

Agiles Testen

Handwerkszeug zur Prävention von Fehlern
und technischen Schulden

Entwicklertag 2014

Lars Alvincz, Daniel Knapp

andrena
OBJECTS

Experts in agile software engineering

Agenda

- **Ziel dieses Vortrags**
- Grundzüge des agilen Testens
- Voraussetzungen für agiles Testen
- Vorstellung der agilen Testquadranten
- Die Testpyramide
- Anwendungsbeispiele für die Umsetzung agiler Tests
- Fazit, Fragen, Diskussion, Kontakt

Ziel dieses Vortrags

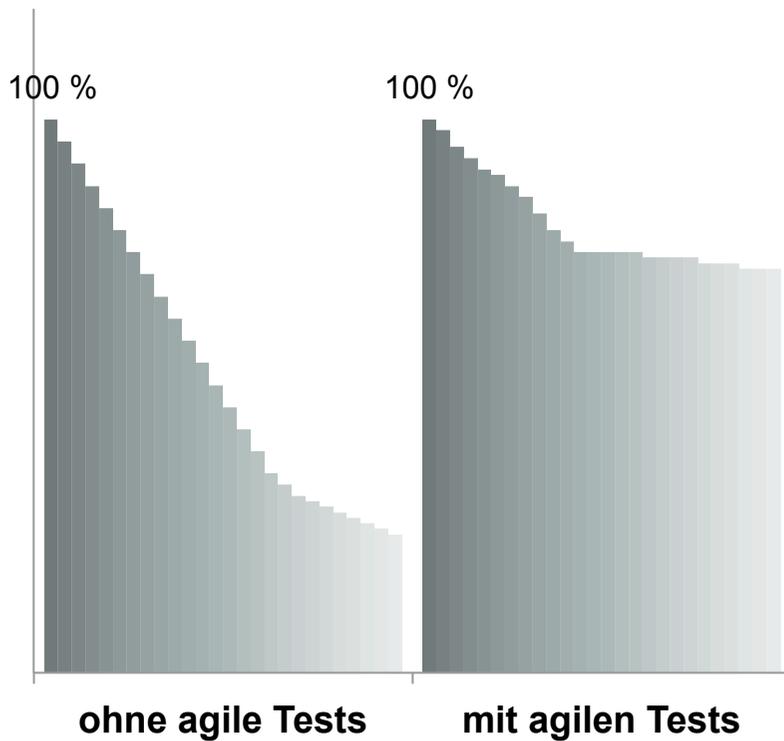
- Der Vortrag gibt eine grundlegende Übersicht, wie Entwicklungsteams von Anfang an dem Aufhäufen von technischen Schulden und Fehlern entgegenwirken können, indem sie konsequent **agile Testtechniken** im Entwicklungsprozess verankern
- Zunächst findet eine grundlegende Begriffsklärung zum agilen Testen statt
- Im Anschluss werden Anwendungsbeispiele verschiedener Aspekte vorgestellt

Agenda

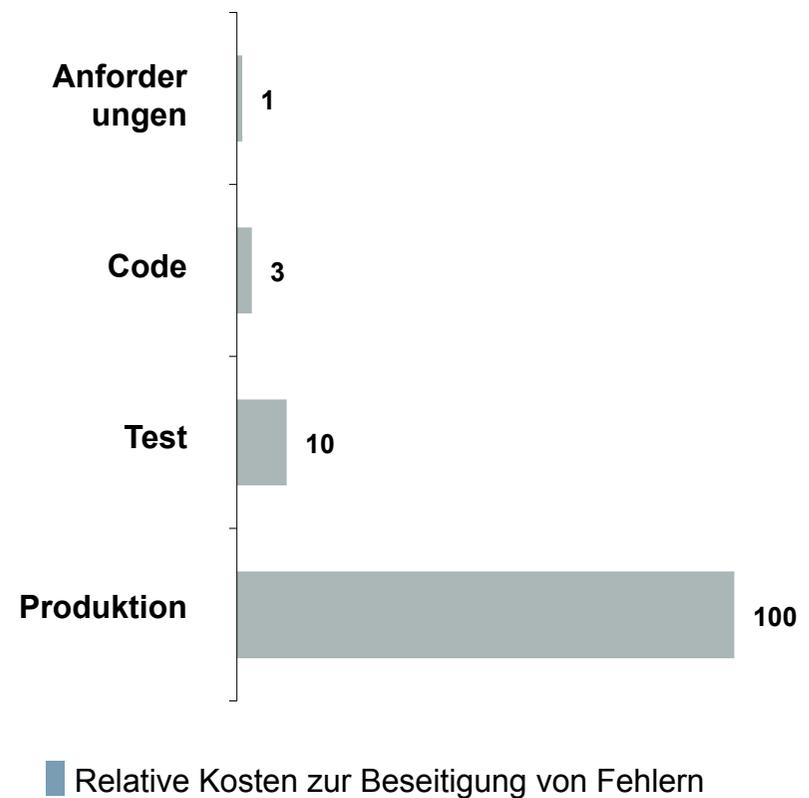
- Ziel dieses Vortrags
- **Grundzüge des agilen Testens**
- Voraussetzungen für agiles Testen
- Vorstellung der agilen Testquadranten
- Die Testpyramide
- Anwendungsbeispiele für die Umsetzung agiler Tests
- Fazit, Fragen, Diskussion, Kontakt

Vorüberlegungen

Kapazität für Features vs. Bugs



Relative Kosten zur Fehlerbeseitigung nach Lebensdauer



Grundzüge des agilen Testens I

Exkurs: agile Entwicklungsprozesse (Scrum etc.) verfolgen das Ziel

- **fertige Software** zum Ende des Entwicklungszyklus auszuliefern
- eine hohe **Entwicklungsgeschwindigkeit** bei effizientem Aufwand zu ermöglichen

Agiles Testen verfolgt den Ansatz

- Testen und Entwickeln **zeitlich und eng zu verzahnen**
- eine möglichst **hohe Testabdeckung** der Software zu erzeugen
- mangelhafte Codequalität bzw. **Bugs so früh als möglich offen** zu legen, um **frühzeitige Korrekturmaßnahmen** zu ermöglichen
- die entwickelte Funktionalität auch über den Entwicklungszyklus hinaus gegen Seiteneffekte bei Erweiterungen abzusichern.

Grundzüge des agilen Testens II

Zielbild

- Entwicklungsteam liefert am Ende des Entwicklungszyklus funktionierende Software ab, die im Anschluss produktiv genutzt werden kann
- Automatisierte Akzeptanztests sichern neben den Entwicklertests das bestehende Verhalten
- Ggf. erforderliche manuelle Tests werden nach einem schlanken Verfahren durchgeführt
- Das fertige Entwicklungssinkrement wird so zeitnah als möglich in die Produktion überführt

Agenda

- Ziel dieses Vortrags
- Grundzüge des agilen Testens
- **Voraussetzungen für agiles Testen**
- Vorstellung der agilen Testquadranten
- Die Testpyramide
- Anwendungsbeispiele für die Umsetzung agiler Tests
- Fazit, Fragen, Diskussion, Kontakt

Voraussetzungen für agiles Testen I

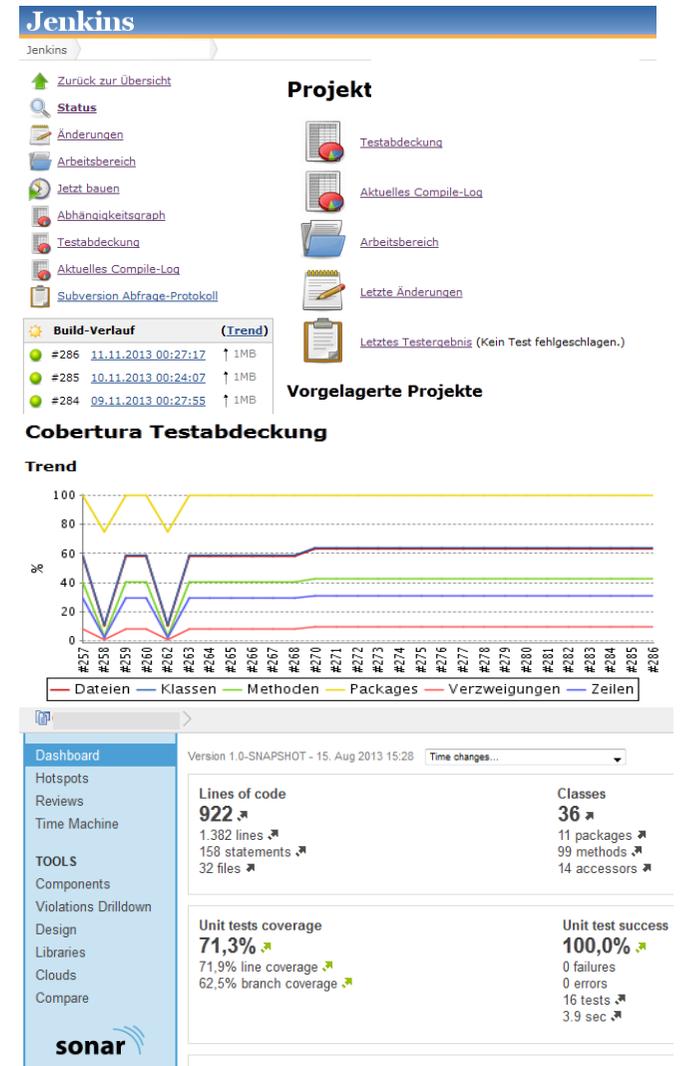
Agile Software Engineering (ASE) I

- Agile Entwicklungstechniken (**XP**, **Clean Code**) innerhalb des Entwicklungsteams
 - **automatisierte Tests** auf verschiedenen Ebenen
 - konsequente **Refactorings** zur Vermeidung technischer Schulden
- **Testisolationswerkzeuge** und -techniken einsetzen, um wiederholbare Tests zu produzieren und Testlaufzeiten zu verkürzen.

Voraussetzungen für agiles Testen II

Agile Software Engineering (ASE) II

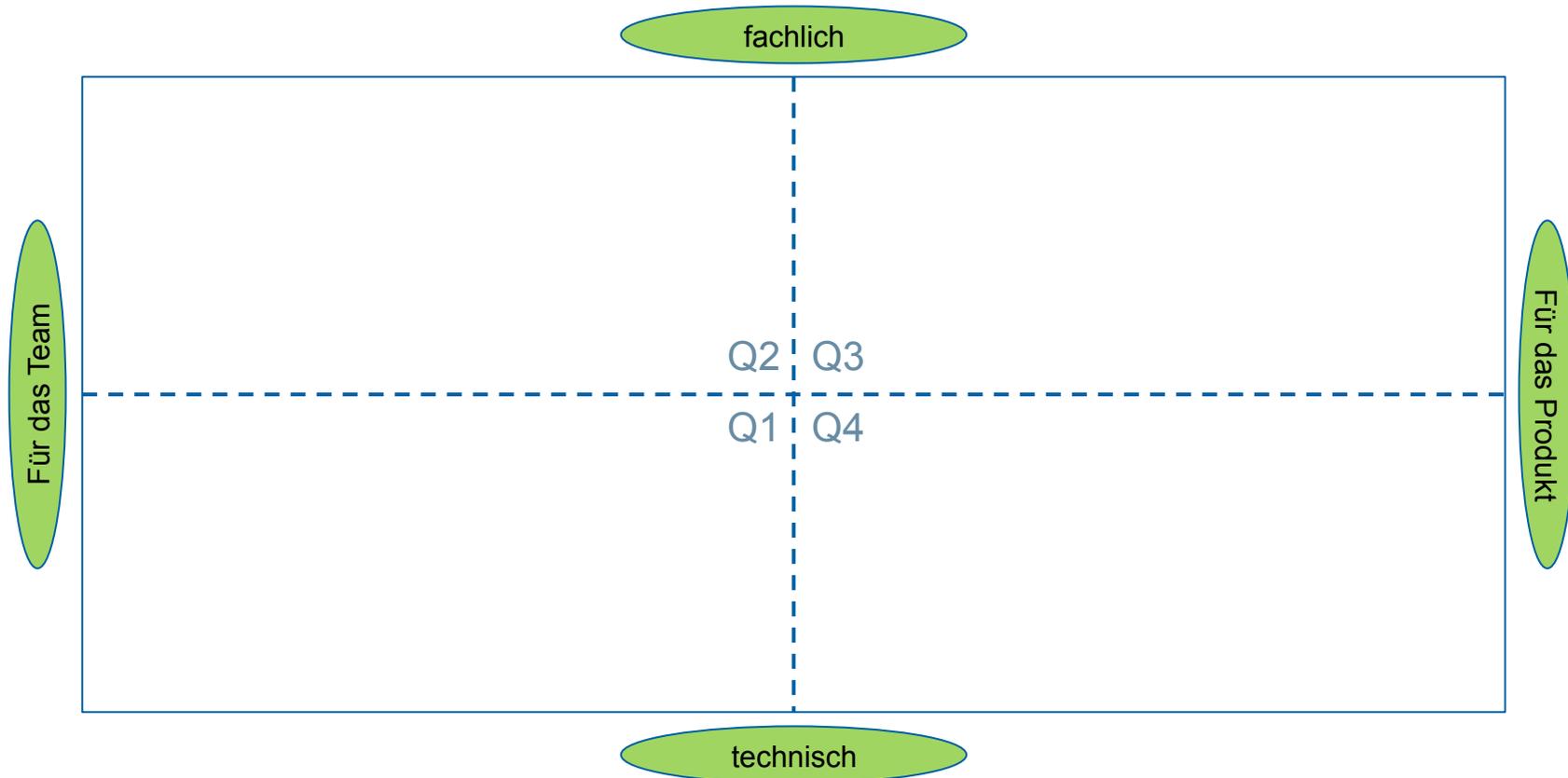
- **Testabdeckungsgrad** einer Software sollte ermittelt werden können, um
 - Testlücken gezielt schließen zu können
 - Refactorings gefahrlos durchführen zu können
- **Ergebnis** der automatisierten Tests muss für Entwickler sichtbar sein
 - Stichwort: „Continuous Integration“
 - Stichwort: „Early Feedback“
 - Rasches Gegensteuern im Fehlerfall



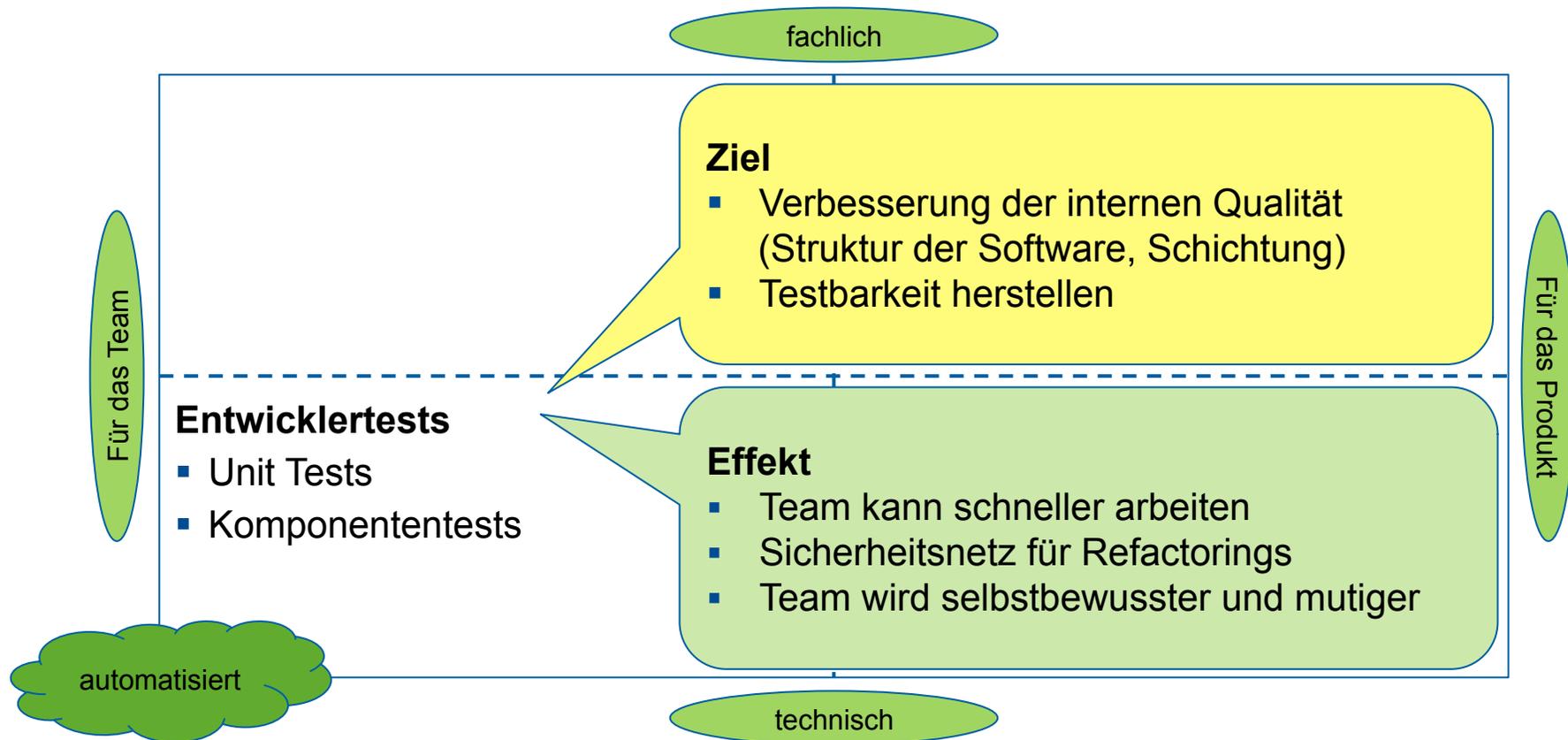
Agenda

- Ziel dieses Vortrags
- Grundzüge des agilen Testens
- Voraussetzungen für agiles Testen
- **Vorstellung der agilen Testquadranten**
- Die Testpyramide
- Anwendungsbeispiele für die Umsetzung agiler Tests
- Fazit, Fragen, Diskussion, Kontakt

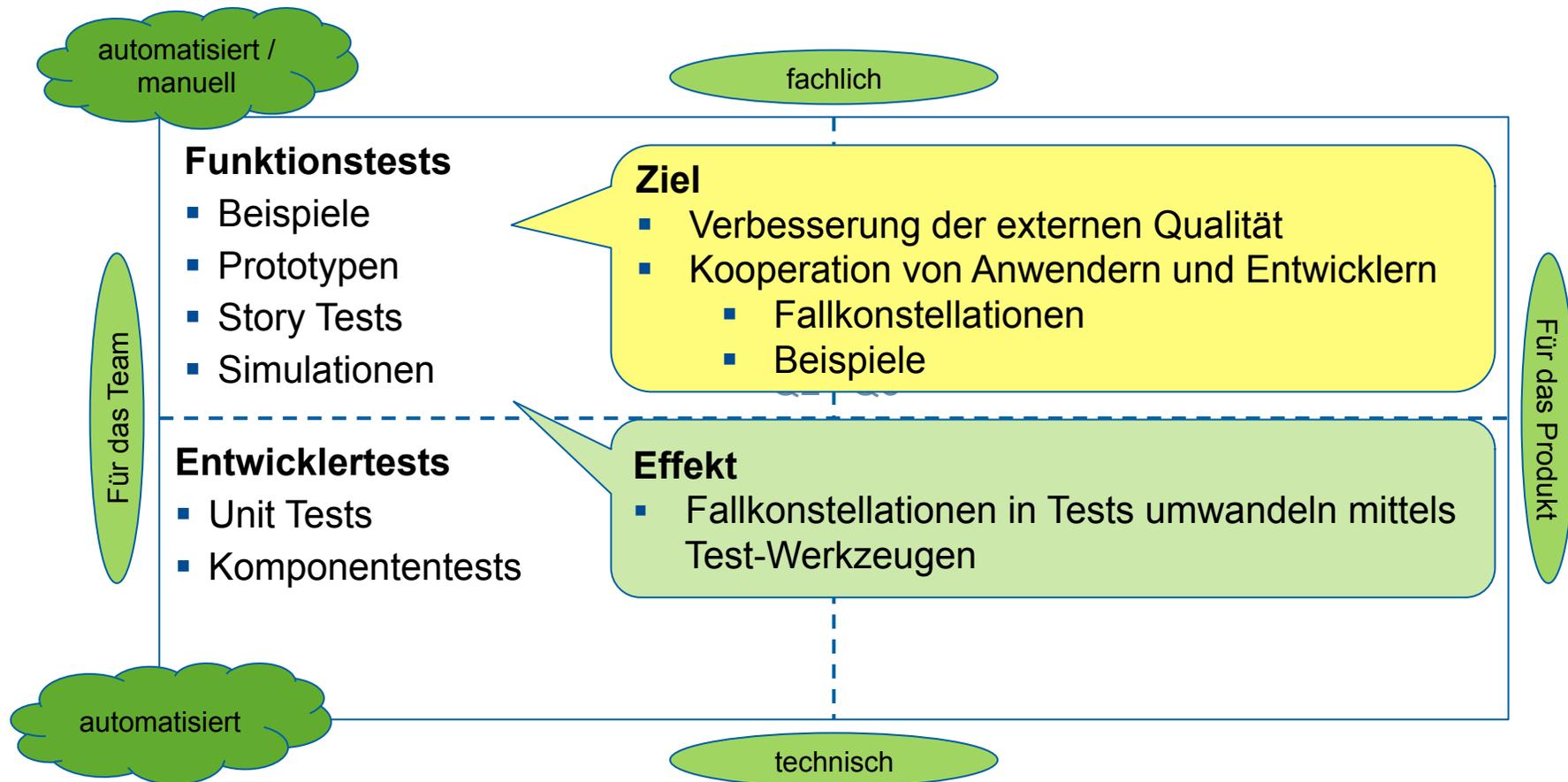
Agile Testquadranten



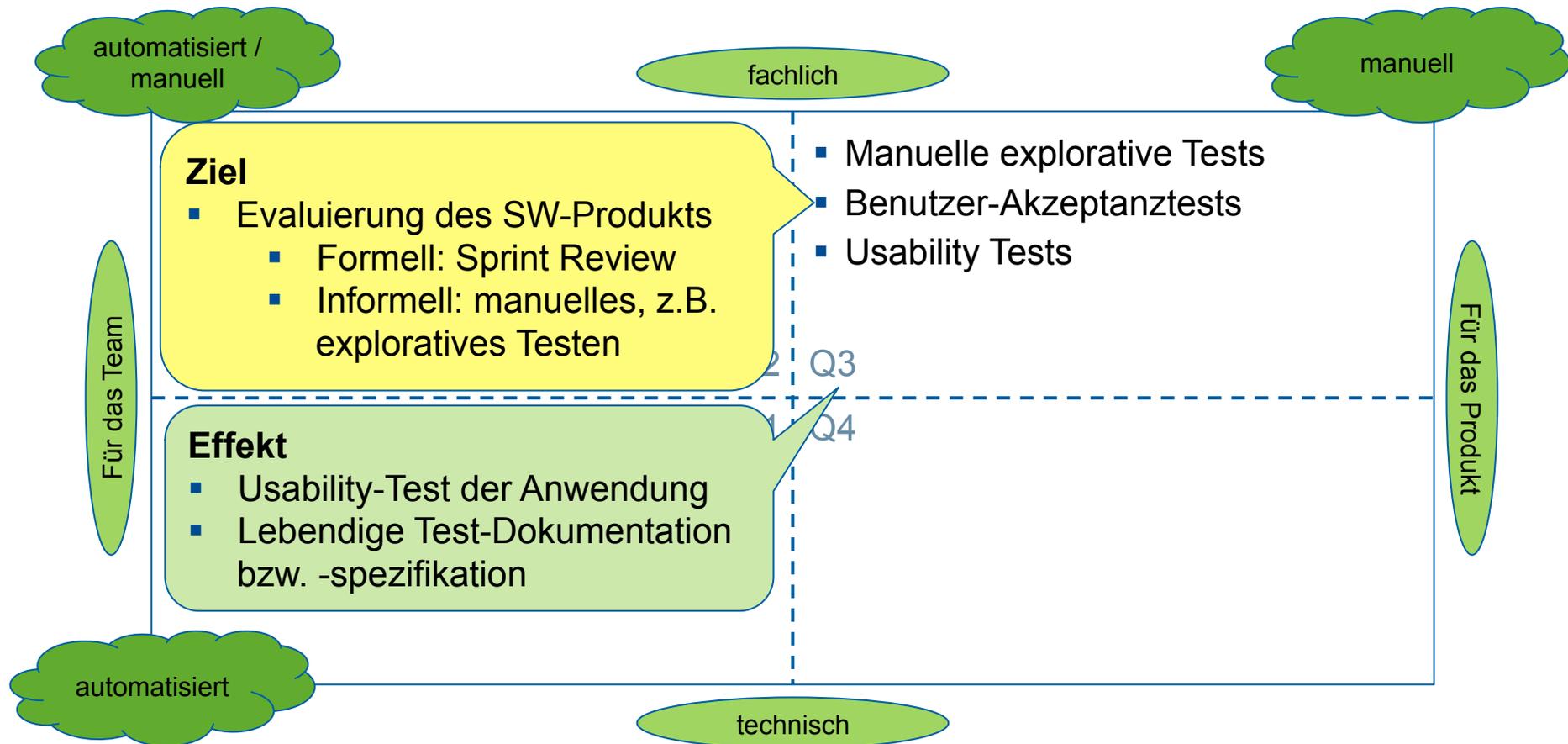
Agile Testquadranten



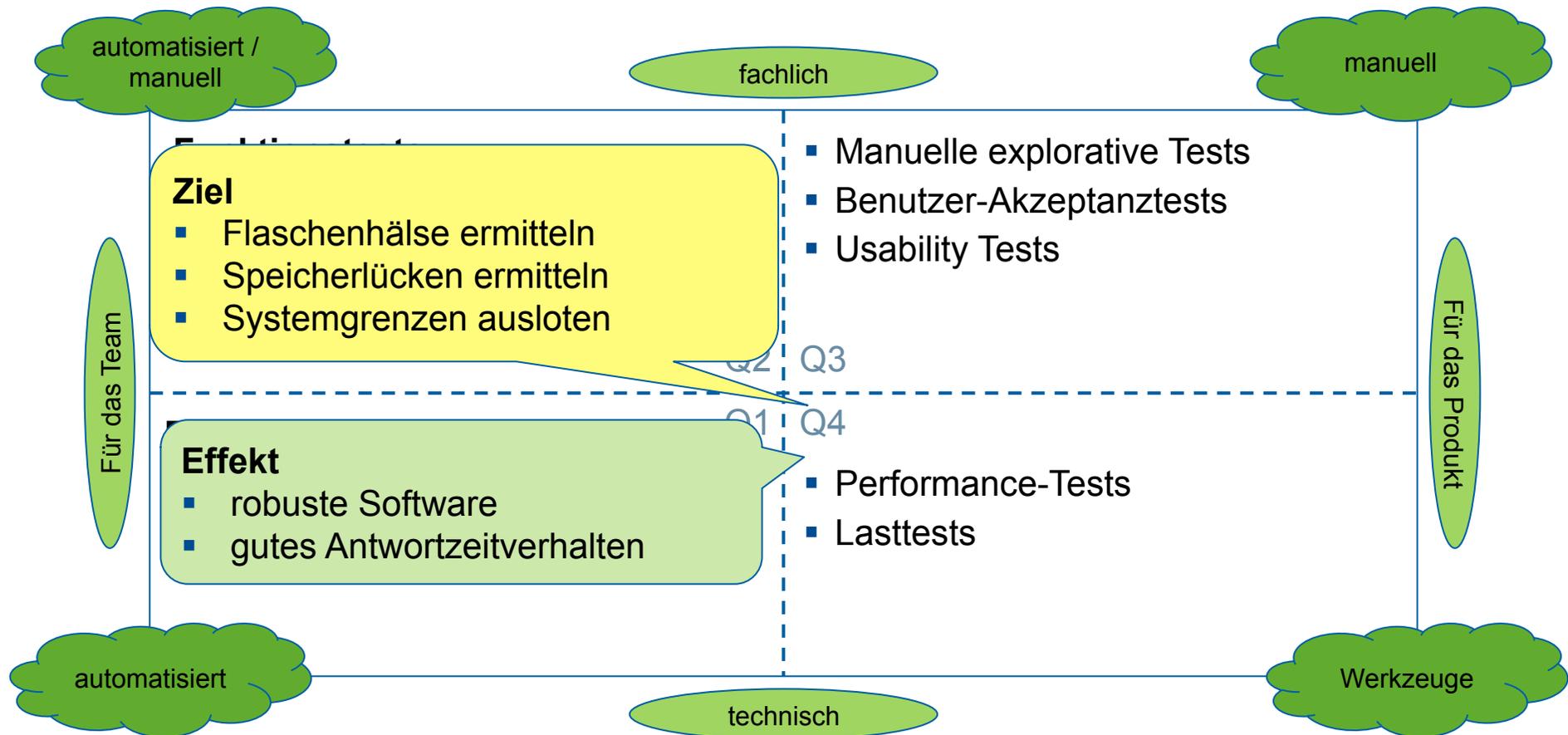
Agile Testquadranten



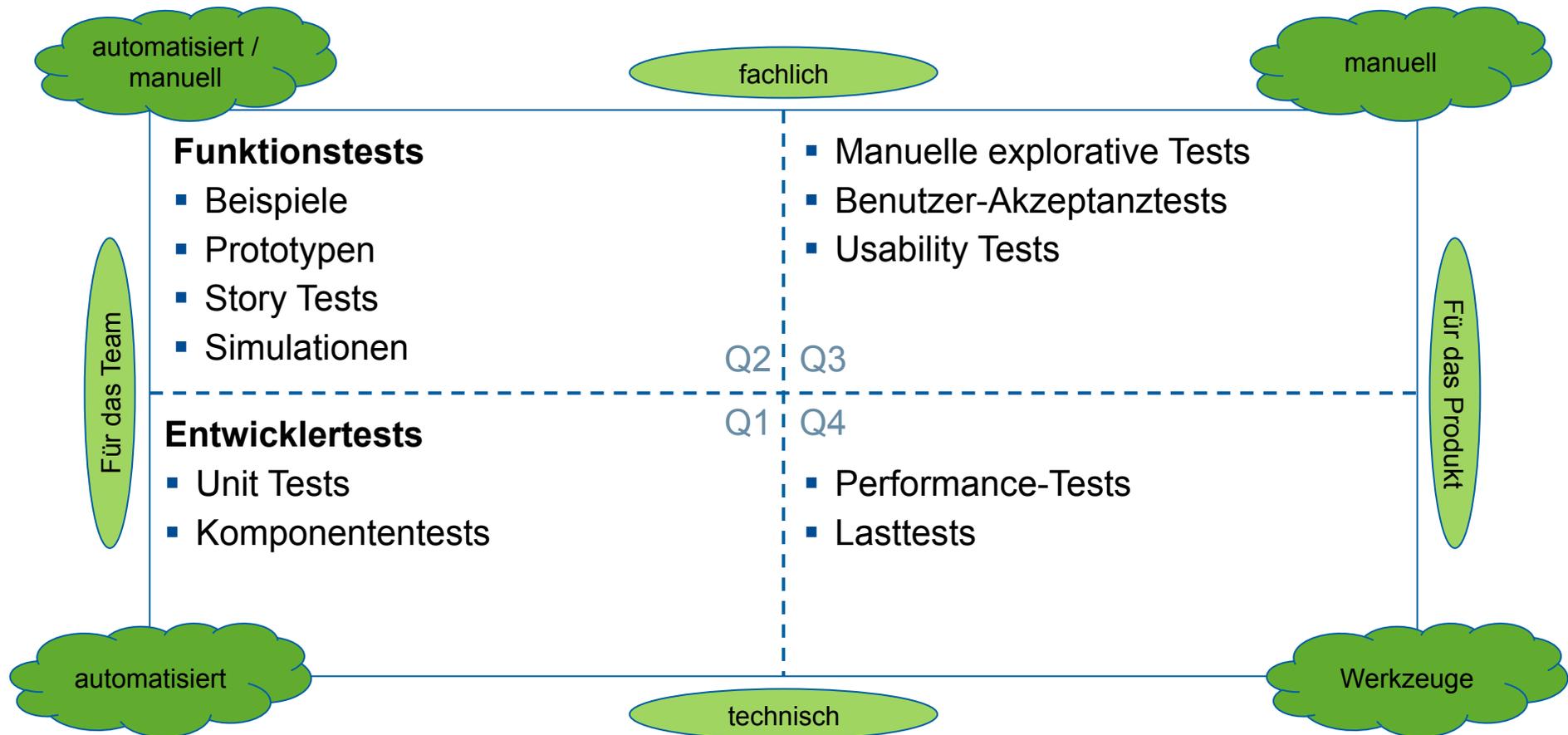
Agile Testquadranten



Agile Testquadranten



Agile Testquadranten



Definition of done

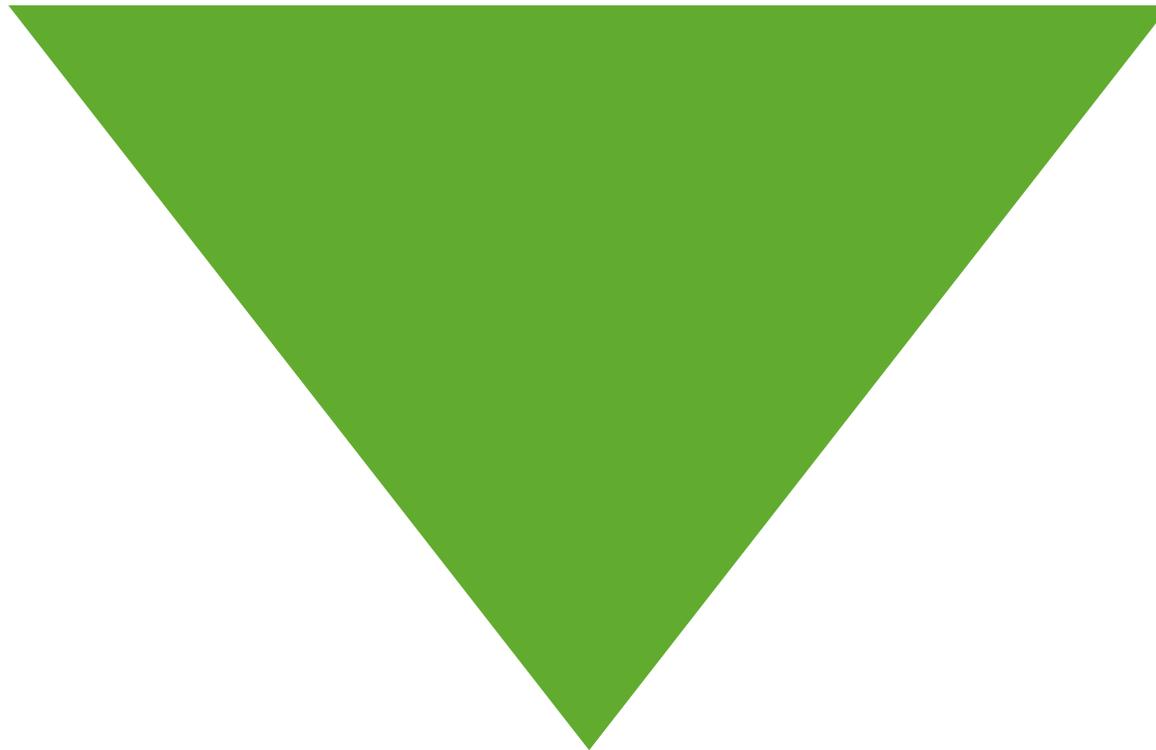
Tests als integraler Bestandteil des Softwareentwicklungsprozesses

- Keine Anforderung ist „fertig“, bis das Testen abgeschlossen ist
- dies betrifft im Zweifelsfalle alle Quadranten
 - d.h. automatisierte und manuelle Regressionstests
- Akzeptanzkriterien der Endanwender werden in lauffähige Tests umformuliert

Agenda

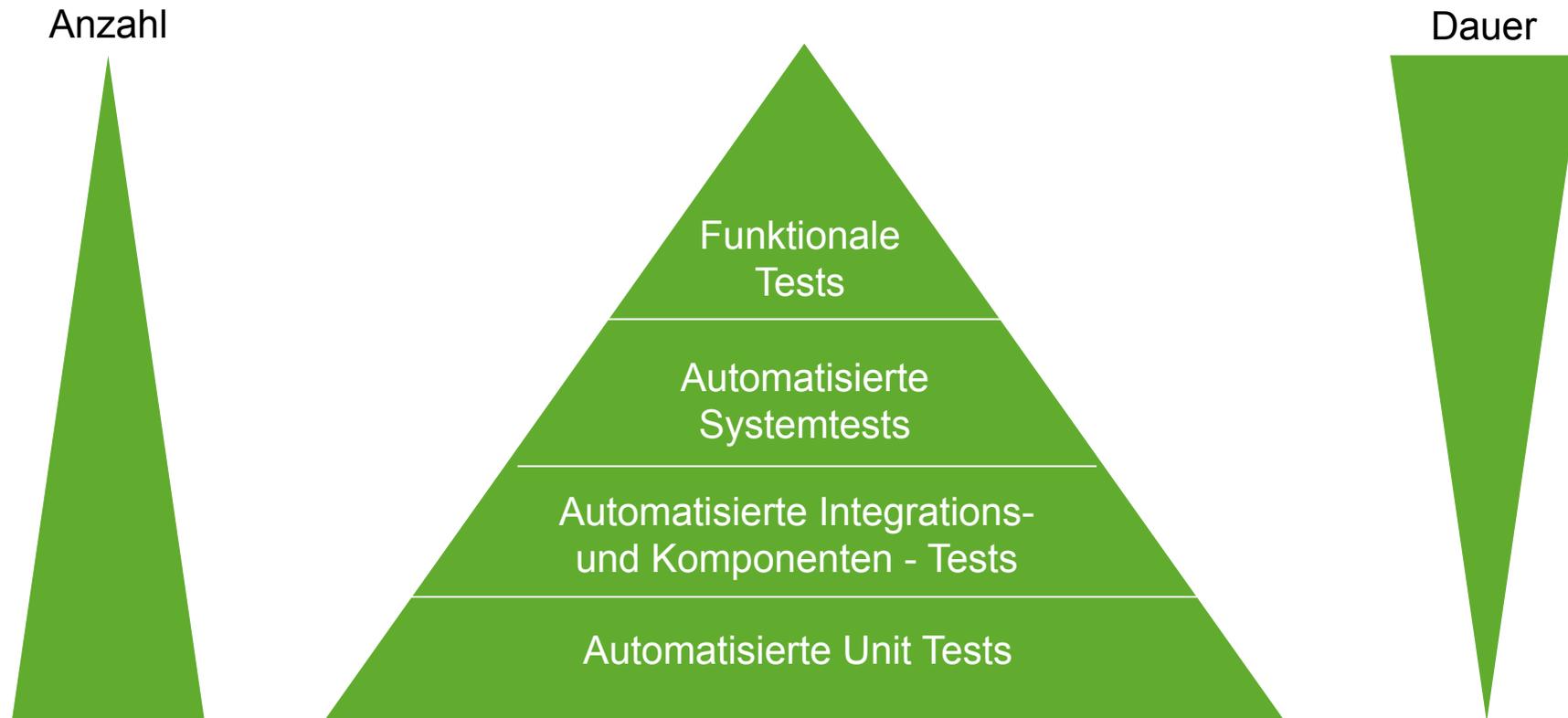
- Ziel dieses Vortrags
- Grundzüge des agilen Testens
- Voraussetzungen für agiles Testen
- Vorstellung der agilen Testquadranten
- **Die Testpyramide**
- Anwendungsbeispiele für die Umsetzung agiler Tests
- Fazit, Fragen, Diskussion, Kontakt

Die Testpyramide ?



Nein, eher ein Abbild der Realität in vielen Projekten...

Die Testpyramide



Die Testpyramide - Umsetzungshinweise

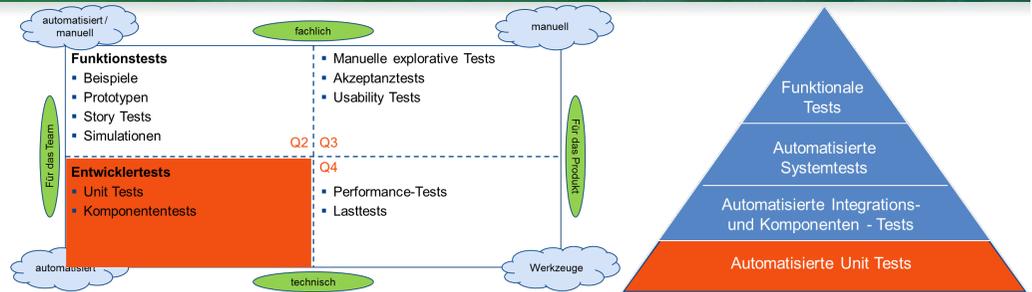
Hinweise

- Zielorientierter Einsatz von Werkzeugen
- Anwender und Entwickler arbeiten gemeinsam
- Lernorientierung hinsichtlich manueller Tests: Gelerntes zur Verbesserung heranziehen

Agenda

- Ziel dieses Vortrags
- Grundzüge des agilen Testens
- Voraussetzungen für agiles Testen
- Vorstellung der agilen Testquadranten
- Die Testpyramide
- **Anwendungsbeispiele für die Umsetzung agiler Tests**
- Fazit, Fragen, Diskussion, Kontakt

Quadrant 1 / Ebene 1: JUnit



```

ProduktRepositoryTest.java
public class ProduktRepositoryTest {

    private static final String EIN_PRODUKT = "einProdukt";
    private ProduktRepository produktRepository;
    private Produkt produkt;

    @Before
    public void setUp() {
        produkt = new Produkt(EIN_PRODUKT);
        produktRepository = new ProduktRepository();
    }

    @Test
    public void testGetAnzahl() {
        assertEquals(new Integer(0), produktRepository.getAnzahl());
    }

    @Test
    public void testAddEinProduktAndGetAnzahl() {
        produktRepository.add(produkt);

        assertEquals(new Integer(1), produktRepository.getAnzahl());
    }

    @Test
    public void testAddZweiProdukteAndGetAnzahl() {
        produktRepository.add(produkt);
        produktRepository.add(new Produkt("einanderesProdukt"));

        assertEquals(new Integer(2), produktRepository.getAnzahl());
    }

    @Test
    public void testAddZweiGleicheProdukteAndGetAnzahl() {
        produktRepository.add(produkt);
        try {
            produktRepository.add(produkt);
        }
    }
        
```

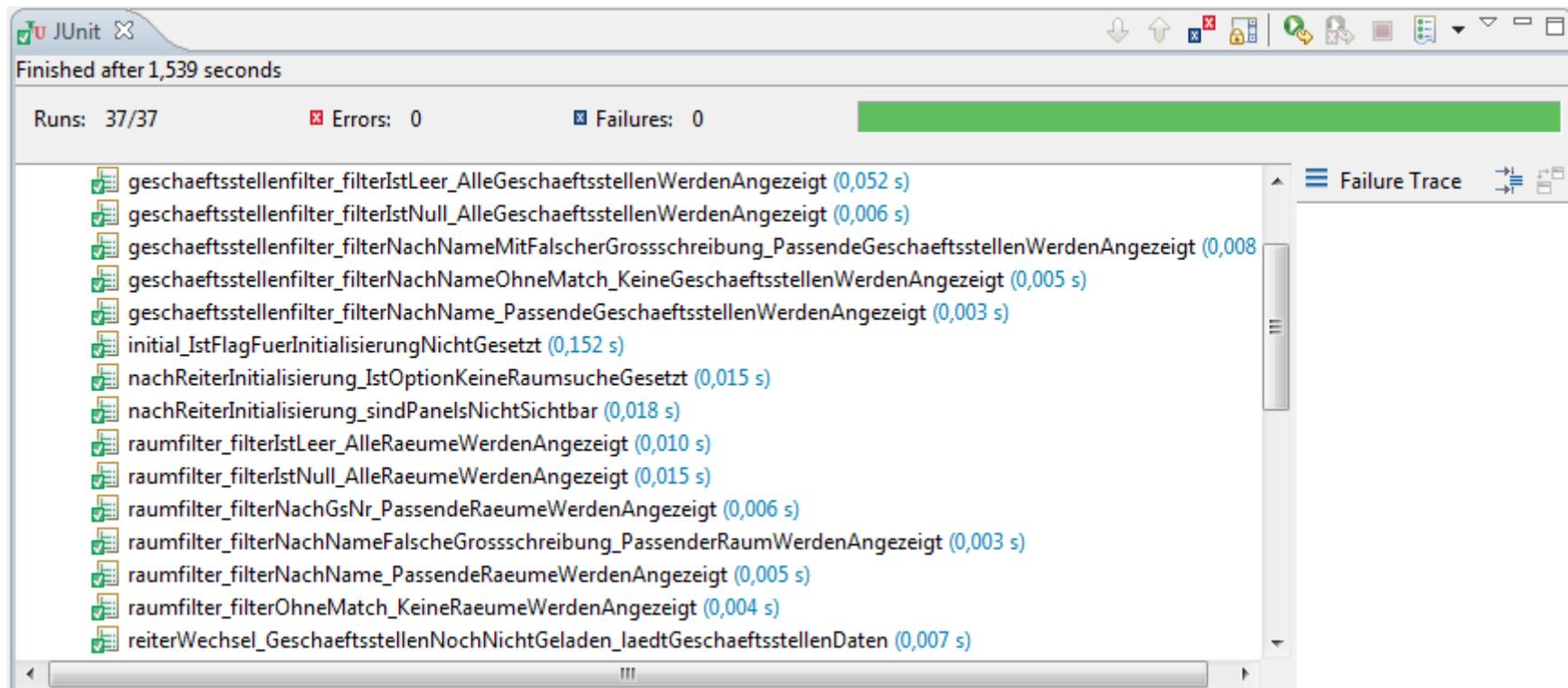
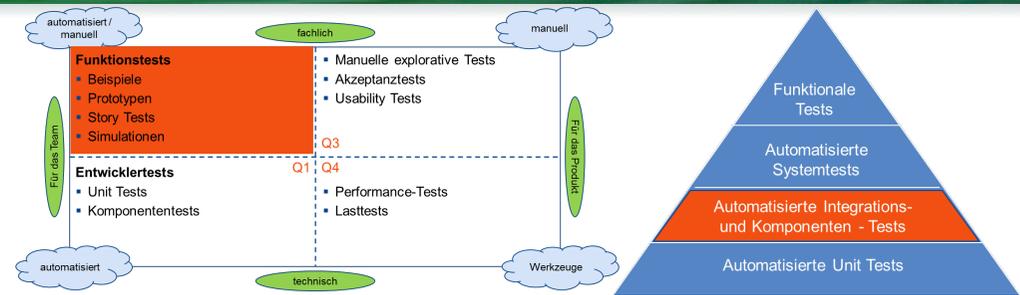
```

JUnit
Finished after 0,016 seconds

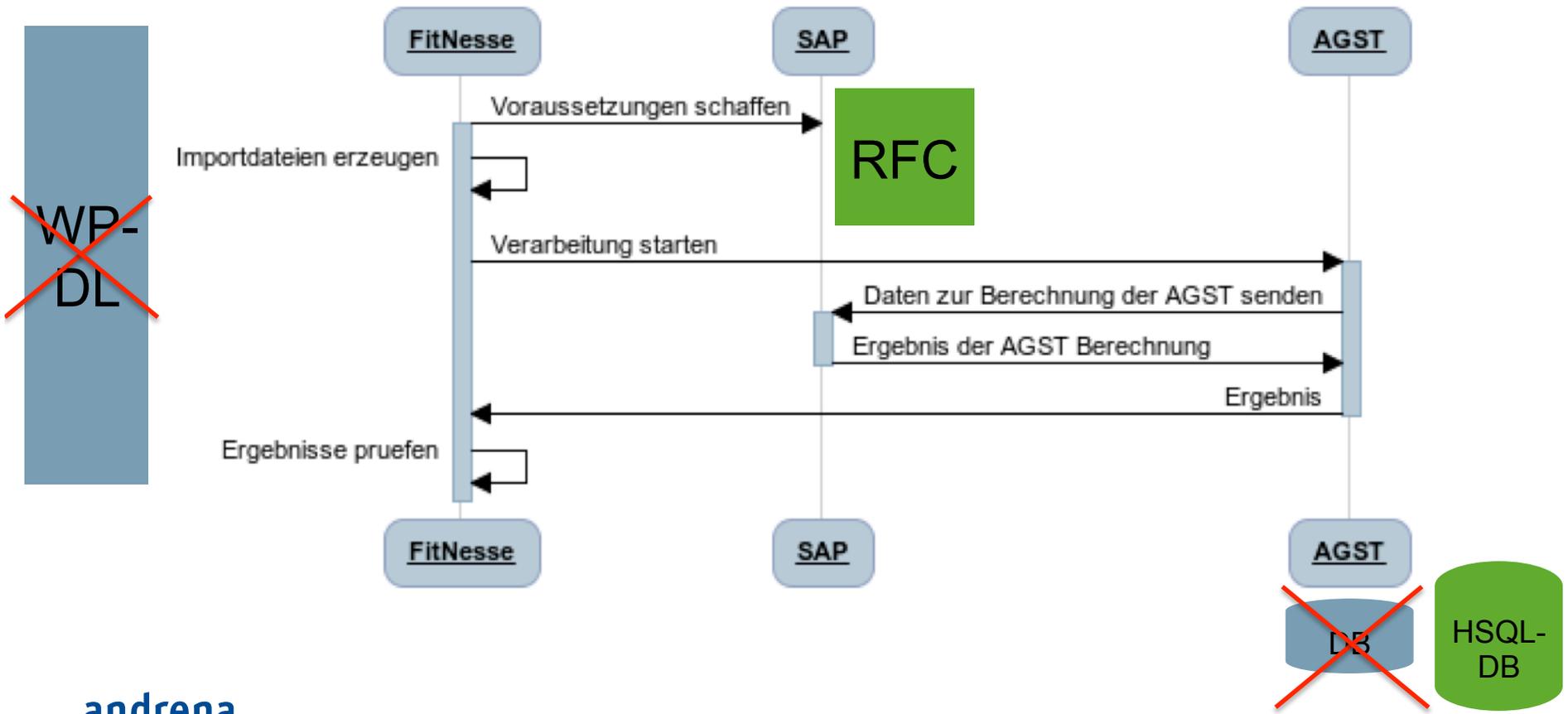
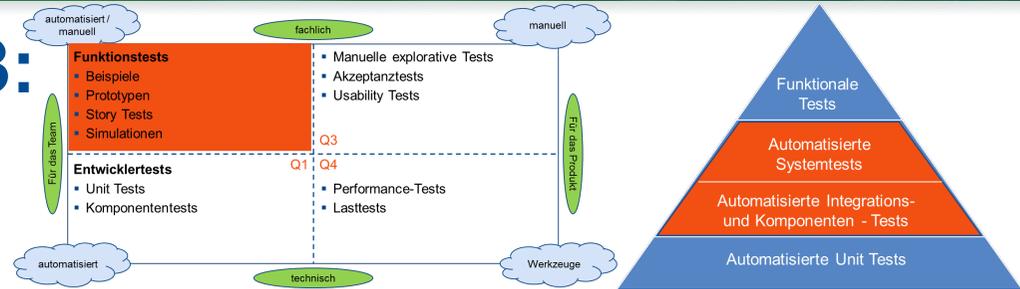
Runs: 8/8  Errors: 0  Failures: 0

tddkurs.uebung5.persistence.ProduktRepositoryTest [
  testGetAnzahl (0,007 s)
  testAddEinProduktAndGetAnzahl (0,000 s)
  testAddZweiProdukteAndGetAnzahl (0,000 s)
  testAddZweiGleicheProdukteAndGetAnzahl (0,000 s)
  testAddZweiGleicheProduktIdsAndGetAnzahl (0,000 s)
  testFindeProduktBekannteld (0,000 s)
  testFindeProduktKeineProduktId (0,001 s)
  testFindeProduktUnbekannteProduktId (0,000 s)
        
```

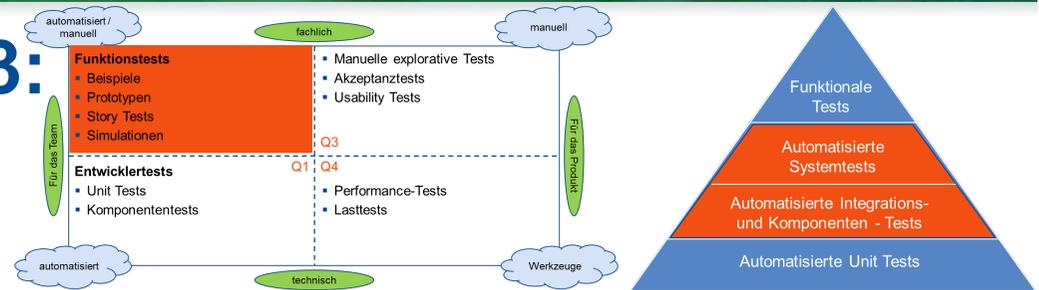
Quadrant 2 / Ebene 2: JUnit



Quadrant 2 / Ebene 2/3: FitNesse



Quadrant 2 / Ebene 2/3: FitNesse





VERSION 2, SERIE 1, FALL 3

Betriebsvermögen, keine Ausbuchung Topf, keine Besteuerung, neue Steuertatbestände

BEREITSTELLEN DER EINGABEDATEN

start	Praepariere Importsatz	
Generiere Geschäftsvorfallnummer mit Testfallnummer 213		Scenario
Abrechnungsdatum	07.02.2017	
Depotnummer	1209100	
Betriebsvermögen Kapitalgesellschaft		Scenario
WKN	970360	
Betrag 1: 1500,00 Euro mit Ertragsart 482		Scenario
Betrag 2: 200,00 Euro mit Ertragsart 483		Scenario
Importsatz hinzufuegen		

DURCHFÜHRUNG DER VERARBEITUNG

Führe Verarbeitung durch mit Buchungsdatum 07.02.2017		Scenario
Lade Datensatz zu Testfallnummer 213		Scenario

PRÜFUNG DER ERGEBNISSE

prüfe Verteilung der Einkünfte 100:0		Scenario
prüfe Standard-Steuersätze		Scenario
check	Prozentsatz KapSt Kunde 1	25,00
check	Prozentsatz KiSt Kunde 1	0,00
check	Ergebnis der steuerlichen Einordnung	K

check	Pauschbetrag urspruenglich	0,00
check	Saldo Pauschbetrag nach Verarbeitung	0,00
check	Saldo VT Aktien nach Verarbeitung	0,00
check	Saldo VT Sonstige nach Verarbeitung	0,00
check	Saldo VT QueSt nach Verarbeitung	0,00
check	Jahresberechnungsgrundlage KapSt nach Verarbeitung	2140,00

PRÜFUNG DER KOMPONENTEN

check	Anzahl Komponenten	2
-------	--------------------	---

start	Ergebniswerte fuer Komponente	1
check	Ertragsart	482
check	Angefragter Betrag	1500,00
check	Aenderung VT Aktien	0,00
check	Aenderung VT Sonstige	0,00

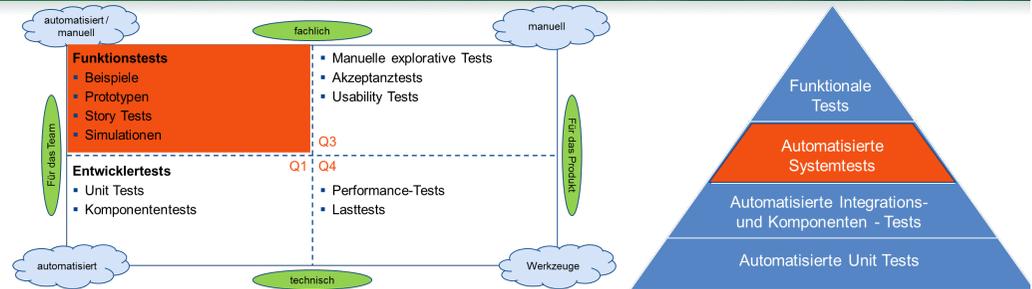
check	Aenderung VT Pauschbetrag	0,00
check	Bemessungsgrundlage vor Abzug der anrechenbaren QueSt	1500,00
check	Aenderung VT QueSt	0,00
check	Berechnungsgrundlage KapSt	1500,00
check	Anteilige KapSt fuer Kunde 1	375,00
check	Anteiliger SolZ fuer Kunde 1	20,62
check	Anteilige KiSt fuer Kunde 1	0,00

start	Ergebniswerte fuer Komponente	2
check	Ertragsart	483
check	Angefragter Betrag	200,00
check	Aenderung VT Aktien	0,00
check	Aenderung VT Sonstige	0,00
check	Aenderung VT Pauschbetrag	0,00
check	Bemessungsgrundlage vor Abzug der anrechenbaren QueSt	200,00
check	Aenderung VT QueSt	0,00
check	Berechnungsgrundlage KapSt	200,00
check	Anteilige KapSt fuer Kunde 1	50,00
check	Anteiliger SolZ fuer Kunde 1	2,75
check	Anteilige KiSt fuer Kunde 1	0,00

STEUERAUSGLEICH

Kein Steuerausgleich	Scenario
----------------------	----------

Quadrant 2 / Ebene 3: Selenium



The screenshot shows Selenium IDE test code on the left and a browser window on the right. The code is for a test named 'GoogleToSeleniumNavigationTest'.

```

import static org.hamcrest.CoreMatchers.is;

public class GoogleToSeleniumNavigationTest {

    private WebDriver driver;
    private GoogleSearchPage googleSearchPage;

    @Before
    public void setUp() throws Exception {
        initDriver();
        initGoogleSearchPage();
    }

    private void initGoogleSearchPage() {
        googleSearchPage = new GoogleSearchPageFactory.initElements(driver, googleSearchPage);
    }

    private void initDriver() {
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("about:home");
    }

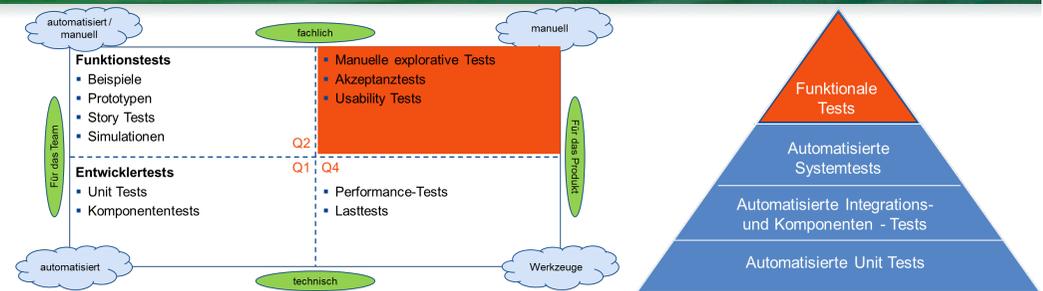
    @After
    public void tearDown() throws Exception {
        driver.quit();
    }

    @Test
    public void testSeleniumExample() throws Exception {
        GoogleResultsPage resultsPage = seleniumHomePage.seleniumHomePage.verifySeleniumHomePage(seleniumHomePage);
    }

    private GoogleResultsPage specifyAndExecute() {
    }
}
    
```

The browser window shows the Google.de homepage with the search bar and 'Google-Suche' button. The Selenium IDE interface shows the test is running successfully (1/1 runs, 0 errors, 0 failures).

Quadrant 3 – manuelles Testen I



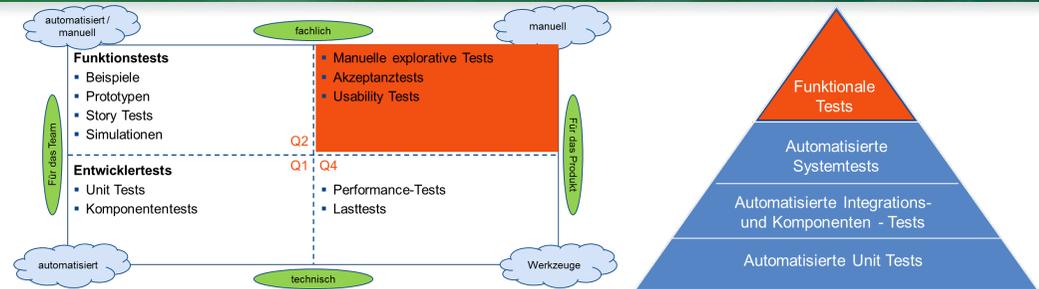
Hauptziel agile Softwareentwicklung:

Das Ausliefern und Bereitstellen neuer Funktionalitäten nach jedem Entwicklungssprint

Anwenden des manuellen Testens um

- nicht-triviale Fehler aufzufinden
 - Definierter Soll-Zustand vs. „Was wäre wenn?“
- zu erforschen, wie eine Anforderung getestet werden kann
- Tests auszuführen, deren Automatisierung zu aufwändig bzw. „teuer“ ist
- „Sonderkonstellationen“ abzudecken

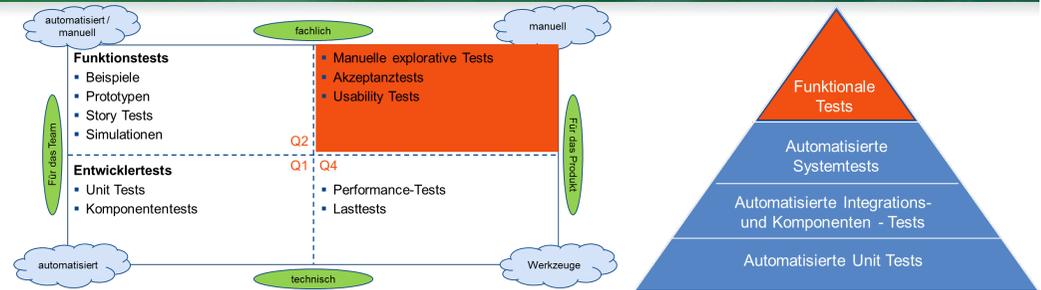
Quadrant 3 – manuelles Testen II



Exploratives Testen als leichtgewichtiger Ansatz für manuelle Tests

- Paralleles Testdesign und -ausführung
- Im Gegensatz zu „fixen“ Testvorgaben nach Plänen (die meist allerdings gewisse Interpretationsspielräume offen lassen)
- Vorgehen nach einem strukturiertem Ansatz
- Beinhaltet Protokolle, auch als Inspiration für künftige Testsitzungen
- Letzter Test beeinflusst den nächsten Test
- Kein Widerspruch zu klassischen Testplänen, sondern ergänzend

Quadrant 3 – manuelles Testen III



Analogie:

Software-Tester vergleichbar einem Tourist

- Man sieht nicht alles, bzw. nur bestimmte Teile der Software
- Es gibt **viele Wege**, um das Ziel zu erkunden
- Man möchte sicherstellen, dass man die **wichtigsten** Punkte gesehen hat

Zutaten:

- Feste Zeitvorgabe (**Timebox**)
- Dokumentation (**Testprotokoll**)
- Umfang und Art der Test-Tour (**Scope**)



Vorgehen:

- Spezifische Touren (FedEx, Supermodel...)

Agenda

- Ziel dieses Vortrags
- Grundzüge des agilen Testens
- Voraussetzungen für agiles Testen
- Vorstellung der agilen Testquadranten
- Die Testpyramide
- Anwendungsbeispiele für die Umsetzung agiler Tests
- **Fazit, Fragen, Diskussion, Kontakt**

Fazit

- Testautomatisierung und agiles Testen erhöhen die **Testabdeckung** und damit mittelbar die **Code-Qualität**.
- Eine hohe Testabdeckung ist elementare **Voraussetzung für Refactorings**
- **Kosten** zur späten **Fehlerbehebung** sinken deutlich
- Entwicklungsteams können eine **hohe Entwicklungsgeschwindigkeit** beibehalten
- Agiles Testen unterstützt insofern Entwickler dabei, **guten Code** zu entwickeln.
- Guter Code ist das Ziel eines jeden Entwicklers – und wirtschaftlich sinnvoll !

Lars Alvincz

lars.alvincz@andrena.de

07 21 - 61 05 - 122

Daniel Knapp

daniel.knapp@andrena.de

07 21 - 61 05 - 13 66

Weitere Fragen?

[meet the SPEAKER@speakerlounge](mailto:meet.the.SPEAKER@speakerlounge)

1. OG DIREKT ÜBER DEM EMPFANG