



IT WORKS – WE CARE

TECHTALK



Scriptbasierte Testautomatisierung für Web-Anwendungen



Scriptbasierte Testautomatisierung

- + Web-Anwendung: Erstes Einsatzgebiet, Ergebnisse aber allgemein übertragbar
- + Test aus Benutzersicht
- Nicht Unit-Test, Integrationstest



Überblick

- Motivation
 - Evolution der TT-Testautomatisierung
 - Gewonnene Erkenntnisse
 - Das Ergebnis
- Not a silver bullet, but for us IT WORKS!



Motivation

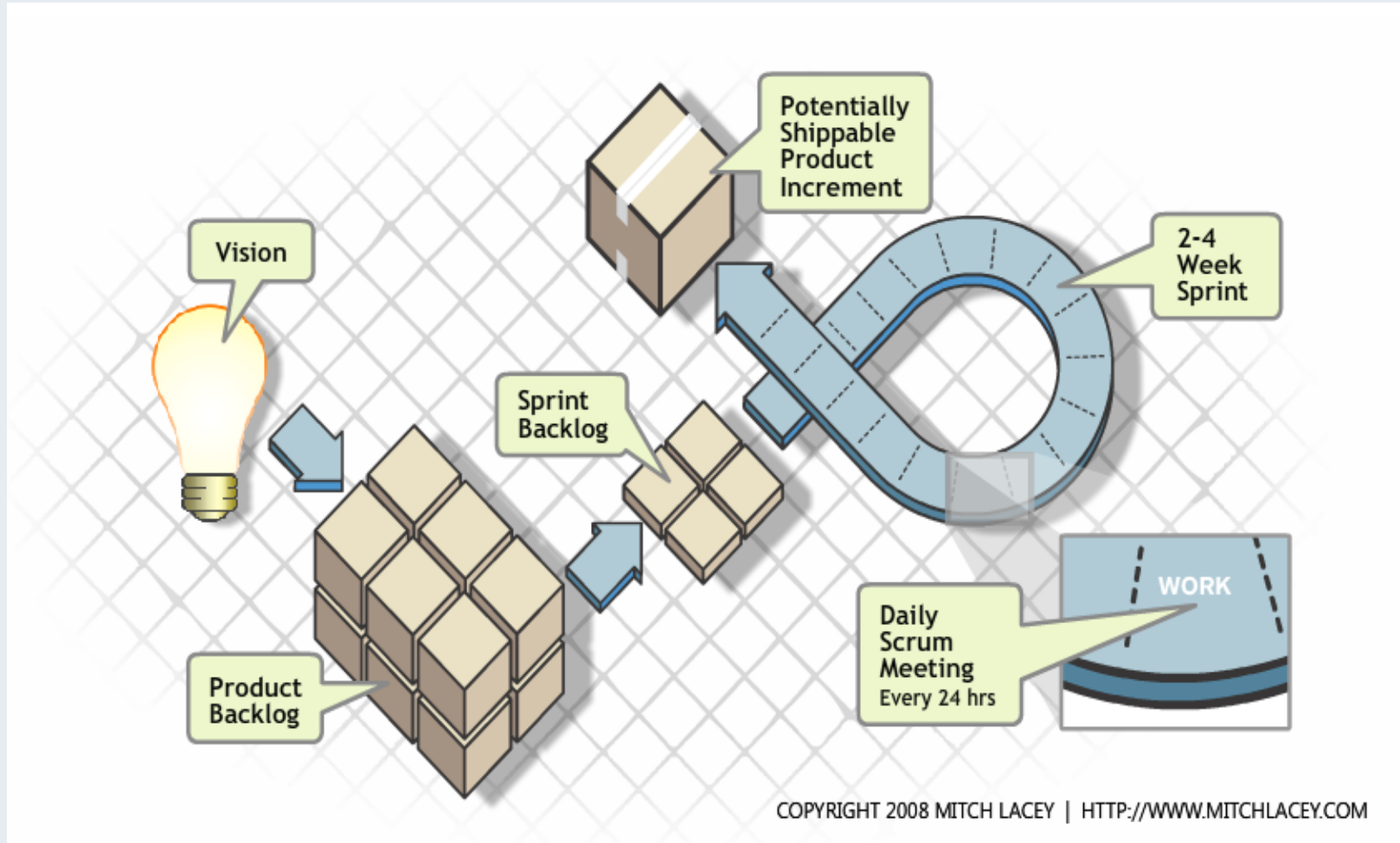
- **TT-Vorgehensmodell basiert auf SCRUM**
 - Laufend Erweiterung, Änderung und Refactoring
 - Regressionstests zur Absicherung
 - Keine Front-Up Analyse, Details im Sprint Planning
 - Konsistenz von Requirements Document und Software
- **TDD on a higher level: BDD**
 - Vorteile von Test Driven Development auch auf Systemtest-Ebene nutzen



IT WORKS - WE CARE

TECHTALK

Der Auslöser...: SCRUM!!!

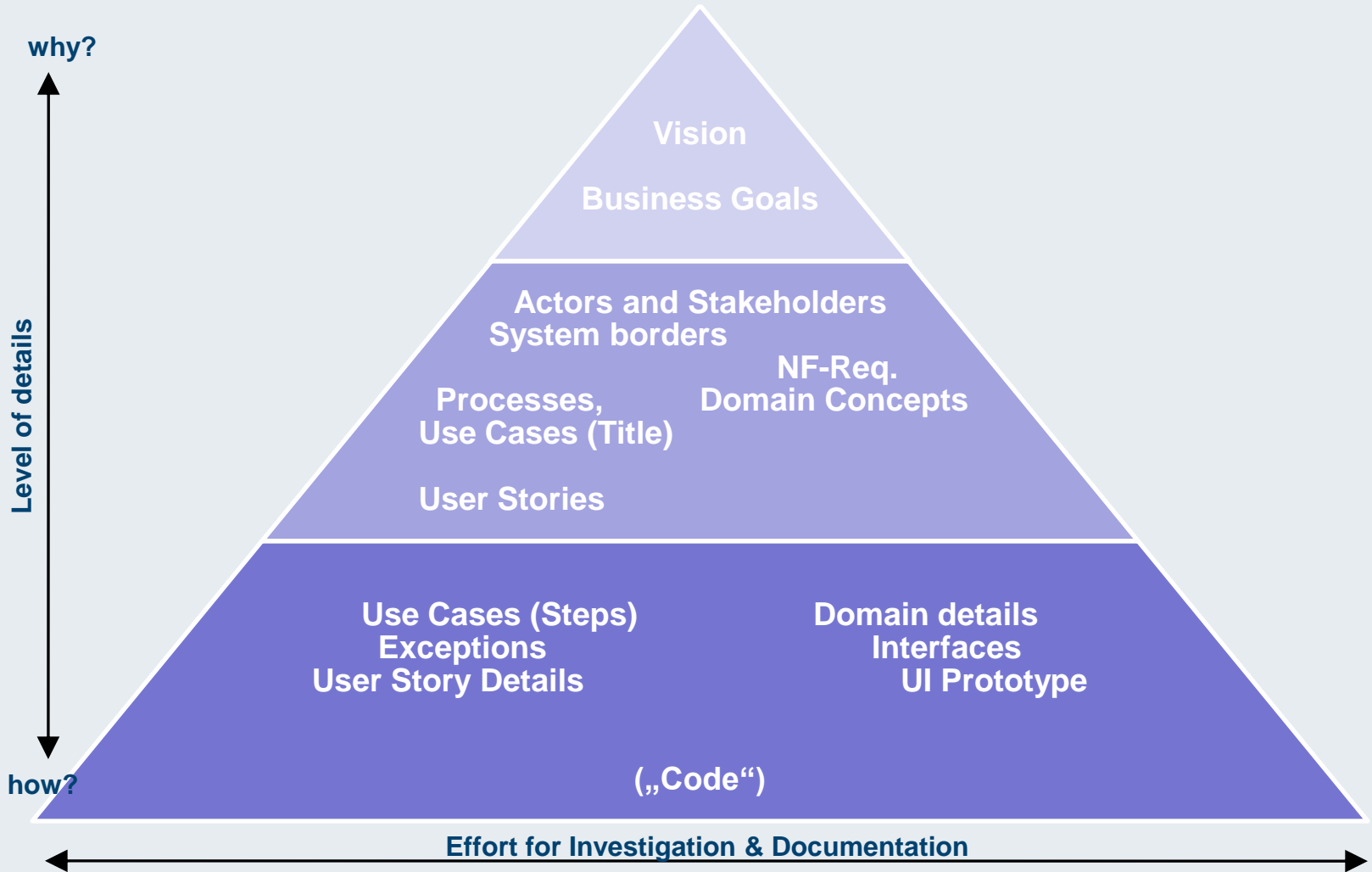




IT WORKS - WE CARE

TECHTALK

What to test???

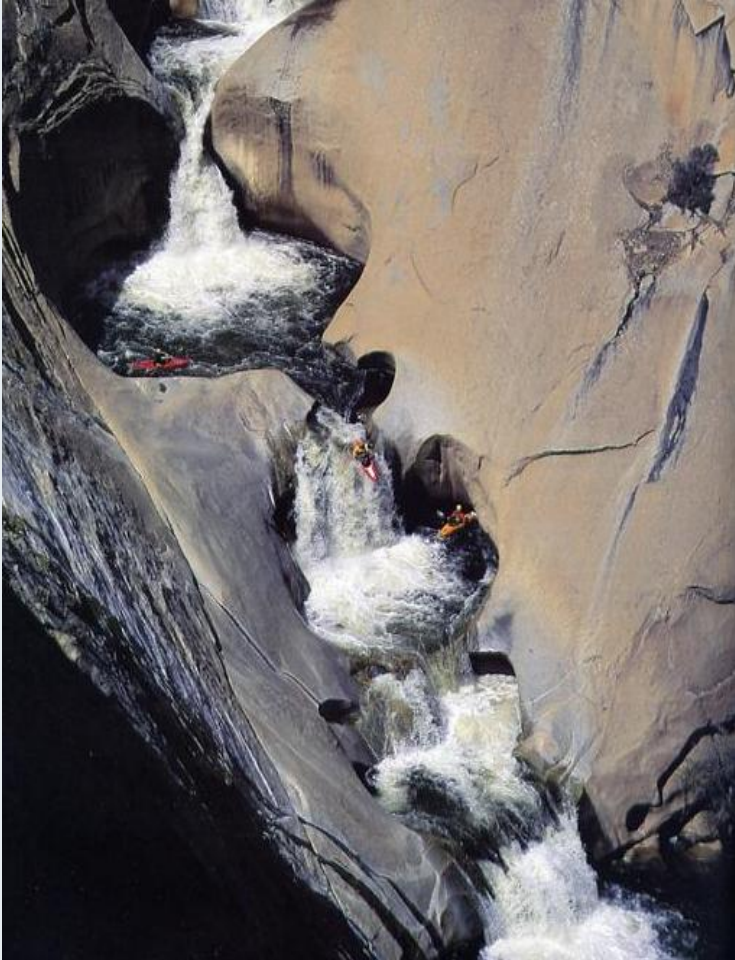




IT WORKS - WE CARE

TECHTALK

Was ändert SCRUM am Testen?



Zwei Hauptaspekte:

- Regressionstests
- Requirements ongoing
 - Dokumentation?
 - Konsistenz?



IT WORKS - WE CARE

TECHTALK

TDD - Test Driven Development

Write a
failing
unit test



Make
the test
pass



refactor





IT WORKS - WE CARE

TECHTALK

Auswirkungen von TDD

- 15-35% Zusatzaufwand in der Entwicklung
- + 80-90% weniger Aufwand bei Änderung
- + 40-90% weniger Fehler

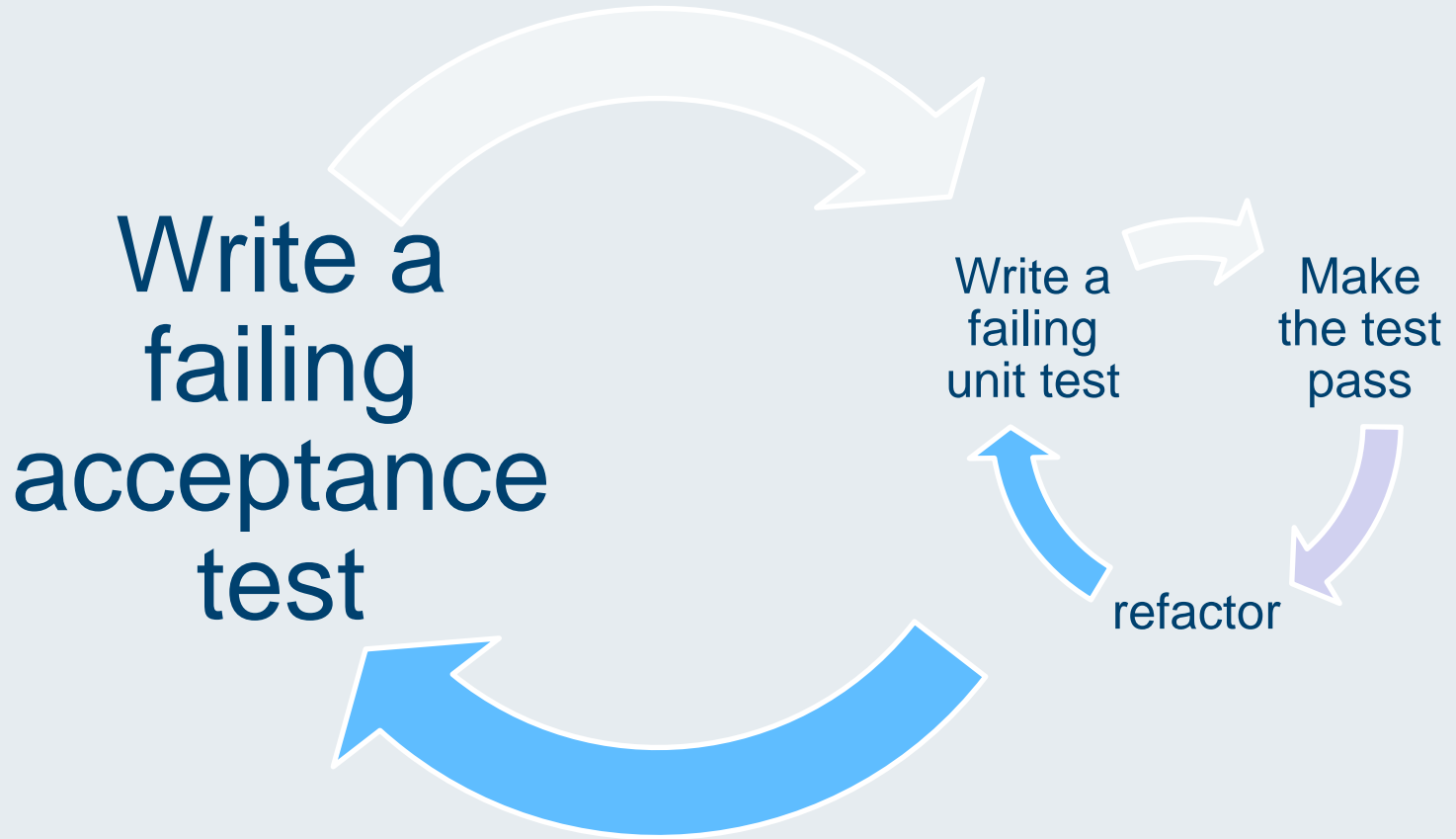
Quelle: <http://www.infoq.com/news/2009/03/TDD-Improves-Quality>



IT WORKS - WE CARE

TECHTALK

BDD—Behaviour Driven Development





Automatische Tests - Nutzen

- Absicherung bei Änderungen
- Zeitnahes Feedback für Entwickler
- Test- und Behaviour Driven Development
- Beschreibung von Verhalten

Hey,
Mr.Customer
is that what
you want???



Hey, R2D2,
does it still
work???





Requirements == Tests

Traditionally Requirements: at the beginning, tests at the end

Both: Describe, how the system is used as formality increases, tests and requirements become indistinguishable requirements are equivalent.

Equivalence hypothesis, Robert C. Martin



IT WORKS - WE CARE

TECHTALK

Business WriteableTest scripts

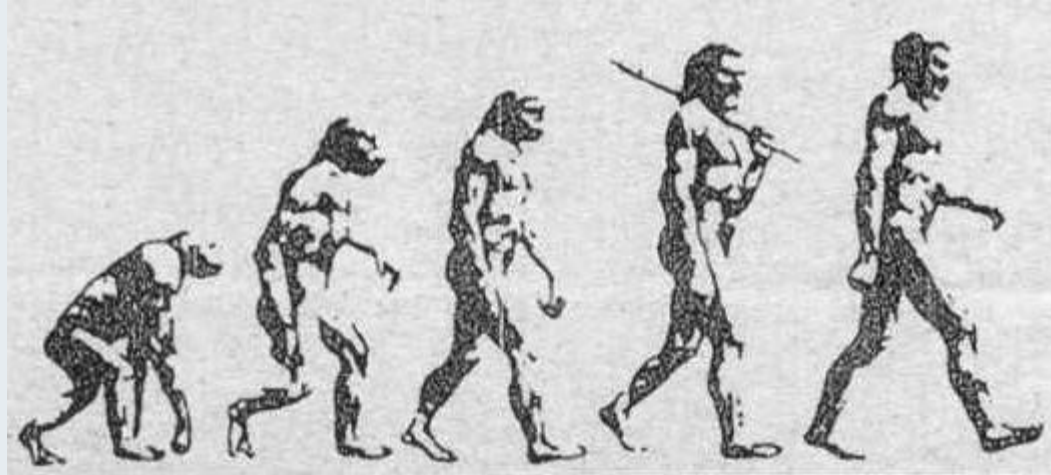
- Ermöglicht BDD: Write Acceptance Test first!
- Dokumentation des Verhaltens (Detail-Spec)
- Automatisch ausführbar: Verifiziert Verhalten



IT WORKS - WE CARE

TECHTALK

Evolution der Testautomatisierung



- Unit Test Framework
 - Capture and Replay
 - Browser Control via Unit-Test
 - Keyword / Script driven



Unit Test Framework

```
[TestMethod]
public void SetGFDatenTest2()
{
    Geschaeftsfall gf = GetNeuenFall();
    tempObjects.Add(gf);

    GeschaeftsfallDatenParameter param = new GeschaeftsfallDatenParameter();
    param.KatastralGemeindeNummer = 4002;
    param.AnkaufsFoerderung = 2000m;
    param.Ausleihungssumme = 20000;
    param.BauvollendungsfristVerlaengert = new DateTime(2010,1,1);
    param.Denkmalschutz = true;
    param.VersicherungAdresse = "1010 Wien, Teststrasse 4711";
    param.VersicherungName = "TestVersicherung";

    ProtokollierungDetailParameter detailParameter = new ProtokollierungDetailParameter(
    detailParameter.Einreichdatum = IPDateTime.Today.AddDays(-1);
    detailParameter.StandortEinreichung = StandOrt.QueryProvider.GetAll().GetItem(0);
    detailParameter.ZeitSchrift = true;
    detailParameter.AntragsArt = KommunikationsArt.Post;
    detailParameter.Modellart = ModellartEnum.IP;

    param.DetailParameter = detailParameter;
    GeschaeftsfallServices.Instance.UpdateGeschaeftsfallDaten(gf, param, new List<Einlag

    Assert.AreEqual(4002, gf.GebietsInformation.KatastralGemeindeNummer);
    Assert.AreEqual(param.AnkaufsFoerderung, gf.AnkaufFoerderung);
    Assert.AreEqual(param.Ausleihungssumme, gf.Ausleihungssumme);
    Assert.AreEqual(param.BauvollendungsfristVerlaengert, gf.BauvollendungsfristVerlaenge
    Assert.AreEqual(param.Denkmalschutz, gf.Denkmalschutz);
    Assert.AreEqual(param.VersicherungAdresse, gf.VersicherungAdresse);
    Assert.AreEqual(param.VersicherungName, gf.VersicherungName);
```



Unit Test Framework

- Vorteile:
 - + Tool + Know-How dazu schon vorhanden
 - + Hohe Wartbarkeit (bei gut gewählter Struktur...)
 - + Versions- u. Variantenkontrolle in TFS
- Nachteil:
 - Kein Kommunikationsmedium
 - Keine UI-Logik testbar
 - Entwickler für Test-Code notwendig



UI-Capture & Replay (z.B. Selenium):

The screenshot shows a web application interface with a form titled "Geeignete Person erfassen". The form is divided into several sections: "Person", "Wohnadresse", "Aktiv", and "Sonstiges". The "Person" section contains fields for Nachname (Magreither), Vorname (Paul), Geburtsdatum (14.07.1975), and Titel. The "Wohnadresse" section contains Strasse (Landstraße 37), PLZ (3500), Ort (Krems an der Donau), and Bildungspassnummer (BPN-123-1578). The "Aktiv" section has a checkbox for "Aktiv" which is checked. The "Sonstiges" section has a text area for "Information über Vertrauenswürdigkeit" and "Bemerkungen". There are "Bearbeiten" and "Löschen" buttons at the top and bottom of the form.

The screenshot shows the Selenium IDE interface. The Base URL is "http://vpc-mdbui/". The "Table" view shows a list of recorded actions:

Command	Target	Value
open	/noel.mdb.test/ind...	
click	link=Sachbereichsl...	
click	link=Suche Strg+S...	
select	ctl00_Content_sear...	label=Person
click	//option[@value='...	
type	ctl00_Content_pers...	Magreither
click	ctl00_Content_ctl0...	
click	link=Magreither	

The "click" command with target "link=Suche Strg+Shift+S" is selected. The "Command" dropdown is set to "click", the "Target" dropdown is set to "link=Suche Strg+Shift+S", and the "Value" field is empty. The "Find" button is visible.

The "Log" view shows the following entry:

```
click(locator)
Arguments:
  • locator - an element locator
Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.
```



UI-Capture & Replay (z.B. Selenium):

- Vorteile:
 - + UI-Logik auch testbar
 - + Open Source
- Nachteil:
 - Komplexe Testfälle (mehrere Screens):
Never-ever try this at home!!! ;-)
- Kombination mit Unit-Test Framework:
 - Capture-Tool für einzelne Testschritte



Cucumber

<http://cukes.info/>

WIKI

Detailed documentation

EXAMPLES

Use your mother tongue

TUTORIALS

By the community

LIGHTHOUSE

Issue tracker

MAILING LIST

Ask questions

IRC

Get instant help



Cucumber

Behaviour Driven Development
with elegance and joy

1: Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

Scenario: Add two numbers
  Given I have entered 50 into the calculator
  And I have entered 70 into the calculator
  When I press add
  Then the result should be 120 on the screen
```

2: Write a step definition

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

3: Run and watch it fail

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # features/additi
    Given I have entered 50 into the calculator # features/step_d
      uninitialized constant Calculator (NameError)
      ./features/step_definitions/calculator_steps.rb:2:in "Given /
      features/addition.feature?:in "Given I have entered 50 into
    And I have entered 70 into the calculator # features/step_d
    When I press add # features/additi
    Then the result should be 120 on the screen # features/additi
```

4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # features/additi
    Given I have entered 50 into the calculator # features/step_d
    And I have entered 70 into the calculator # features/step_d
    When I press add # features/additi
    Then the result should be 120 on the screen # features/additi
```

6. Repeat 2-5 until green like a cucumber

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # features/additi
    Given I have entered 50 into the calculator # features/step_d
    And I have entered 70 into the calculator # features/step_d
    When I press add # features/step_d
    Then the result should be 120 on the screen # features/step_d
```

7. Repeat 1-6 until the money runs out



Cucumber

1: Describe behaviour in plain text

Feature: Addition

In order to avoid silly mistakes

As a math idiot

I want to be told the sum of two numbers

Scenario: Add two numbers

Given I have entered 50 into the calculator

And I have entered 70 into the calculator

When I press add

Then the result should be 120 on the screen



Cucumber

2: Write a step definition

```
Given /I have entered (.*) into the calculator/ do |n|  
  calculator = Calculator.new  
  calculator.push(n.to_i)  
end
```



Cucumber

3: Run and watch it fail

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # features/additi
    Given I have entered 50 into the calculator # features/step_d
      uninitialized constant Calculator (NameError)
      ./features/step_definitons/calculator_steps.rb:2:in `Given /
      features/addition.feature:7:in `Given I have entered 50 into
    And I have entered 70 into the calculator # features/step_d
    When I press add # features/additi
    Then the result should be 120 on the screen # features/additi
```



Cucumber

4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```



Cucumber

5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # features/additi
    Given I have entered 50 into the calculator # features/step_d
    And I have entered 70 into the calculator # features/step_d
    When I press add # features/additi
    Then the result should be 120 on the screen # features/additi
```



Cucumber

6. Repeat 2-5 until green like a cuke

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # features/additi
    Given I have entered 50 into the calculator # features/step_d
    And I have entered 70 into the calculator # features/step_d
    When I press add # features/step_d
    Then the result should be 120 on the screen # features/step_d
```



IT WORKS - WE CARE

TECHTALK

Cucumber goes .NET: Specflow



[specflow](#)

[downloads](#)

[about](#)

BINDING BUSINESS REQUIREMENTS TO .NET CODE

SpecFlow aims at bridging the communication gap between domain experts and developers by binding business readable behavior specifications to the underlying implementation.

Our mission is to provide a pragmatic and frictionless approach to Acceptance Test Driven Development and Behavior Driven Development for .NET projects today.

SpecFlow is open source (BSD License).



Specflow

Define your scenarios in plain-text:

ScoreCalculation.feature

Feature: Score Calculation

In order to know my performance

As a player

I want the system to calculate my total score

Scenario: Gutter Game

Given a new bowling game

When all of my balls are landing in the gutter

Then my total score should be 0

Scenario: All Strikes

Given a new bowling game

When all of my rolls are strikes

Then my total score should be 300



Specflow

... and let SpecFlow execute them:

The screenshot shows the SpecFlow test runner interface for 'Unit Test Sessions - Session #1'. The interface includes a toolbar with various icons for navigation and execution. Below the toolbar, a summary bar indicates 'Tests failed: 1, passed: 1, ignored: 0'. The test tree shows a hierarchy starting with '<Bowling.Specflow> (2 tests)', followed by 'Bowling.Specflow (2 tests)', and then 'ScoreCalculationFixture (2 tests)'. Under 'ScoreCalculationFixture', two tests are listed: 'GutterGame' (passed) and 'AllStrikes' (failed). The 'GutterGame' test is expanded, showing its Gherkin-style description: 'Given a new bowling game', 'When all of my balls are landing in the gutter', and 'Then my total score should be 0'. The 'AllStrikes' test is also expanded, showing its Gherkin-style description: 'Given a new bowling game', 'When all of my rolls are strikes', and 'Then my total score should be 300'. Below the description, the error message is displayed: '-> error: Expected: 300' and 'But was: 330'.

ScoreCalculationFixture.GutterGame : Passed

```
Given a new bowling game
When all of my balls are landing in the gutter
Then my total score should be 0
```

ScoreCalculationFixture.AllStrikes : Failed

```
Given a new bowling game
When all of my rolls are strikes
Then my total score should be 300
-> error:   Expected: 300
          But was:  330
```

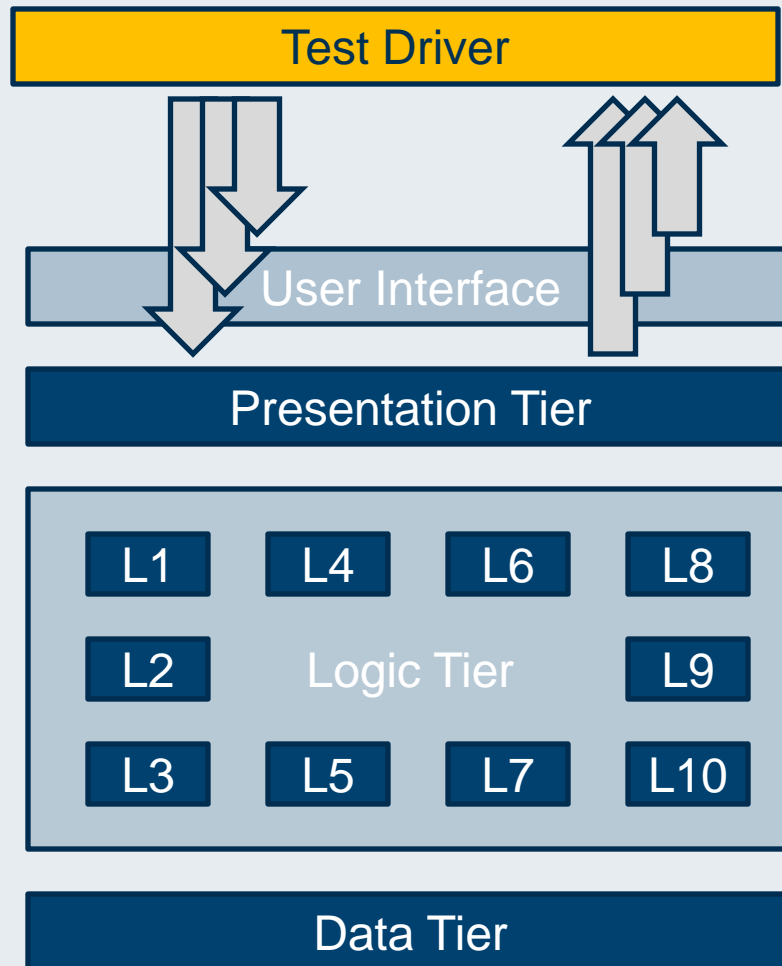


Cucumber (.NET: Specflow):

- Vorteile:
 - + Beschreibt Systemverhalten
 - + „Business Writeable“
 - + „Binding“: Beliebige Ebene (UI, Presenter, ...)
- Nachteil:
 - Testautomatisierung ist auch SW-Entwicklung...
(but keep in mind: Quality is free – if you spend much money on it.)



Ganz Gallien! – Wirklich GANZ...?? ;-)





IT WORKS - WE CARE

TECHTALK

Erkenntnisse: Do's and dont's

- UI oder nicht UI, das ist hier die Frage
- „kurze“ vs. „lange“ Testfälle



UI oder NICHT-UI, das ist die Frage...

- Wenn UI, dann..
 - Never-ever „lange“ Testfälle (Wartbarkeit)
 - Problem: Vorbedingungen schaffen?
 - Captured Script aus Unit-Test-Framework aufrufen
- Funktionalität im UI
 - Validierung, etc: Möglichst Vermeiden („Design 4 Testability“)
- Probleme am UI
 - Resultate verifizieren (Listen, ...)



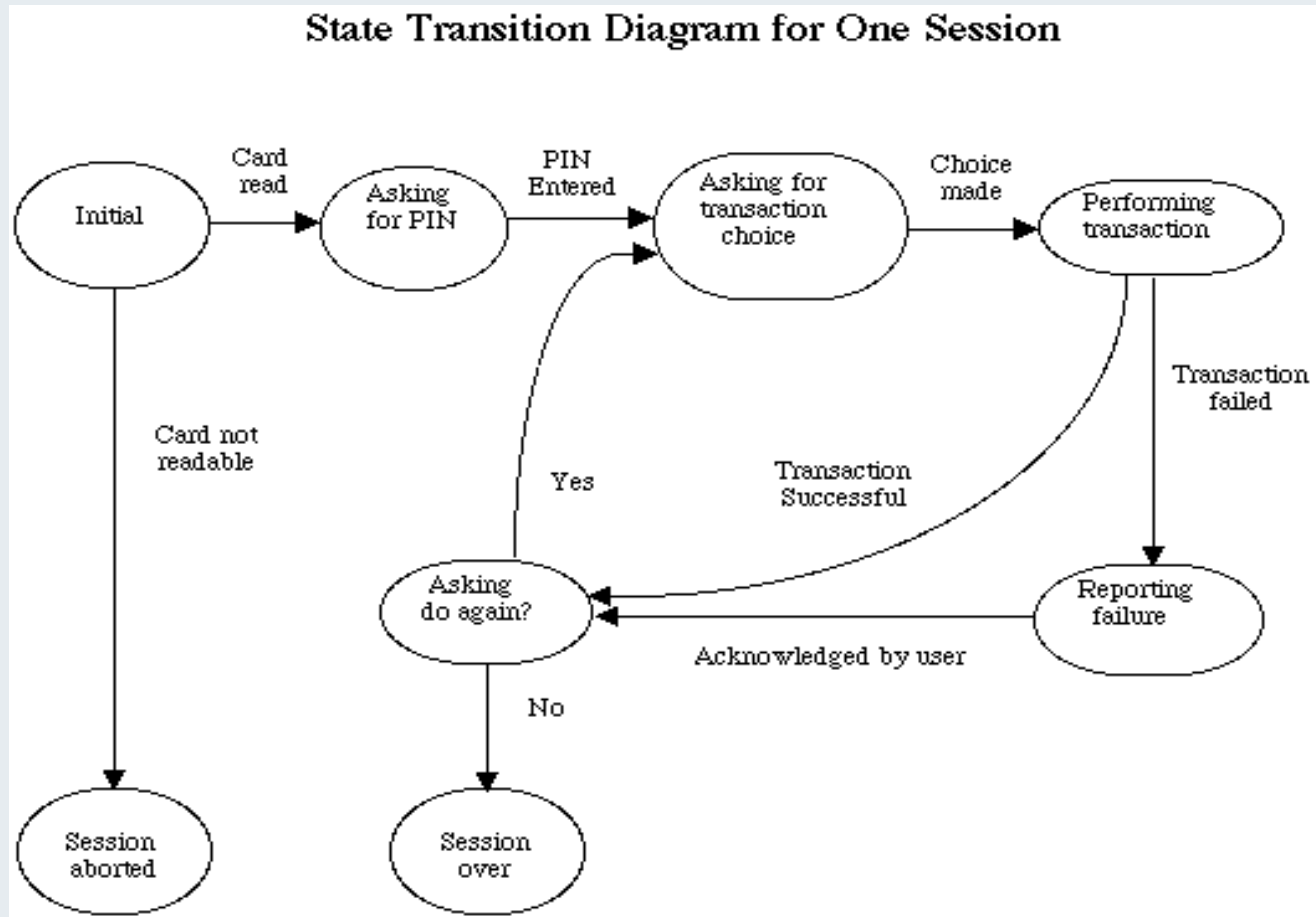
UI oder Nicht-UI, das ist die Frage...

- So viel wie nötig...
 - Client-seitige Validierung, etc
 - Eingabelänge, erlaubte Zeichen, etc
 - Optische Erscheinung, Usability
- ... aber so wenig wie möglich!
 - Testziel auch über Business-Logic (Daten-Variationen, Funktionalität)
 - Bessere Performance ohne UI-Autom.



„Kurze“ vs. „Lange“ Testfälle

State Transition Diagram for One Session



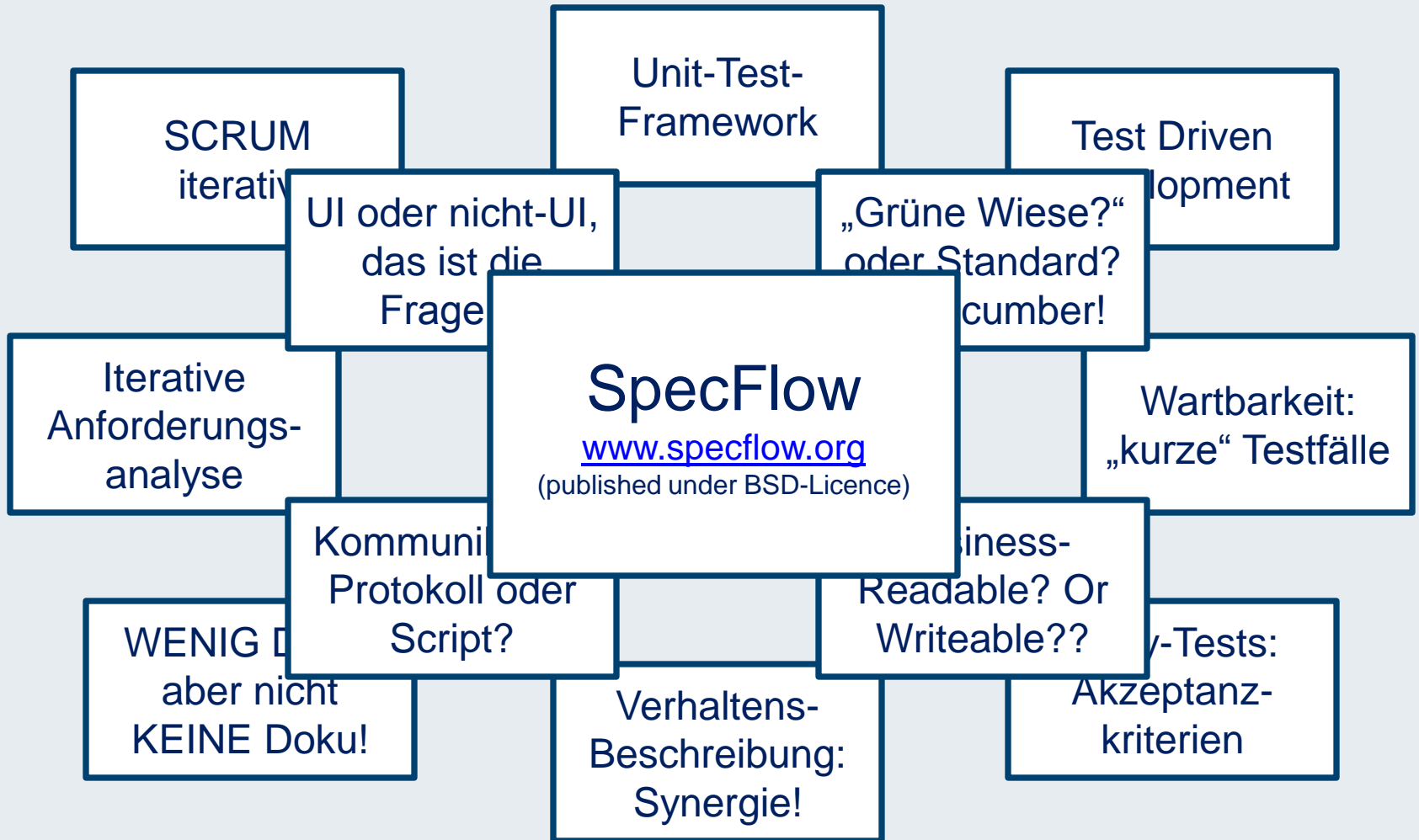


„Kurze“ vs. „Lange“ Testfälle

- „KURZ“
 - „single responsibility“
 - Traceability: Stories-Akzeptanzkriterien
 - Bessere Wartbarkeit
 - Anzahl: Geringerer Ordnung
- „LANG“
 - „Kombinatorische Explosion“
 - Höhere Änderungswahrscheinlichkeit



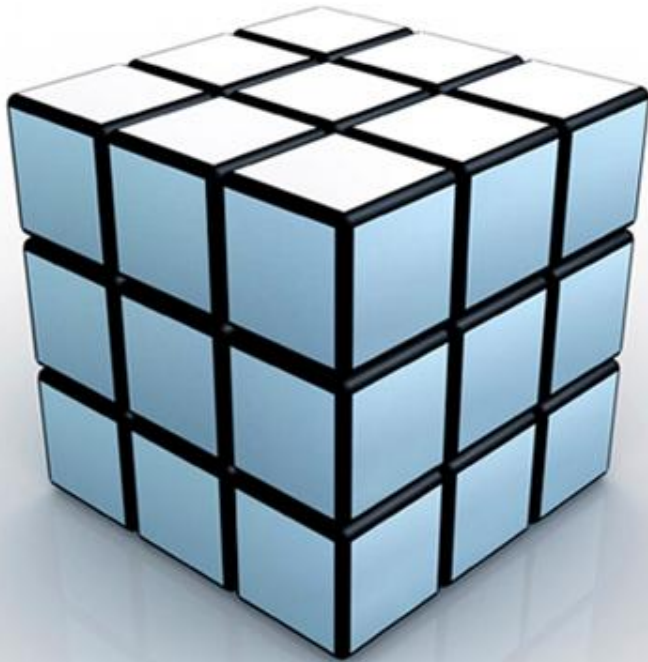
Kurz gesagt...





IT WORKS – WE CARE

TECHTALK



Herzlichen Dank!

als@techtalk.at