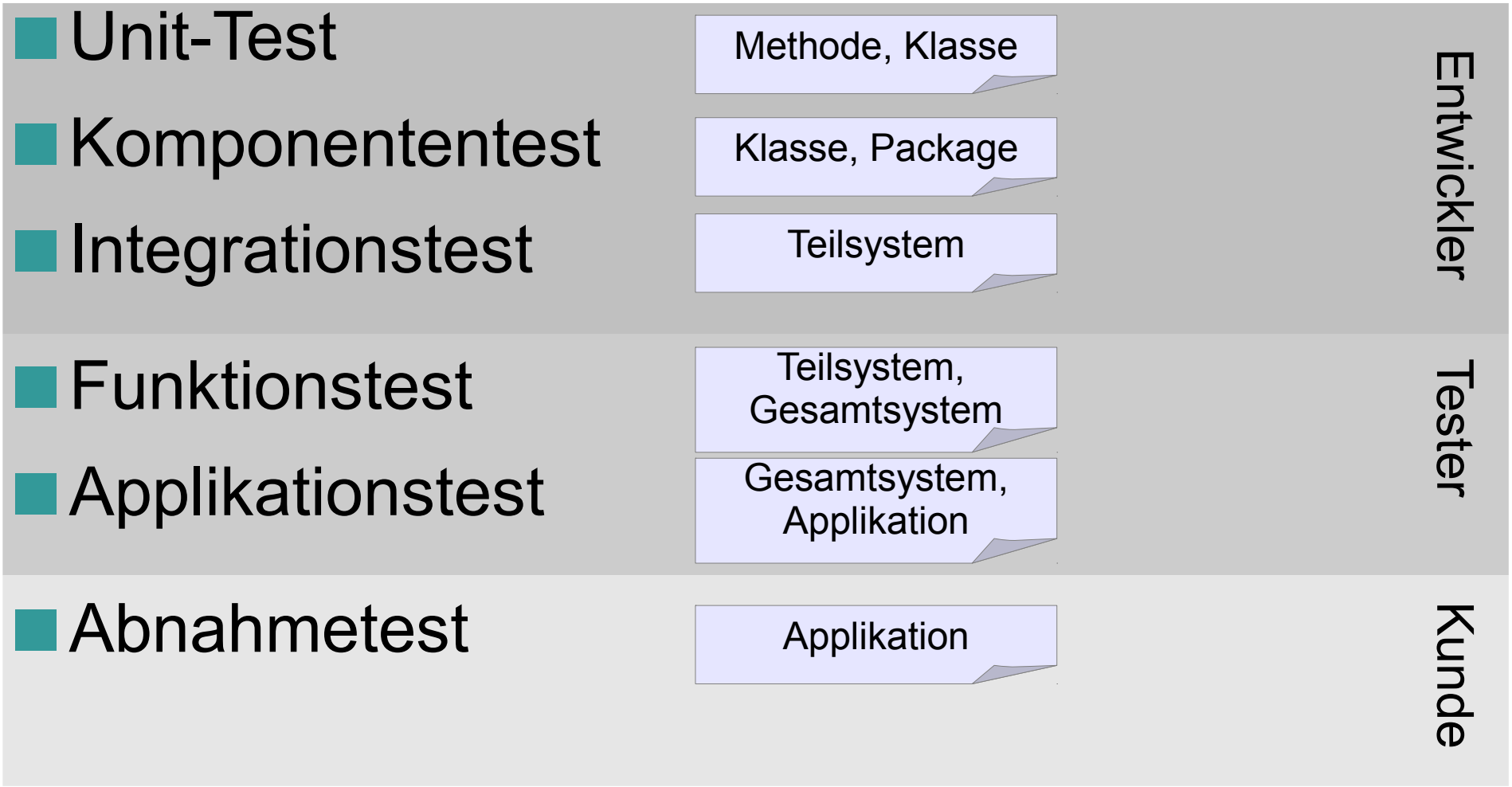
Fortgeschrittene Programmiertechniken

Unit-Test mit JavaScript
Ein kurzer Überblick

Unit-Test
Einordnung im Software-Test



Unit-Test
Begriff

- Test der kleinsten Einheiten
  - Funktionen
  - Methoden
- Isolierte Betrachtung
- Erstellung einzelner Testfälle
- Testfälle als eigenständiges Programm
- Siehe Programmier-Paradigmen wie z.B.:
  - Extreme Programming
  - Test-/Feature-/Behaviour-Driven Development (TDD/FDD/BDD)

Unit-Test
Test-/Feature-/Behaviour-getrieben?

- Test-Driven Development (TDD)
  - Ausrichtung eher technisch
  - Gesamtmenge der Testfälle
    deckt Gesamtmenge der Funktionalität ab
- Features-Driven Development (FDD)
  - Ausrichtung fachlich oder technisch
  - Gesamtmenge der Testfälle
    beschreibt die Gesamtmenge der Features

Achtung, keine erschöpfende Darstellung ;-)

Unit-Test
Test-/Feature-/Behaviour-getrieben?

- Behaviour-Driven Development (BDD)
  - Ausrichtung eher fachlich
  - Gesamtmenge der Testfälle
    beschreibt das Verhalten der Anwendung,
    oft auch unter Einbeziehung der Anwender-Sicht

Achtung, keine erschöpfende Darstellung ;-)

Unit-Test – Beispiel
Test-Driven Development (TDD)

```
suite('Array', function(){
  setup(function(){
    // ...
  });

  suite('#indexOf()', function(){
    test('should return -1 when not present', function(){
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

Beispiel: mocha, http://visionmedia.github.io/mocha/#interfaces

Unit-Test – Beispiel Feature-Driven Dev. (FDD)

```
describe('Array', function(){
 before(function(){
   // ...
 });

 describe('#indexOf()', function(){
   it('should return -1 when not present', function(){
     [1,2,3].indexOf(4).should.equal(-1);
   });
 });
});
```

Beispiel: mocha, http://visionmedia.github.io/mocha/#interfaces

Unit-Test
Positive Effekte

- Unit-Test-Frameworks liefern klare Ergebnisse
- Qualität wird messbar(er)
- Test sind wiederholbar
- Bessere Fokussierung bei Fehlersuche möglich
- Nähe zur Implementierungsphase
- Know-How-Gewinn
- Änderungen werden sicher(er) durchführbar

Unit-Test Begriffe

Fixture	Ausführungs-Umgebung bzw. Randbedingungen (Daten, …)
Spy	Funktion, die alle Argumente, Return-Werte und Exceptions aufzeichnet („Wrapper“)
Stubs	(hier:) Vgl. Spy, jedoch mit vor-definiertem Verhalten
Mock	Stub (s.o.), zusätzlich mit mit definierten Soll-Ergebnis

Unit-Test Standard-Schema

- Vorbereitung der Ausführungsumgebung („Fixtures“)
- Erzeugung des Test-Kandidaten
- Definition des Inputs
- Definition des erwarteten Outputs
- Ausführung der zu testenden Methode
- Vergleich: tatsächlicher == erwarteter Output?
- Aufräumen der Ausführungsumgebung

Testfall

Test-Frameworks und Tools

NodeJS	als Plattform für automatisiertes Testen
PhantomJS	Headless Browser, auch mit NodeJS
Jasmine	TDD/FDD, v.a. Browser, auch PhantomJS, mit Spy-Support
QUnit	TDD, v.a. Browser, auch PhantomJS
Mocha	TDD/BDD, auf node.js (headless), auch mit PhantomJS
nodeunit	TDD, speziell für NodeJS
Chai	expext()-Bibliothek, z.B. mit Mocha
Sinon	Spy-/Stub-/Mock-Bibliothek, z.B. mit Mocha

Quellen & Links

- Jasmine JS test framework
  https://jasmine.github.io
- mocha JS test framework
  http://visionmedia.github.io/mocha/
- NodeJS JavaScript Runtime Environment
  http://nodejs.org/
- PhantomJS Headless Browser Environment
  http://phantomjs.org
- „Design for Testability“ auf MSDN
  http://msdn.microsoft.com/en-us/magazine/dd263069.aspx
-

Fortgeschrittene Programmiertechniken

Vielen Dank für Ihre Aufmerksamkeit!