

Programmierschnittstellen und Softwarequalität

Sommersemester 2022

Hochschule Bochum

JBehave

Akzeptanztests

Philipp Wegner

Khaled Youssef

Agenda

Einleitung

- › Begriffserklärung: BDD, TDD, ATDD
 - › BDD vs. ATDD
 - › ATDD vs. TDD

Hauptteil

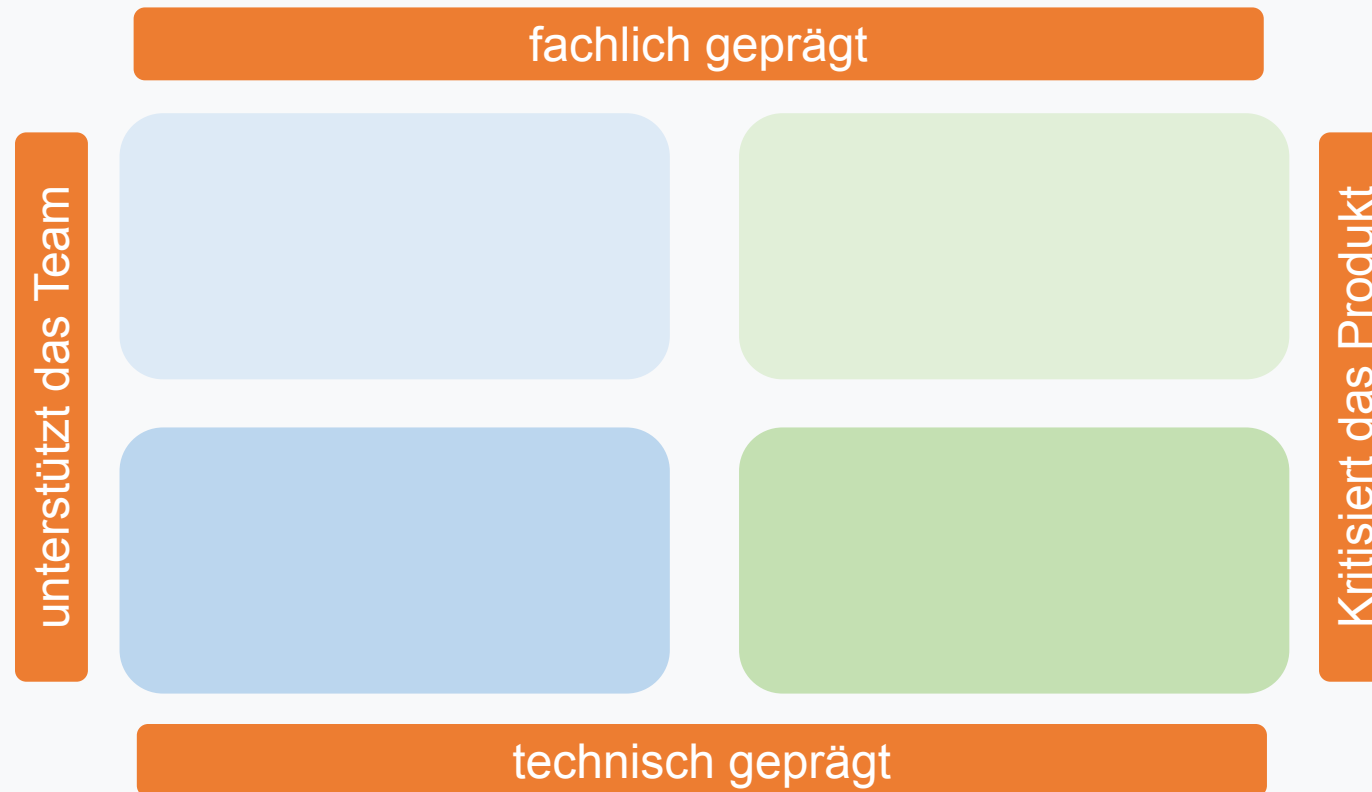
- › Akzeptanztests
 - › Was sind Akzeptanztests?
 - › Warum Akzeptanztests?
 - › Wie führt man Akzeptanztests durch?
- › Vorstellung des Frameworks JBehave

Abschluss

- › Do's and Don'ts
- › Conclusion

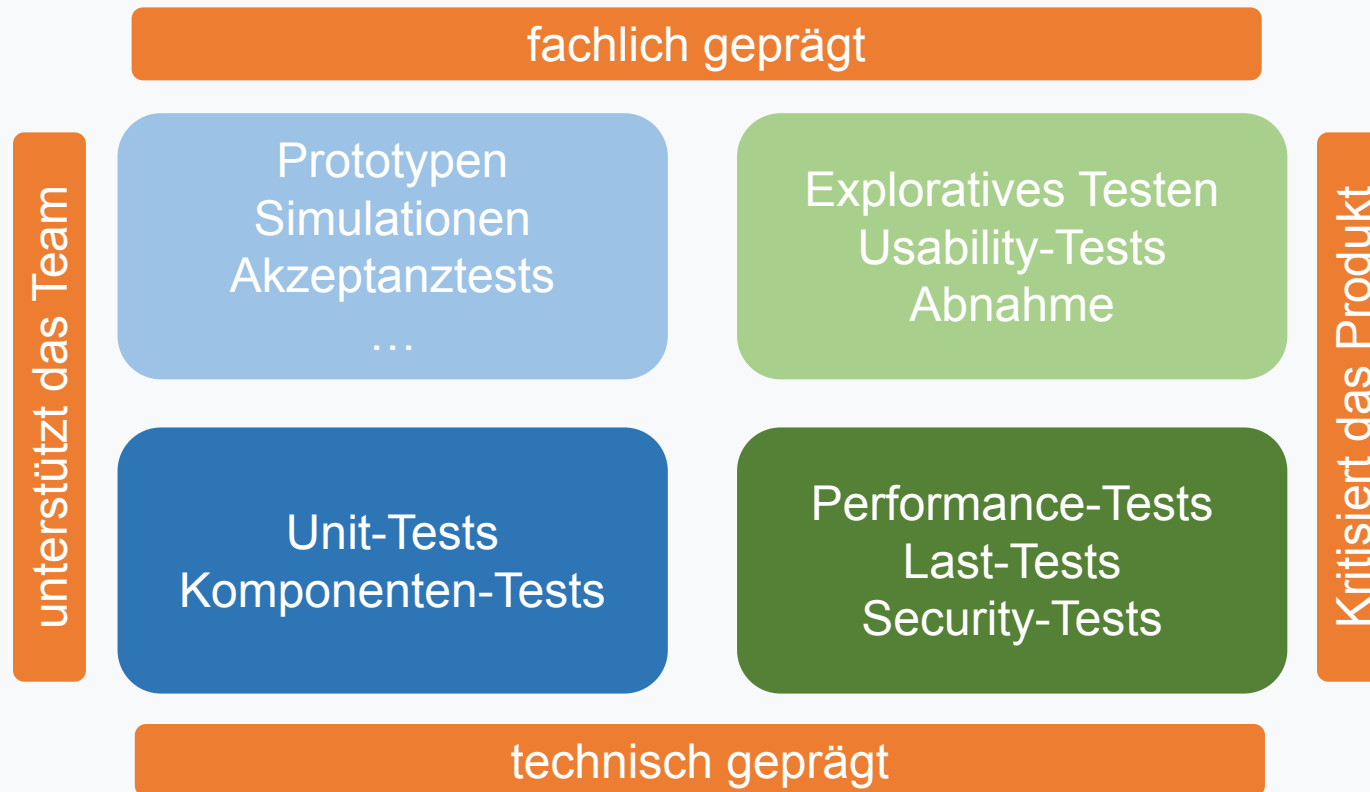
Einleitung

› Test-Quadrant:



Einleitung

› Test-Quadrant:



Einleitung

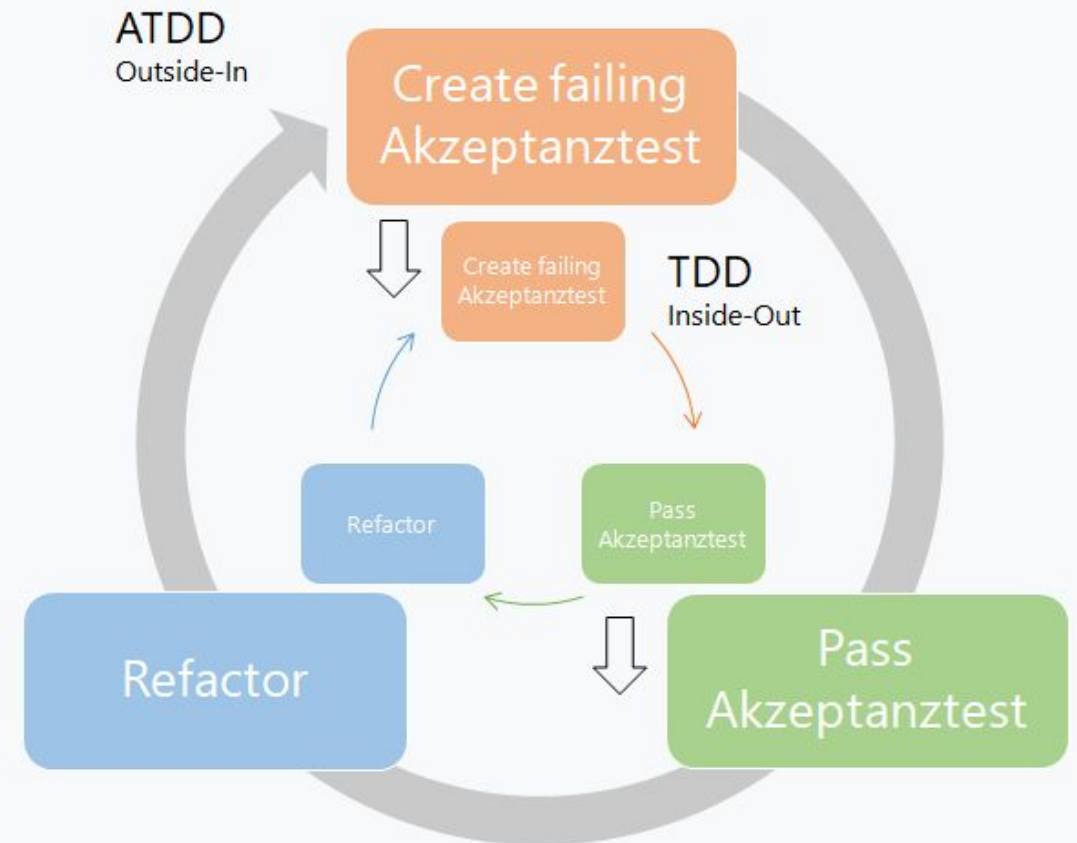
› ATDD vs. TDD:

ATDD:

- › Höheres Abstraktionslevel
- › Test für gesamtes Feature, dann Klassen und abschließend einzelne Methoden / Funktionen

TDD:

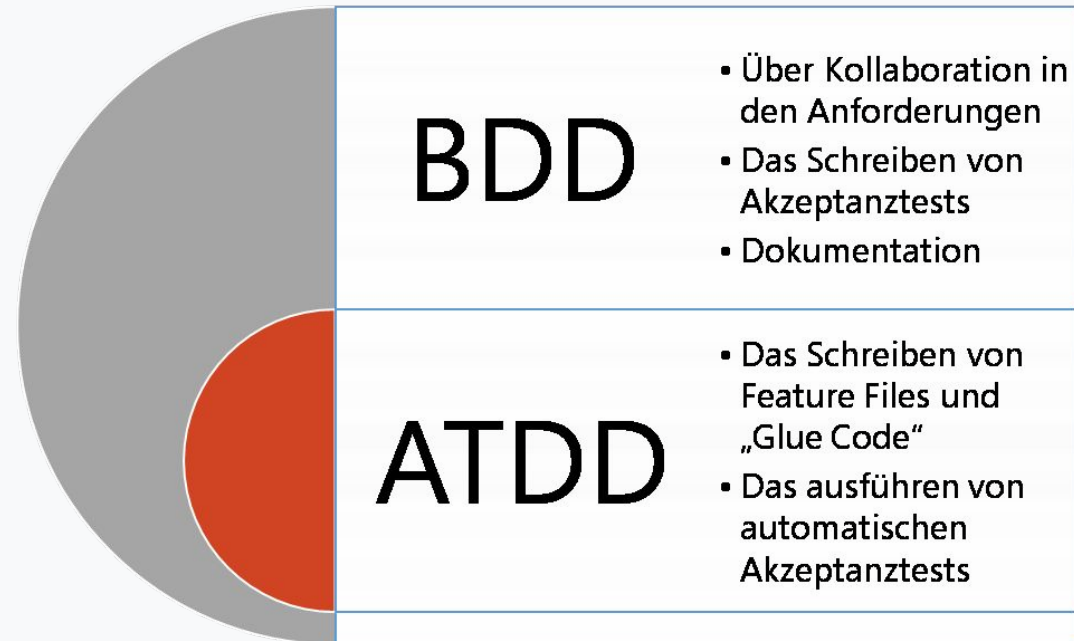
- › Auf dem Level der Klasse
- › Testbeschreibung für Methoden / Funktion, dann wird man größer bis zum Gesamttest



Einleitung

› ATDD vs. BDD:

- › Werden oft als Synonyme füreinander verwendet
- › BDD der methodische Teil
- › ATDD der technische Teil
- › BDD ↔ ATDD
 - › Je nach Ressourcen



Akzeptanztests

Was sind Akzeptanztests?

- › Entscheidungsträger, Product Owner, Finanzmanager etc.
 - › nicht auf der Codeebene Unterwegs
 - › Anforderungen bis zum fertigen Produkt müssen definiert werden
 - › Hier kommen Akzeptanztests ins Spiel



Akzeptanztests

Was sind Akzeptanztests?

- › Entscheidungsträger, Product Owner, Finanzmanager etc.
 - › nicht auf der Codeebene Unterwegs
 - › Anforderungen bis zum fertigen Produkt müssen definiert werden
 - › Hier kommen Akzeptanztests ins Spiel
- › Kriterien an Akzeptanztests:
 - › Von allen Lesbar sein
 - › Ausführbar sein (Reproduzierbar)
 - › Report, sprich formale Dokumentation



Akzeptanztests

Was sind Akzeptanztests?

- › Entscheidungsträger, Product Owner, Finanzmanager etc.
 - › nicht auf der Codeebene Unterwegs
 - › Anforderungen bis zum fertigen Produkt müssen definiert werden
 - › Hier kommen Akzeptanztests ins Spiel
- › Kriterien an Akzeptanztests:
 - › Von allen Lesbar sein
 - › Ausführbar sein (Reproduzierbar)
 - › Report, sprich formale Dokumentation



Überzeugen das Liefern des Systems an den Kunden

Akzeptanztests

Wie sind Akzeptanztests aufgebaut?

› Stakeholder

- › Natürliche Sprache
- › Outside-In-Ansatz
- › Nutzen von BDD

› Nutzen von BDD

- › User-Stories
- › Reports
- ›



Akzeptanztests

User-Story

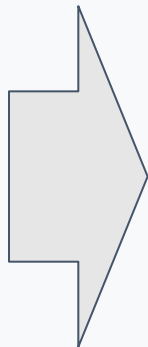
› User-Stories habe einen anderen Fokus als:

› Modultest

› Konzepte

› Protokolle

› Lasten- oder Pflichtenhefte



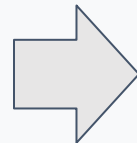
Tester

Entwickler

Management

Kunden

Ziel: eine Sprache zu sprechen, die **alle** verstehen!



User-Stories

Akzeptanztests

User-Story

- › User-Story besitzt Narrative aus 3 Fragen
 - › Wer ist der User des Systems?
 - › Was ist das Feature?
 - › Für welchen Zweck?

Zum Beispiel:

- › Feature: Kundenauthentifizierung
- › Als Kunde der Website
- › Ich möchte mich mit Username und Passwort authentifizieren
- › Sodass ich einkaufen kann.

Akzeptanztests

User-Story

- › User-Story besitzt Narrative aus 3 Fragen
 - › Wer ist der User des Systems?
 - › Was ist das Feature?
 - › Für welchen Zweck?
- › Szenarien
 - › User-Stories bestehen aus Szenarien
 - › Szenario entspricht Akzeptanzkriterium
 - › Szenario besteht aus Sequenzen von Schritten
 - › Jeder Schritt ist ein "Behavior" (Verhalten)

Zum Beispiel:

- › Feature: Kundenauthentifizierung
- › Als Kunde der Website
- › Ich möchte mich mit Username und Passwort authentifizieren
- › Sodass ich einkaufen kann.

Zum Beispiel:

- › Szenario: Kunde mit ungültigem Passwort erhält Fehlermeldung
- › Given: Kunde geht zu local:8080/gui/login
- › When: username ist „myUser“ und Passwort „inncorrectPwd“
- › Then: Fehlermeldung „Username oder Passwort nicht korrekt“

Teaser: JBehave Github Repository

- › Link zum Github-Repository:
https://github.com/PhilippWegner/pssq_jbehave



master	1 branch	0 tags	Go to file	Add file	Code
pwegner final commit			b5b9d78 13 minutes ago 26 commits		
pssq_akzeptanztest_01	final commit	14 minutes ago			
pssq_akzeptanztest_02	final commit	14 minutes ago			
pssq_akzeptanztest_03	final commit	13 minutes ago			
pssq_akzeptanztest_04	final commit	14 minutes ago			
pssq_akzeptanztest_05	final commit	14 minutes ago			
.gitignore	gitignore	5 days ago			
README.md	Initial commit	5 days ago			

Languages



JBehave

Ein Framework für BDD:

I. Write story

Plain
text

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0

When stock is traded at 5.0

Then the alert status should be OFF

When stock is traded at 16.0

Then the alert status should be ON

JBehave

Ein Framework für BDD:

1. Write

Scenario: A trader

Given a stock and

When stock is traded

Then the alert status

When stock is traded

Then the alert status

2. Map steps to Java

POJO

```
public class TraderSteps {  
    private TradingService service; // Injected  
    private Stock stock; // Created  
  
    @Given("a stock and a threshold of $threshold")  
    public void aStock(double threshold) {  
        stock = service.newStock("STK", threshold);  
    }  
    @When("the stock is traded at price $price")  
    public void theStockIsTraded(double price) {  
        stock.tradeAt(price);  
    }  
    @Then("the alert status is $status")  
    public void theAlertStatusIs(String status) {  
        assertThat(stock.getStatus().name(), equalTo(status));  
    }  
}
```


JBehave

Ein Framework für BDD:

1. Write

Scenario: A trader

Given a stock and a price
When stock is traded
Then the alert is sent
When stock is traded
Then the alert is sent

2. Map

```
public class TraderSteps {
    private TradingService tradingService;
    private Stock stock;

    @Given("a stock and a price")
    public void aStockAndPrice() {
        stock = service.getStockAndPrice();
    }

    @When("the stock is traded")
    public void theStockIsTraded() {
        stock.tradeAt(price);
    }

    @Then("the alert is sent")
    public void theAlertIsSent() {
        assertThat(stock.getAlert(), is(true));
    }
}
```

3. Configure Stories

```
public class TraderStories extends JUnitStories {

    public Configuration configuration() {
        return new MostUsefulConfiguration()
            .useStoryLoader(new LoadFromClasspath(this.getClass()))
            .useStoryReporterBuilder(new StoryReporterBuilder()
                .withCodeLocation(codeLocationFromClass(this.getClass()))
                .withFormats(CONSOLE, TXT, HTML, XML));
    }

    public List<CandidateSteps> candidateSteps() {
        return new InstanceStepsFactory(configuration(),
            new TraderSteps(new TradingService())).createCandidateSteps();
    }

    protected List<String> storyPaths() {
        return new StoryFinder().findPaths(codeLocationFromClass(this.getClass()),
            "**/*.story");
    }
}
```

Only
once

JBehave

Ein Framework für BDD:

1. Write

Scenario: A trader

Given a stock and a price

When stock is traded

Then the alert is sent

When stock is not traded

Then the alert is not sent

2. Map

```
public class TraderStory {
    private TradingService tradingService;
    private Stock stock;

    @Given("a stock and a price")
    public void aStockAndPrice() {
        stock = service.getStock();
    }

    @When("the stock is traded")
    public void theStockIsTraded() {
        stock.tradeAt(price);
    }

    @Then("the alert is sent")
    public void theAlertIsSent() {
        assertThat(stock.isTraded(), is(true));
    }
}
```

3. Configure


```
public class TraderStory {
    public Configuration configuration() {
        return new MostFavorableConfiguration()
            .useStoryLoader(new StoryLoader())
            .useStoryReporter(new StoryReporter())
            .withContext(new Context())
            .withFramework(new Framework());
    }

    public List<Candidate> candidates() {
        return new InstanceOfCandidateProvider().candidates();
    }

    protected List<String> stories() {
        return new StoryFinder().findStories("**/*.story");
    }
}
```

4. Run Stories

With any of



JBehave

Ein Framework für BDD:



Der Kunde

Abstimmung mit dem Kunden

- › Einbindung des Kunden im Entwicklungsprozess
- › User-Stories und Akzeptanzkriterien gemeinsam mit dem Kunden entwickeln
- › Vorteil, fachliche Fehler früh erkennen
- › User-Stories dienen als Werkzeug



Die Qualitätsabnahme (QA)

Abstimmung mit der QA

- › QA wird früh eingebunden
- › Dadurch bessere Kommunikation
- › Akzeptanzkriterien erweitern den Testkatalog
- › Testfälle müssen nicht nachträglich definiert werden



Umsetzung der Tests



Integration

› Automatisierte Builds (Jenkins)

› Build → Test → Deployment → Akzeptanztests → Dokumentation

› Laufzeit

- › bei mehreren Akzeptanztests, Jenkins kann Jobs parallelisieren
- › Ergebnisse werden vereint

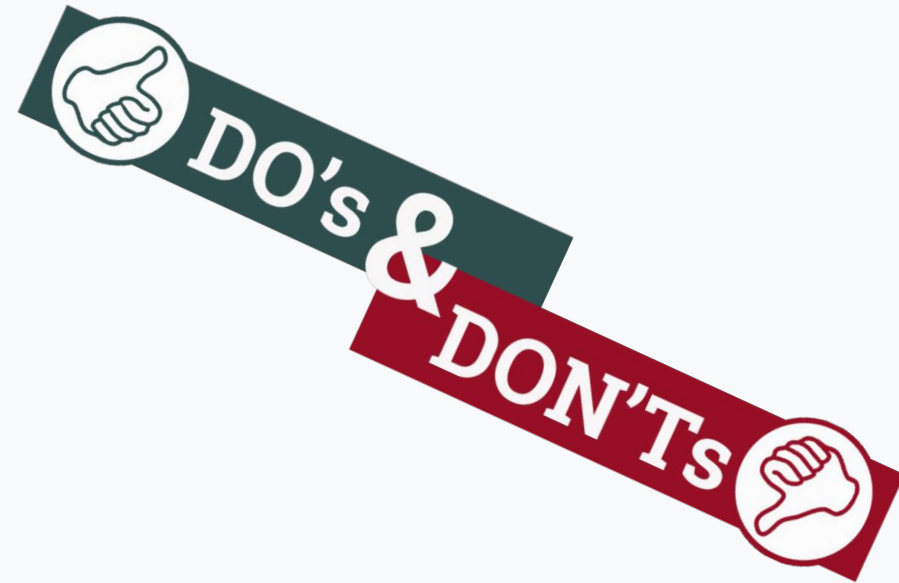
› Bei Fehlern

- › Product-Owner und der zuständige technische Architekt werden automatisch benachrichtigt
- › Laufzeitprobleme, vordefinierte Annotationen (z.B. @VIP) führen nur priorisierte Szenarien aus



Do's and Don'ts

- › **Nicht alles Automatisieren**
- › **Szenarien müssen lesbar sein**
- › **So einfach wie möglich formulieren**
- › **Wartbarkeit im Hinterkopf behalten**
- › **Robustheit, Robustheit, Robustheit**
- › **BDD ist mehr als nur automatisieren und Testen**



Conclusion

› **Gleiche Sicht:**

- › Mit BDD, alle Beteiligten haben die gleiche Sicht auf ein Produkt

› **Dokumentationsaufwand reduzieren**

- › Redundante Dokumente werden reduziert

› **Hoher Aufwand**

- › Zu Beginn BDD recht aufwendig
- › rentabel nur bei großen Projekten

JBehave

Akzeptanztests

Vielen Dank für die Aufmerksamkeit!

Philipp Wegner

Khaled Youssef