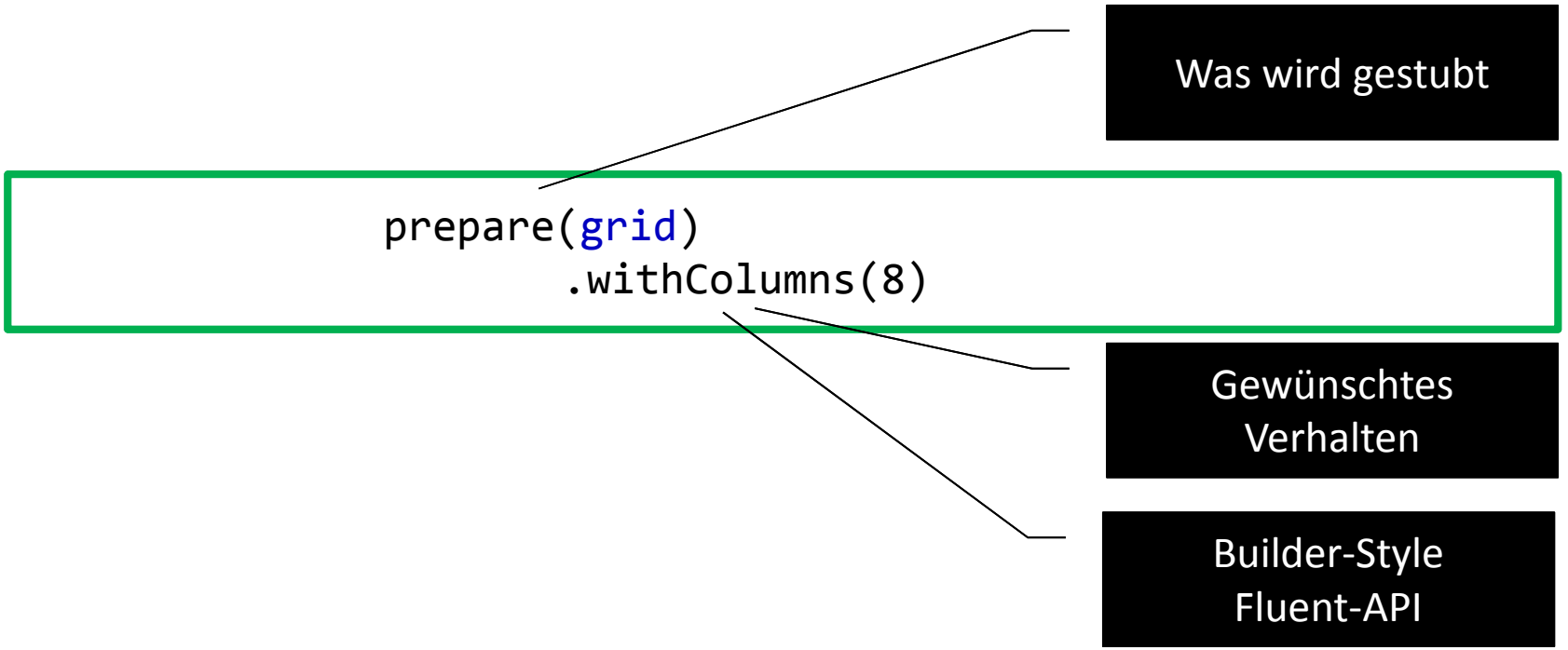
Smell

```
when(grid.isRowComplete(anyInt(), any(Color.class)))
        .thenAnswer(completeInRow(1, RED));
```

Boilerplate

Excessive StubSetup

Pattern



andrena
OBJECTS

CONDITIONS AND LOOPS

Conditions and Loops

```
@Test
public void testIsFinished() throws Exception {
  for (int i = 0; i < 5; i++) {
    grid.put(i, 0, YELLOW);
  }
  for (int j = 0; j < 5; j++) {
    if (j == 0) {
      assertThat(grid.isRowComplete(j, YELLOW), is(true));
    } else {
      assertThat(grid.isRowComplete(j, YELLOW), is(false));
    }
  }
}
```

Conditions and Loops

```
@Test
public void testRowCompleteIf4InARow() throws Exception {
  grid.put(0, 0, YELLOW);
  grid.put(1, 0, YELLOW);
  grid.put(2, 0, YELLOW);
  grid.put(3, 0, YELLOW);
  assertThat(grid.isRowComplete(0, YELLOW), is(true));
}
@Test
public void testRowCompleteIf4InOtherRow() throws Exception {
  grid.put(0, 0, YELLOW);
  grid.put(1, 0, YELLOW);
  grid.put(2, 0, YELLOW);
  grid.put(3, 0, YELLOW);
  assertThat(grid.isRowComplete(1, YELLOW), is(false));
}
```

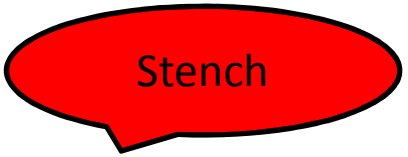
Conditions and Loops

Stench

Zwei Testfälle

```
if (j == 0) {
  assertThat(grid.isRowComplete(j, YELLOW), is(true));
} else {
  assertThat(grid.isRowComplete(j, YELLOW), is(false));
}
```

Conditions and Loops



Algorithmischer Style

```
for (int i = 0; i < 5; i++) {
    grid.put(i, 0, YELLOW);
}
```

Conditions and Loops

Pattern

Expliziter Style

```
grid.put(0, 0, YELLOW);
grid.put(1, 0, YELLOW);
grid.put(2, 0, YELLOW);
grid.put(3, 0, YELLOW);
grid.put(4, 0, YELLOW);
```

andrena
OBJECTS

IGNORE SETUP

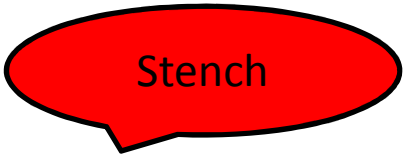
Ignore setup

```
[SetUp]
public void SetUp()
{
        _game = CreateTestData();
}
[Test]
public void IsFinishedTestRed()
{
        var game = new Game();
        game.Drop(RedDisc, 1);
        …
}
[Test]
public void IsFinishedTestYellow()
{
        _game.Drop(YellowDisc, 1);
        …
}
```

Ignore setup

```
[Test]
public void IsFinishedTestRed()
{
        var game = new Game();
        game.Drop(RedDisc, 1);
        …
}
[Test]
public void IsFinishedTestYellow()
{
        var game = CreateTestData();
        game.Drop(YellowDisc, 1);
        …
}
```

Ignore setup



Globales und lokales Setup

```
_game = CreateTestData();
…
var game = new Game();
```

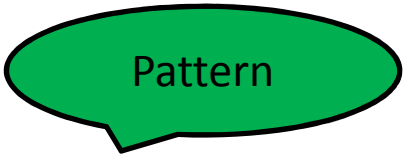
Ignore setup

Smell

Globales Deklaration und lokales Setup

```
_game = new Game();
```

Ignore setup



Factory-Methode

```
var game = CreateGameWith3RedInARow();
```

andrena
OBJECTS

C4 RELOADED

C4-Fragen helfen beim Testen:

- Concrete: geht es konkreter?
- Concise: geht es kürzer/knapper?
- Compact: geht es kompakter/dichter?
- Consequent: geht es konsequenter/systematischer?

- Wenn ja => Refactoring

Kenne die Systeme, denen du folgst:

- AAA Pattern
- Nur Grenzfälle testen
- Nur einen Zustand prüfen
- Naming Pattern für Test-Methoden
- Minimale Fixtures (an richtiger Stelle)

Fragen?

Vielen Dank fürs Mitmachen

1. OG DIREKT ÜBER DEM EMPFANG