

# Qualitätssicherung in Software Projekten

**Web Development**  
Vertiefungsmodul im Medieninformatik Bachelor

02.05.2019, Dirk Breuer

A dramatic, low-key lighting photograph of a man's face. He is wearing dark sunglasses and a camouflage military-style jacket. His expression is serious and focused, with his gaze directed towards the right side of the frame. The background is dark and out of focus.

**WHAT IF I TOLD YOU**

**THERE WILL ALWAYS BE BUGS**

# Aspekte von Qualitätssicherung

Dokumentation

Unit Tests

TDD

Acceptance Tests

Security

Prozesse

Manuell

BDD

Continuous Integration

Audits

Code Reviews

Funktional

Automatisch

Nicht-Funktional

User Tests

Performance

- Definierter Prozess
- Code Reviews
- Automatische Tests

# Definierter Prozess

# ERR TOO MANY BOOKS



CODE COMPLETE

Head First Servlets & JSP

The Art of UNIX Programming

Java Hacks

Using JRuby

ComptIA A+ Complete

Mac Kung Fu

Ajax Hacks

Expert One-on-one Design and Development

Java Concurrence in Practice

Essential Java idiom 3.0

sendmail

Don't Make Me Think!

Java Persistence API 2.0

Java Design Patterns

Java Concurrency in Practice

Java Performance Tuning

Java Persistence API 2.0

Java Design Patterns

Java Persistence API 2.0

Java Performance Tuning

Java Design Patterns

Java Persistence API 2.0

Java Design Patterns

# Definierter Prozess

- Strukturierung der Anforderungen
- Gemeinsames Verständnis der Anforderung
- Fortschritt messbar machen
- Transparenz und Nachvollziehbarkeit
- Kontinuierliche Verbesserung

Analyse (max. 5)



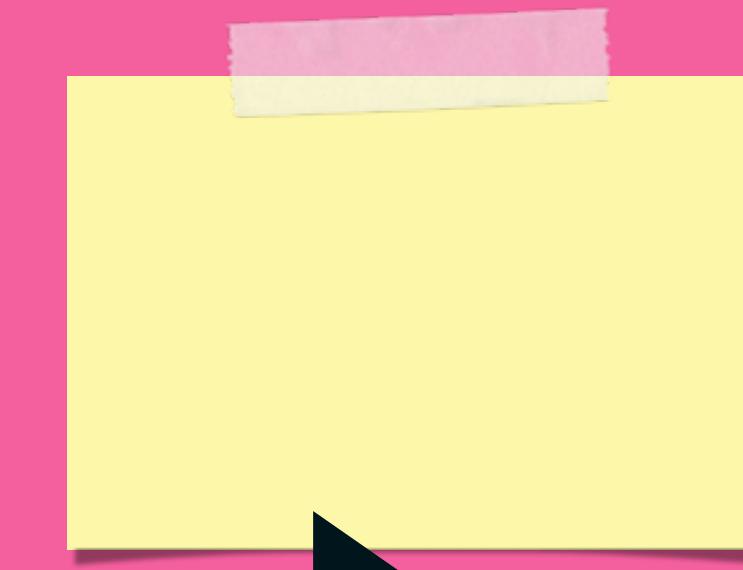
Entwicklung (max. 2)



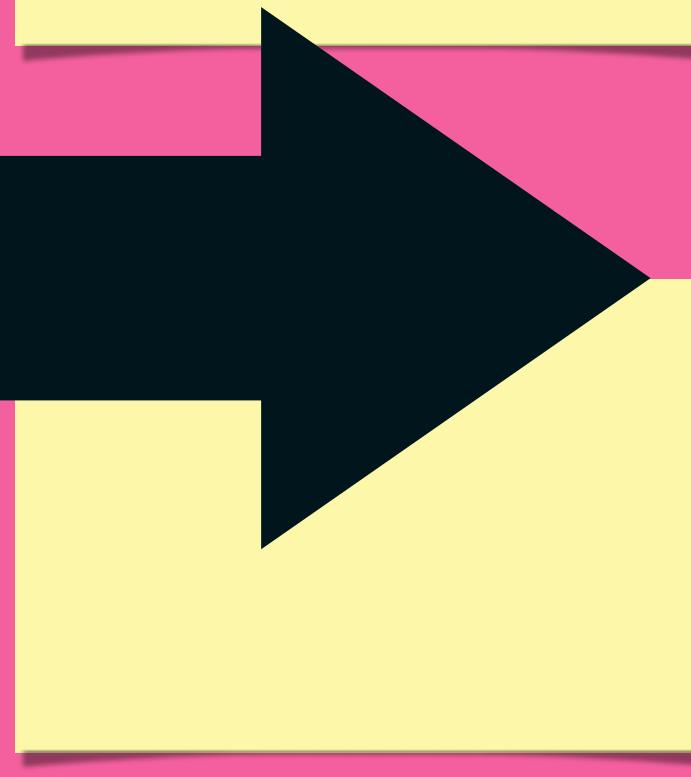
Testing (max. 3)



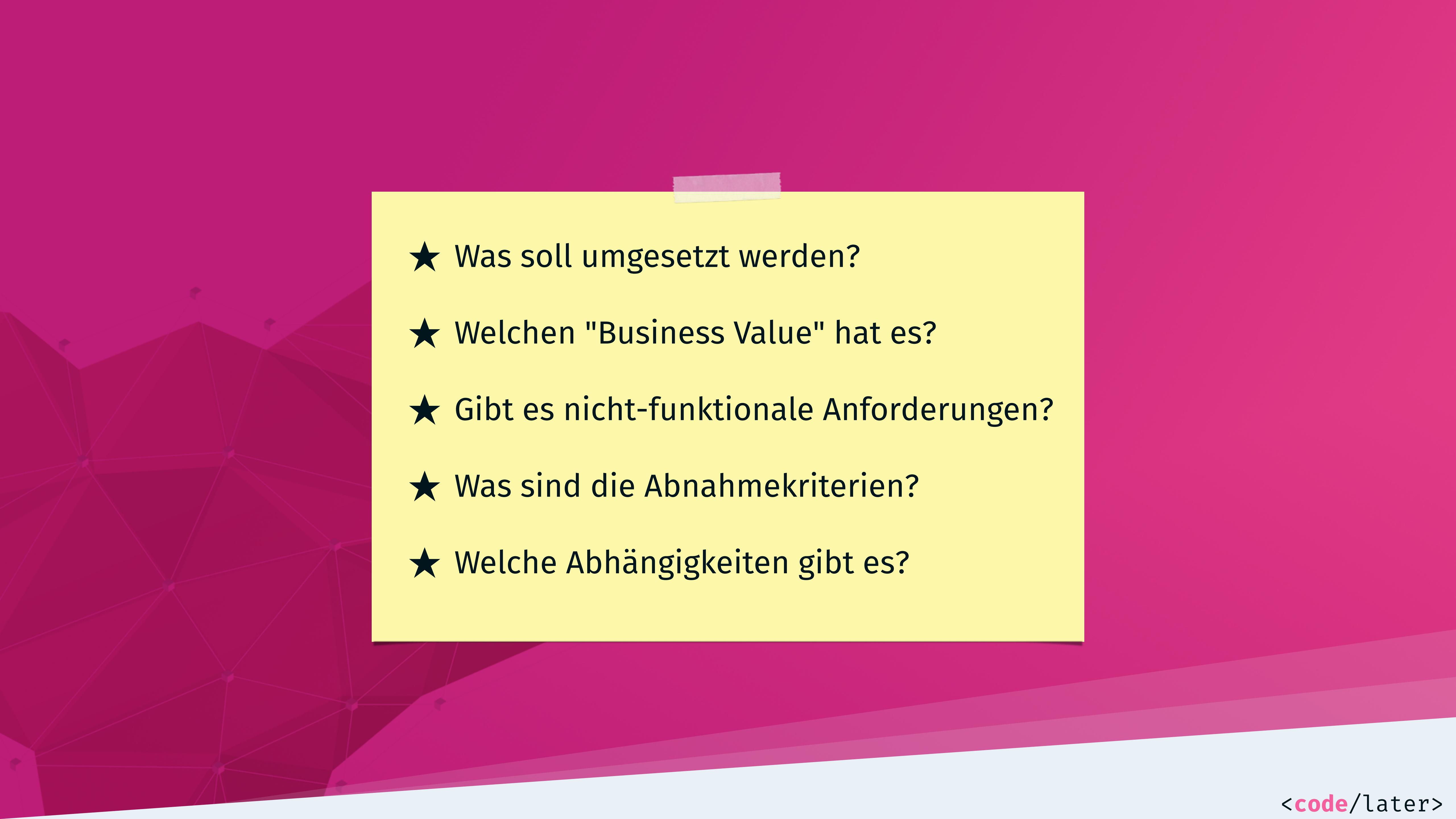
Auslieferung (max. 1)



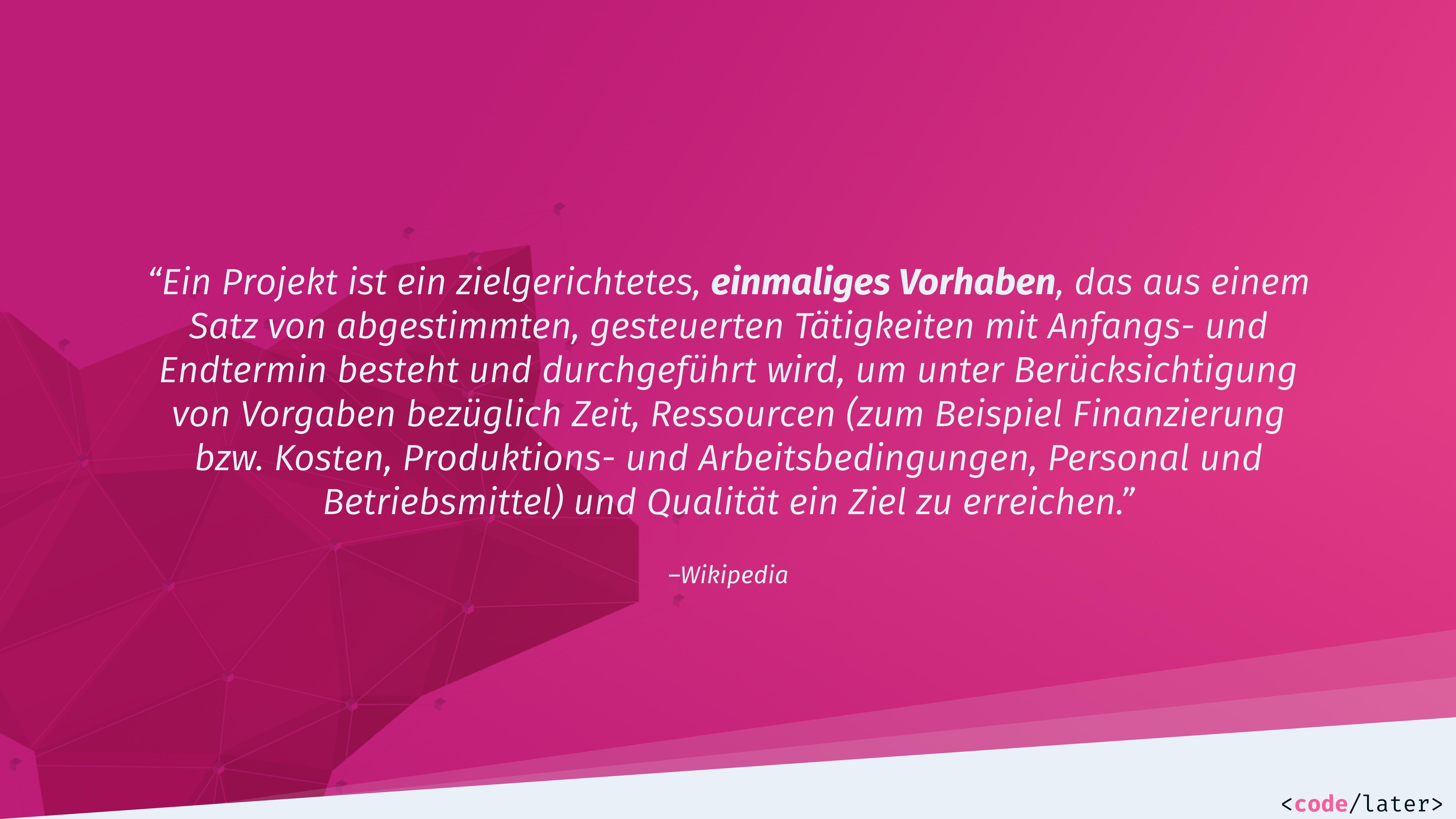
Arbeitsfluss



<code/later>

- 
- ★ Was soll umgesetzt werden?
  - ★ Welchen "Business Value" hat es?
  - ★ Gibt es nicht-funktionale Anforderungen?
  - ★ Was sind die Abnahmekriterien?
  - ★ Welche Abhängigkeiten gibt es?

# GitHub Project Boards

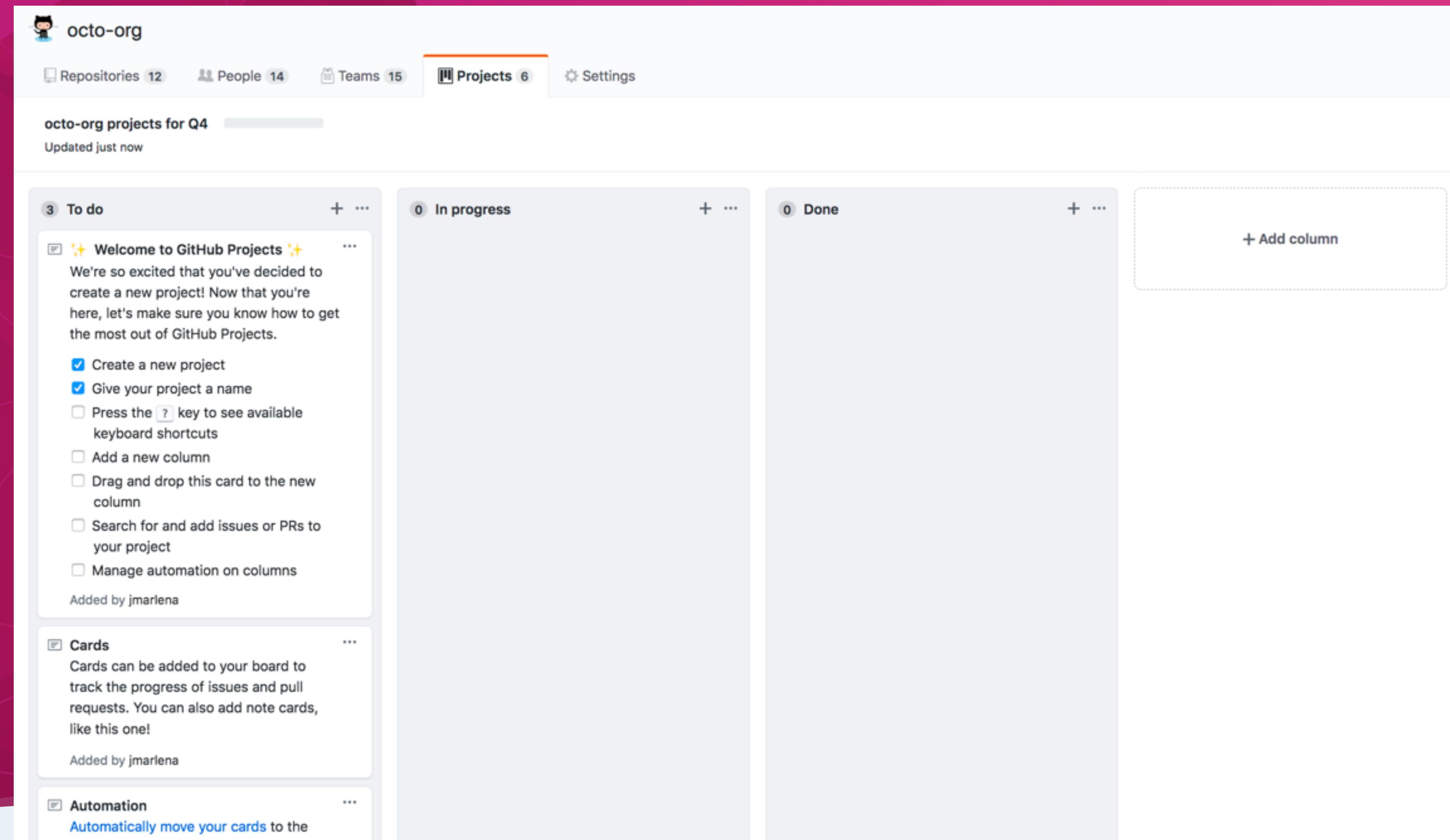


*“Ein Projekt ist ein zielgerichtetes, **einmaliges Vorhaben**, das aus einem Satz von abgestimmten, gesteuerten Tätigkeiten mit Anfangs- und Endtermin besteht und durchgeführt wird, um unter Berücksichtigung von Vorgaben bezüglich Zeit, Ressourcen (zum Beispiel Finanzierung bzw. Kosten, Produktions- und Arbeitsbedingungen, Personal und Betriebsmittel) und Qualität ein Ziel zu erreichen.”*

–Wikipedia

- Trifft auf Projektarbeiten im Studium zu
- Sowohl im Unternehmen als auch bei Open Source handelt es sich aber eher um Produkte und/oder Systeme
- **Merke: Softwaresysteme sind in der Regel nie fertig**

# Product GitHub Project Boards



The screenshot shows the GitHub Project Boards interface for the organization 'octo-org'. The top navigation bar includes 'Repositories 12', 'People 14', 'Teams 15', 'Projects 6' (which is selected), and 'Settings'. The main title is 'octo-org projects for Q4' (Updated just now). The interface features three columns: 'To do', 'In progress', and 'Done'. The 'To do' column contains a card titled 'Welcome to GitHub Projects' with the following text: 'We're so excited that you've decided to create a new project! Now that you're here, let's make sure you know how to get the most out of GitHub Projects.' Below this, there is a list of checkboxes: 'Create a new project' (checked), 'Give your project a name' (checked), 'Press the ? key to see available keyboard shortcuts' (unchecked), 'Add a new column' (unchecked), 'Drag and drop this card to the new column' (unchecked), 'Search for and add issues or PRs to your project' (unchecked), and 'Manage automation on columns' (unchecked). The card was added by 'jmarlena'. Below the 'To do' column, there are sections for 'Cards' and 'Automation'. A button '+ Add column' is located at the bottom right of the board area.

# Code Review

# Code Review

- Mehr Leute sehen mehr Fehler
- Wissensteilung
- Code Ownership aufbrechen
- Kommunikation
- Kollaboration

# Code Review

- Sind Teil des Jobs
- Genauso gewissenhaft zu erledigen wie die Entwicklung selbst
- Keine Garantie für keine Fehler ;-)

# Reviewee

- Freundlich, respektvoll und sachlich
- Kleine, zusammengehörige Änderungen (< 400 LOC)
- Warum sind die Änderungen nötig?
- Welche Entscheidungen/Kompromisse wurden getroffen? Und warum?
- Auswirkungen auf die Software / den Betrieb / das Team?
- Besonders zu beachtende Punkte

# Reviewer

- Freundlich, respektvoll und sachlich
- Lob aussprechen
- Fragen statt Anweisungen geben
- Wurden alle Anforderungen erfüllt?
- Entsprechen die Änderungen den Qualitätsansprüchen?
- Gibt es Verständnisprobleme?
- Dokumentation / Tests vorhanden?

# Beispiel

# Automatische Reviews

- Statische Codeanalyse
- Einhalten von Styleguides
- Analyse auf Schwachstellen
- Ausführen der Test Suite

# Automatische Tests

# Manuelles Testing

- Notwendig (vor allem für UI/UX)
- Aufwändig
- Zeitintensiv
- Fehleranfällig

# Automatische Tests

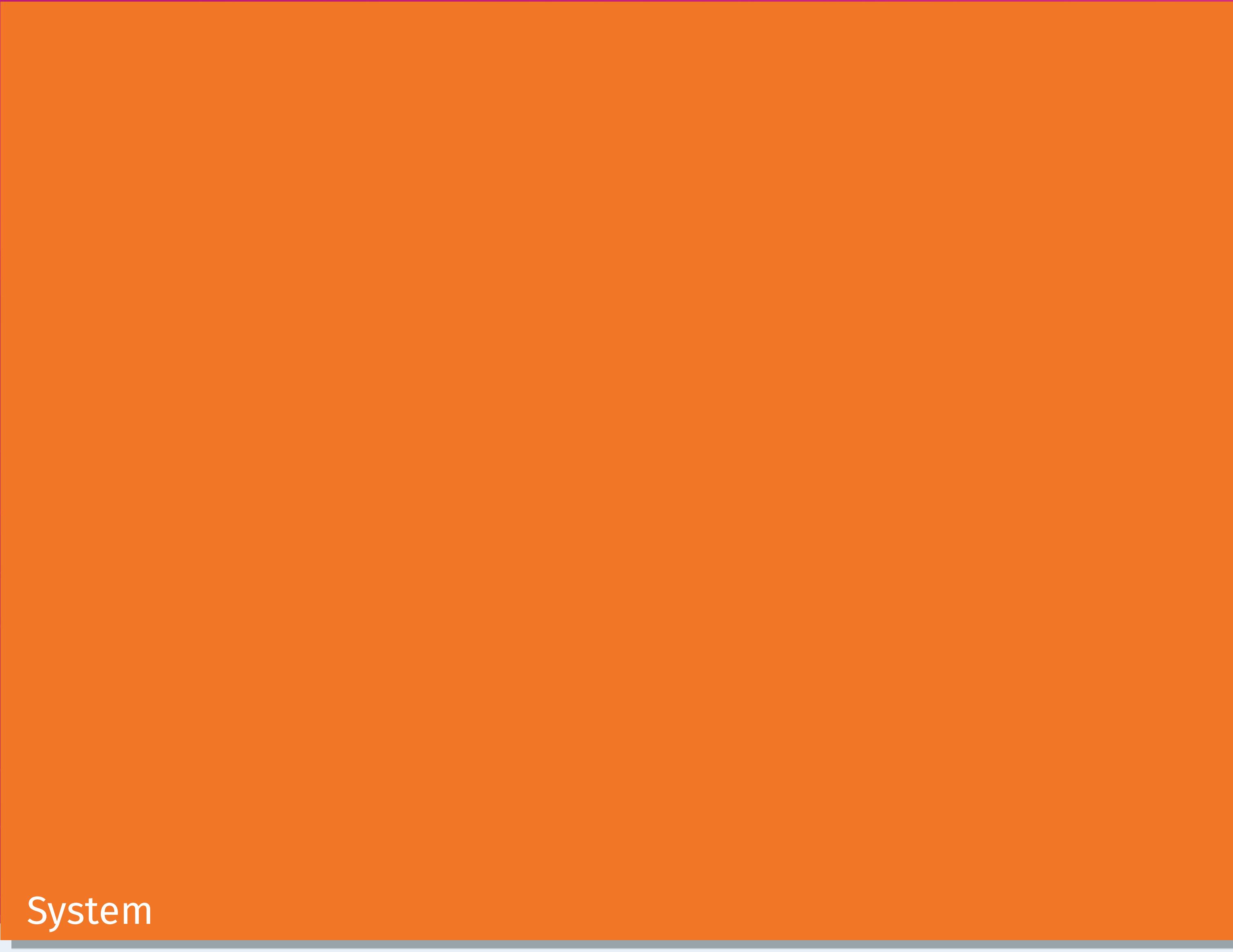
- Kein "Nice to have"
- Keine Garantie für perfekte Software
- Geben Sicherheit bei
  - Weiterentwicklung
  - Refactorings
  - Updates



<code/later>

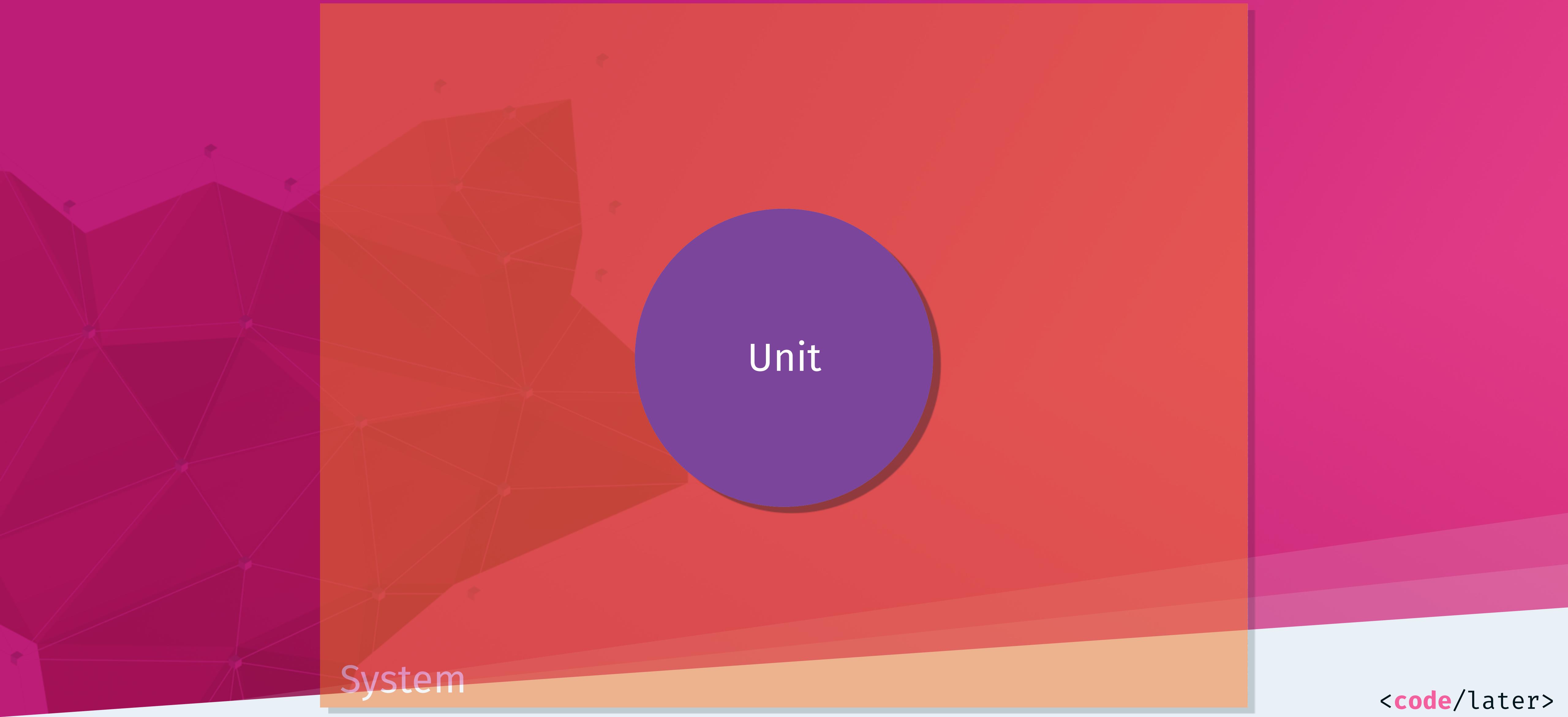
# Unit Tests & Acceptance Tests

# Unit Test

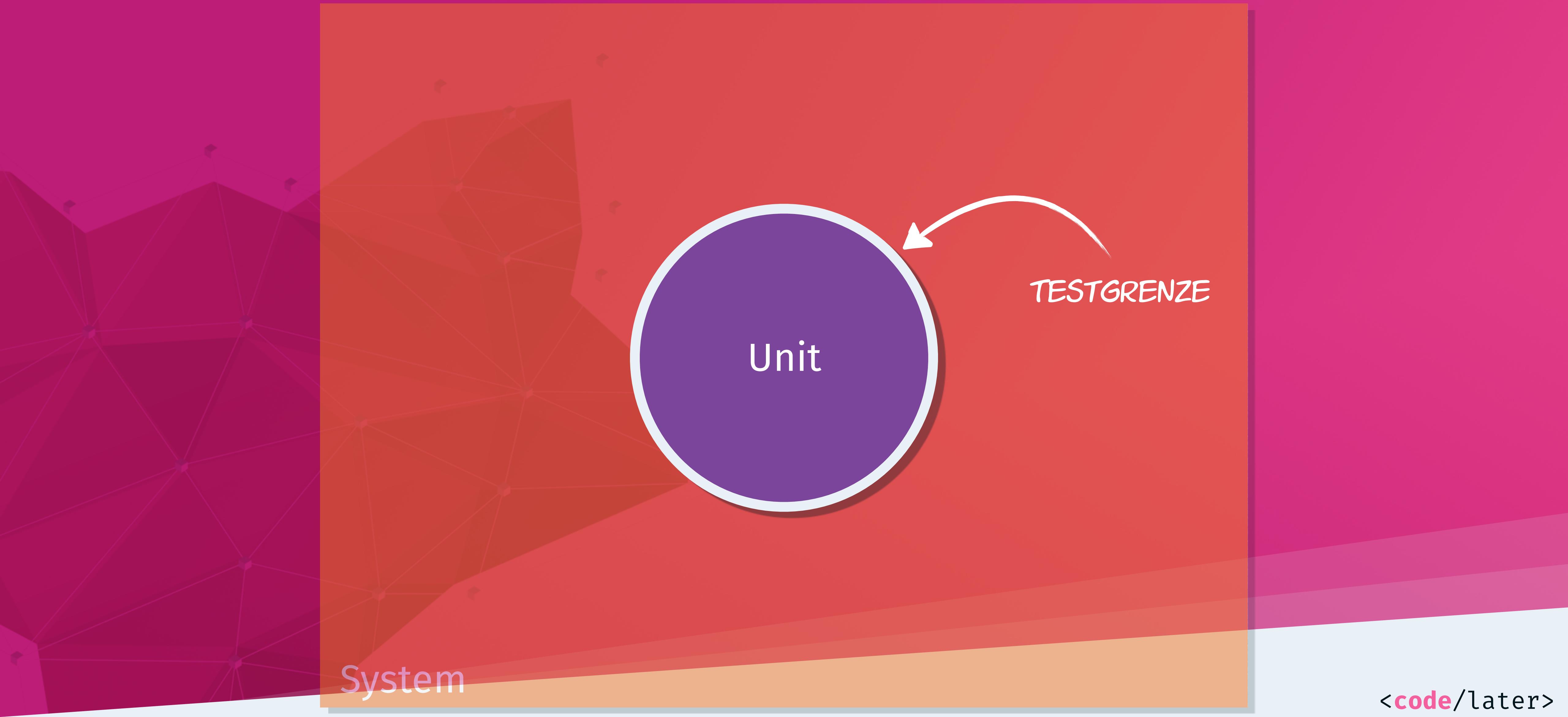


<code/later>

# Unit Test



# Unit Test

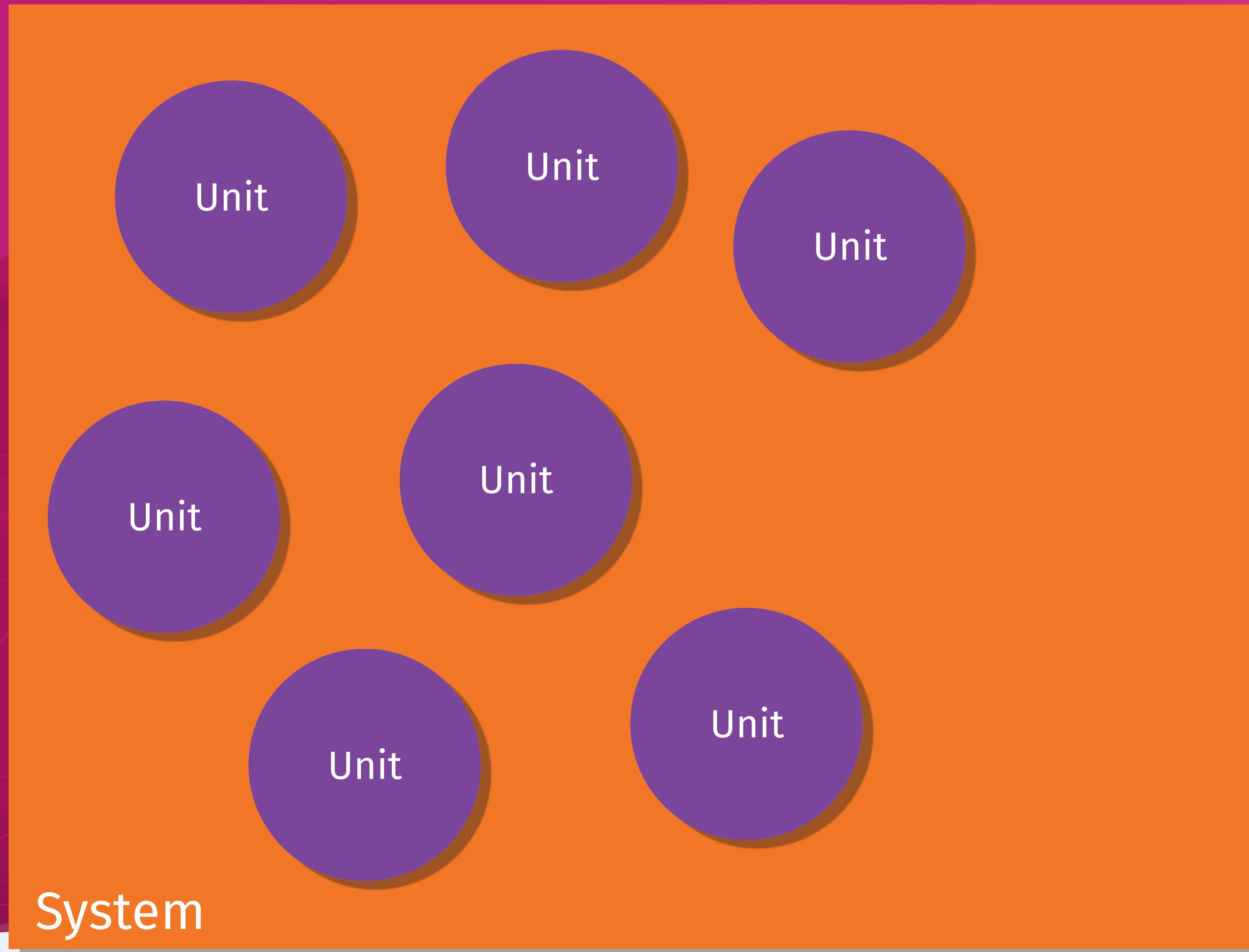


# Acceptance Test

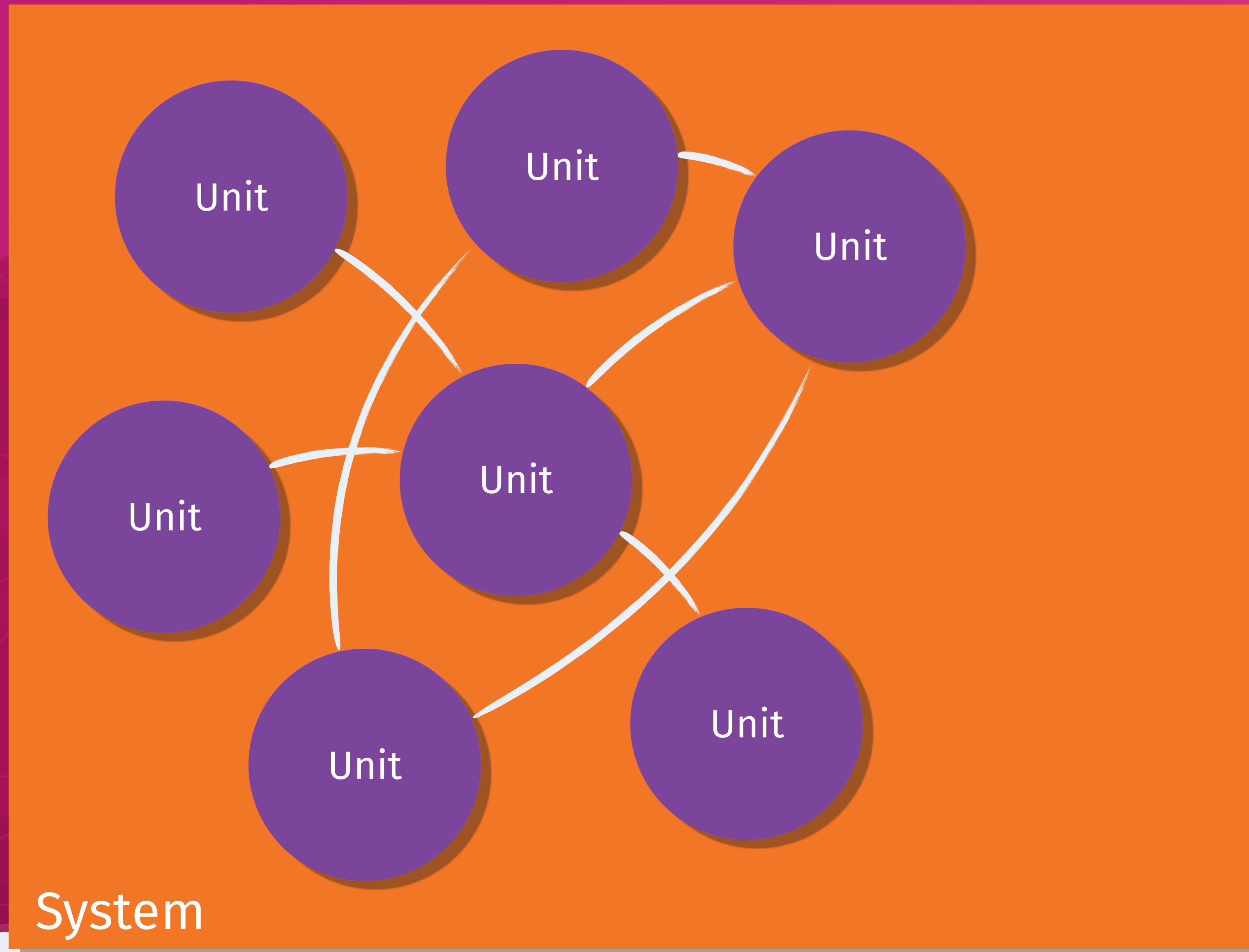
System

<code/later>

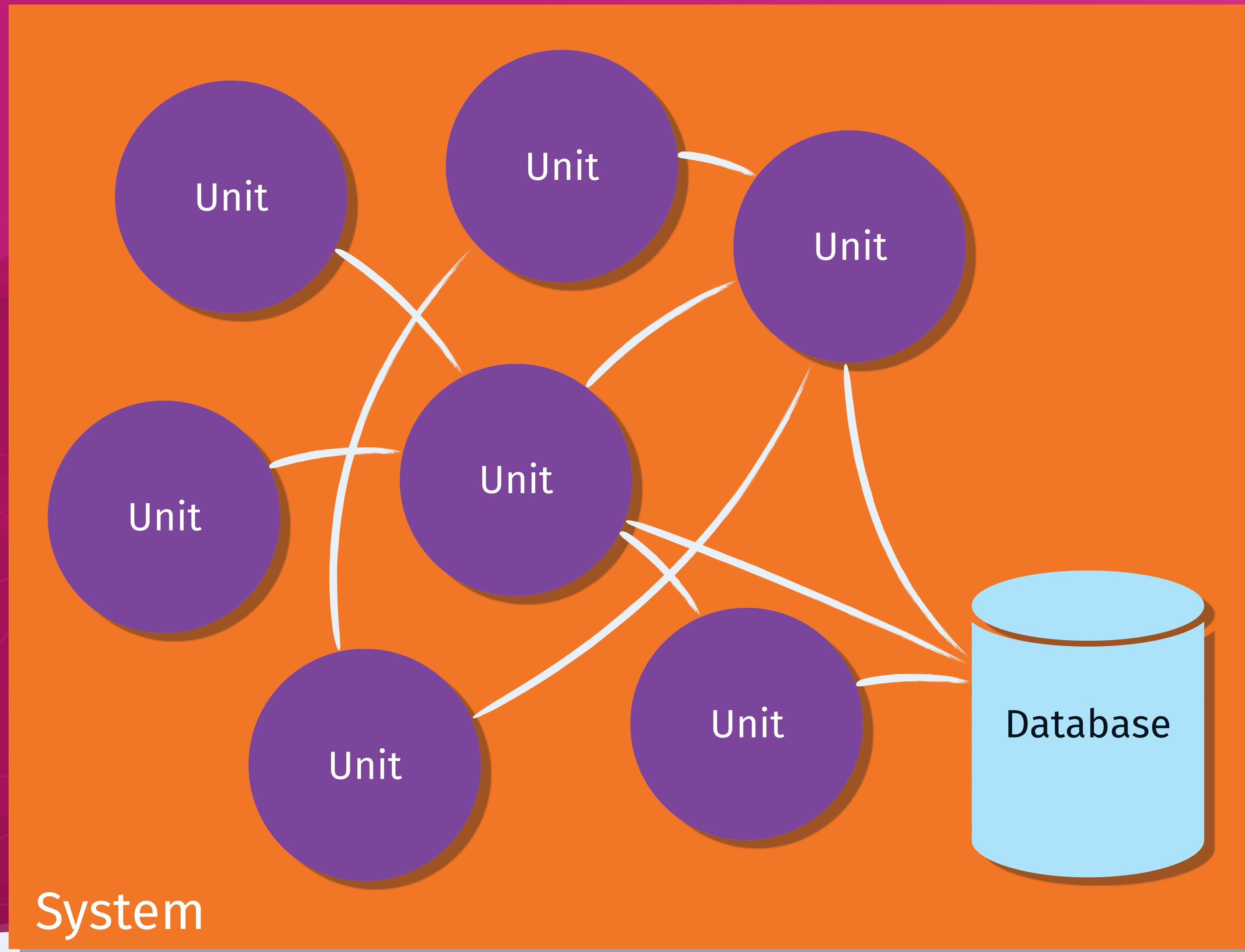
# Acceptance Test



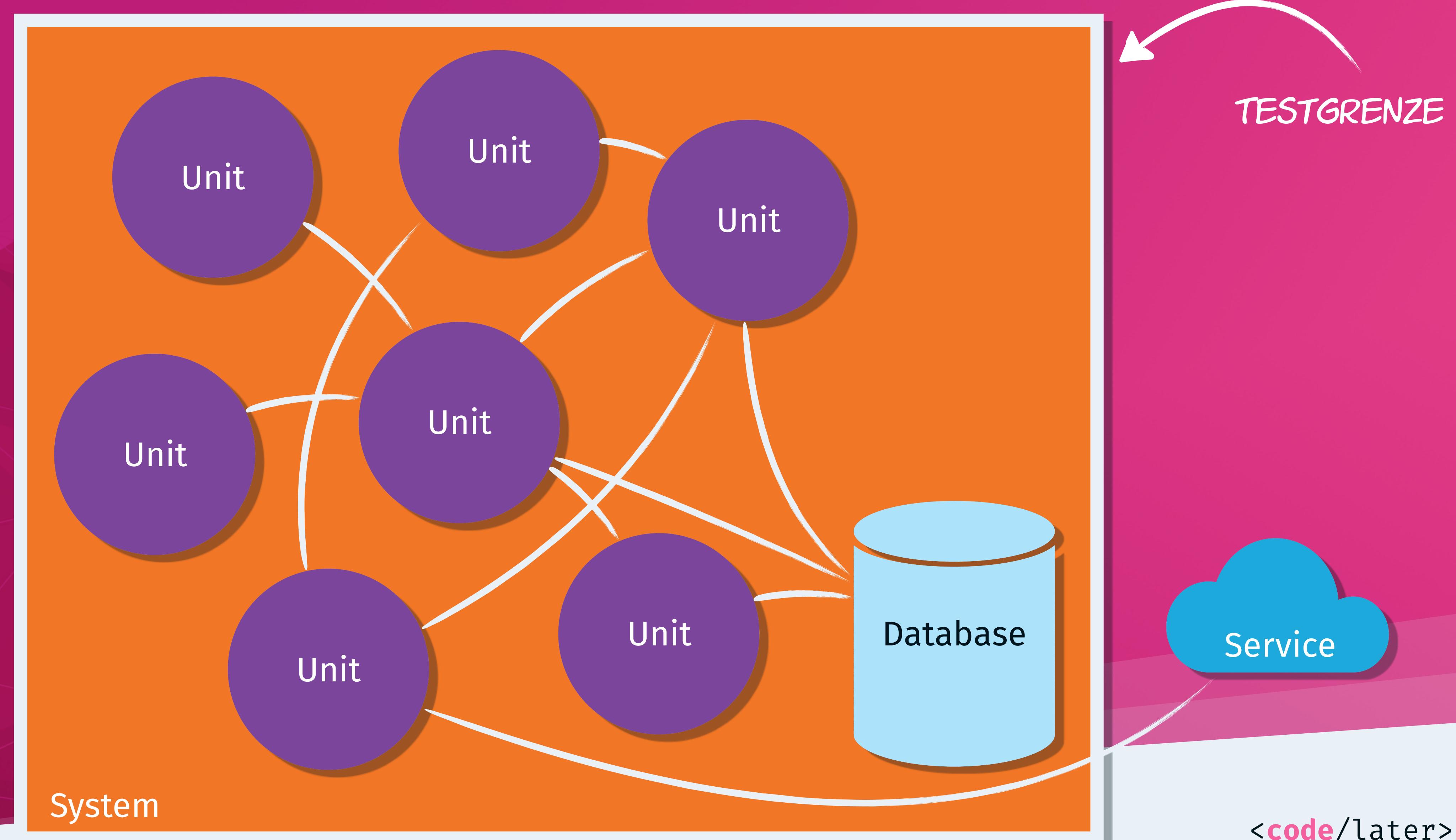
# Acceptance Test



# Acceptance Test



# Acceptance Test



# Test Struktur

- Setup
- Ausführung
- Validierung
- Aufräumen

# Unit Test

## System under Test (SUT)

```
const adder = (x, y) => x + y
```

```
test('add two numbers', () => {
```

```
    expect(adder(2, 1)).toBe(5)
```

```
})
```

Assertion

Isolierter Test

Errechneter Wert

Erwarteter Wert

```
$ npx jest __tests__/simple.test.js
PASS __tests__/simple.test.js
  ✓ add two numbers (3ms)
```

Test Suites: 1 passed, 1 total  
Tests: 1 passed, 1 total  
Time: 0.519s, estimated 1s

```
$ npx jest __tests__/simple.test.js
FAIL __tests__/simple.test.js
× add two numbers (6ms)
```

- add two numbers

```
expect(received).toBe(expected) // Object.is equality
```

```
Expected: 5
Received: 3
```

```
2 |
3 | test.only('add two numbers', () => {
> 4 |   expect(add(2, 1)).toBe(5)
|   ^
5 | })
6 |
```

```
at Object.toBe (__tests__/simple.test.js:4:23)
```

# Best Practises

- Keine langsamten Tests
- Fokussierte Tests
- Keine Abhängigkeiten zwischen Tests
- Keine Implementierung Details 😐

# Test-Driven Development

- Zuerst den Test implementieren
- Dann die Funktionalität
- In kleinen Schritten
- Schnell ausführbar

# Test-Driven Development

- (Sehr) Kurze Entwicklungszyklen
- Einfacheres Software Design
- Vertrauen in die Korrektheit der Implementierung
- "Synonyme": Behaviour-Driven Development, Test-First Development



# Tests sind auch Code

# Hands-On



# JEST\*

<https://jestjs.io/>

\*Nur eins von vielen: <https://blog.bitsrc.io/top-javascript-testing-frameworks-in-demand-for-2019-90c76e7777e9>

# Demo

# Wörter zählen

- Häufigkeiten von Wörtern in einem Text
- Wörter müssen **mindestens 1 Zeichen** haben
- Keine Unterscheidung nach Groß- und Kleinschreibung
- Ausgabe absteigend sortiert nach Häufigkeit

# Automatische Reviews

# Automatische Reviews

- Viele Dinge in einem Code Review sind für Menschen mühselig...
- ...aber für Computer einfach:
  - Korrekte Einrückung
  - Nutzung von Variablen
  - Weitergabe von Parametern
  - Verwendung alter (unsicherer) Bibliotheken
  - uvm

# Statische Analyse

- Analyse des Codes ohne, dass das Programm laufen muss
- In kompilierten Sprachen übernimmt diese Aufgabe (zum Großteil) der Compiler
- In Skript-Sprachen unterschiedlich schwer
- JavaScript lässt sich gut statisch analysieren

# eslint

- Prüft Code gegen bestimmte Regeln
  - Formatierung
  - Fehlerquellen
  - Performanceprobleme
  - uvm

```
function myFunc (x) {  
    return x * x  
}  
console.log(myFunc(2))  
  
// Further down in the file  
myFunc = (x, y) => {  
    return x * y  
}  
console.log(myFunc(2, 3))
```

# Hands-On

# Acceptance Tests



- Ende-zu-Ende Testing
- Testet Web-Anwendungen im Browser



# Fernbedienung für den Browser

```
beforeEach(() => {
  cy.visit('https://example.cypress.io/commands/actions')
})

it('.submit() - submit a form', () => {
  // https://on.cypress.io/submit
  cy.get('.action-form')
    .find('[type="text"]').type('HALFOFF')
  cy.get('.action-form').submit()
    .next().should('contain', 'Your form has been submitted!')
})
```

# Best Practises

- Keine langsamten Tests 🤔
- Fokussierte Tests 🤔
- Keine Abhängigkeiten zwischen Tests
- Keine Implementierung Details

# Hands-On

# Continuous Integration

# Continuous Integration

- Erinnerung: Unabhängiges arbeiten in Branches ist sinnvoll
- Problem: Wann (und wie) erkennen wir Probleme bei der Integration der einzelnen Branches
- Für diese Probleme bedarf es nicht einmal einen Merge-Konflikt

Feature A



Nutzt

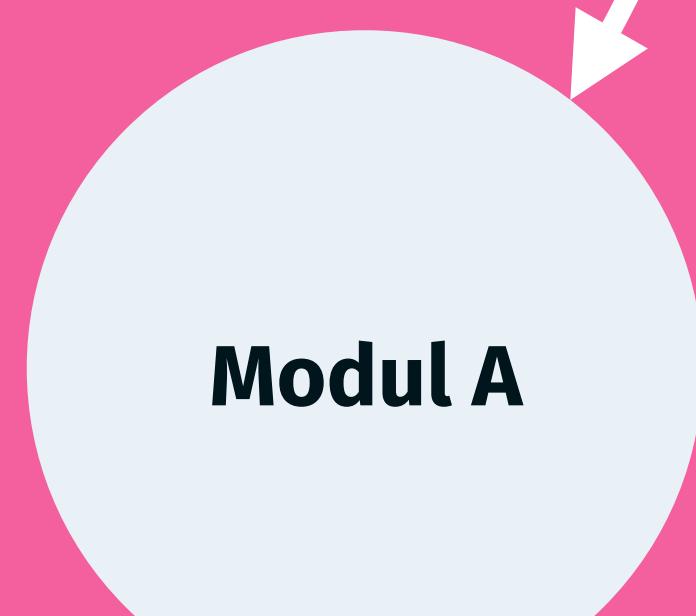
Modul A

master

Feature B



Nutzt



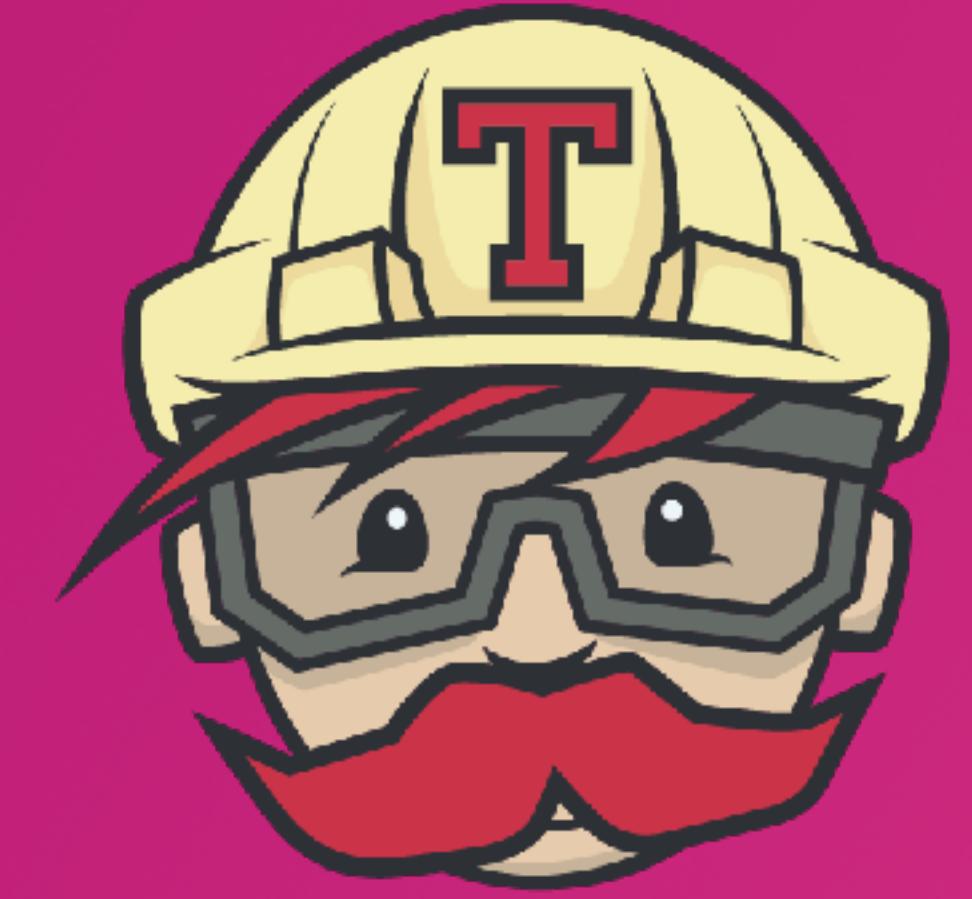
Erweitert

<code/later>

- Syntaktisch keine Fehler
- Review bei beiden gemacht
- Tests in jedem Branch OK
- **Aber** Änderungen an **Modul A** in **Feature B** brechen die (vorher korrekte) Funktionalität in **Feature A**
- Im master Branch 😱

# Lösung(en)

- master Branch regelmäßig in Feature Branch mergen
- Nach jeder Änderung auf dem Feature Branch **alle** Tests ausführen
- Vor der Zusammenführung von master und Feature Branch sicherstellen, dass die Tests danach noch grün sind



https://travis-ci.org

# Hands-On