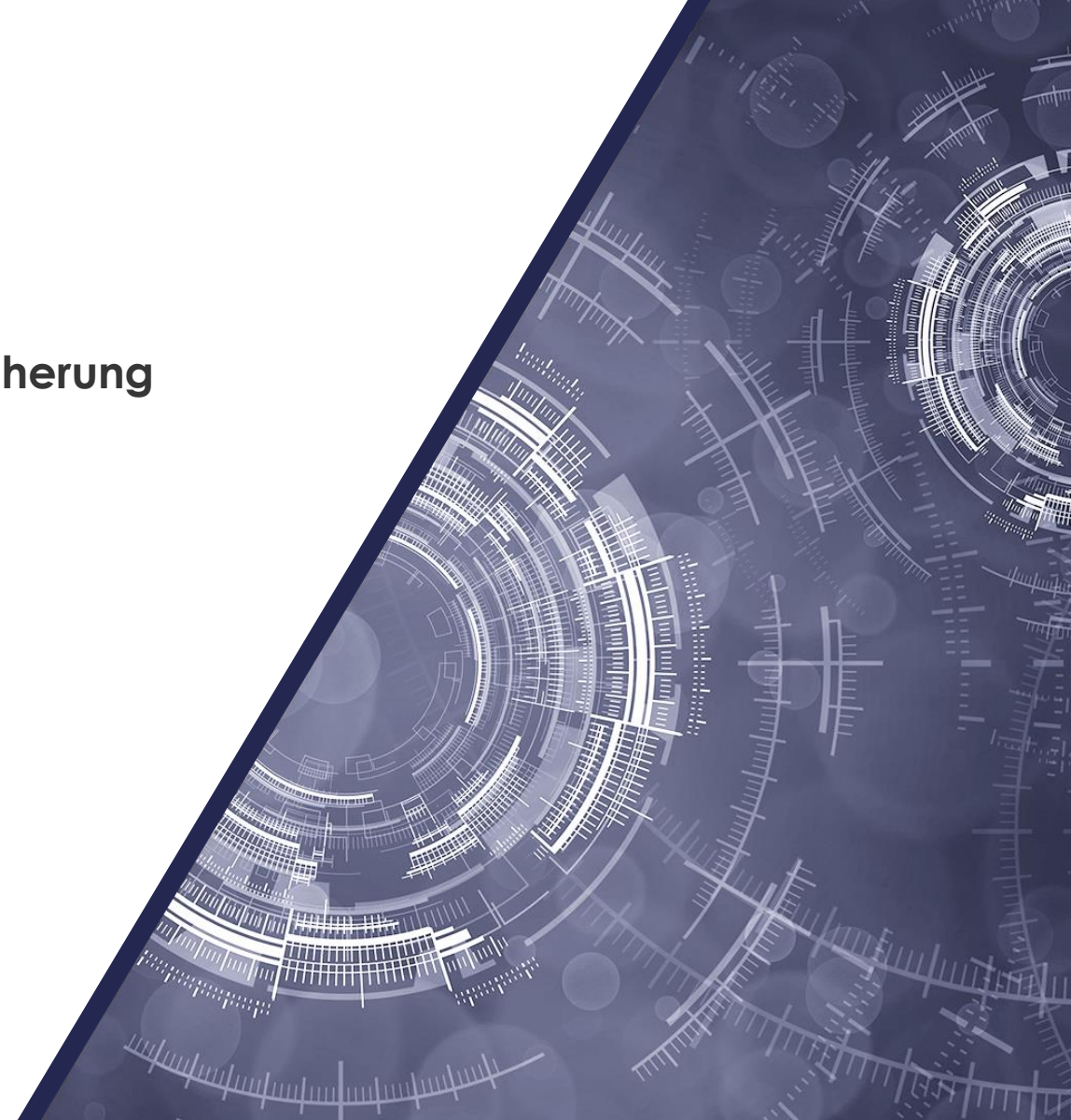
# Agiles Testen

**180.764 Software-Qualitätssicherung**

21.11.2024 - WS 2024

Wolfgang Gruber

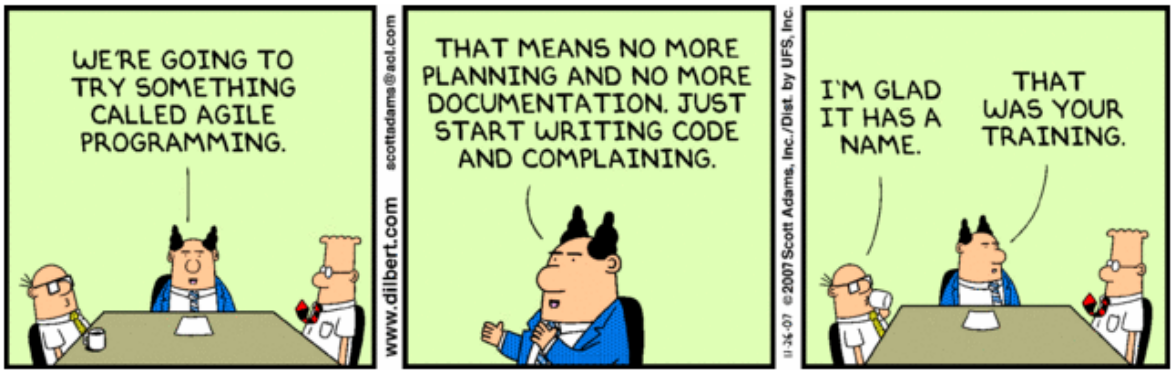
Research Group for Industrial Software (INSO)
https://www.inso.tuwien.ac.at



# Agenda

- Grundlagen der agilen Software-Entwicklung
- Agile Vorgehensmodelle
  - Extreme Programming
  - Scrum
  - Kanban
- Agile Testing
  - Agile Testmethoden
  - Testautomatisierung
  - Agile Testquadranten

# Grundlagen der agilen Software-Entwicklung



# Herausforderungen in IT-Projekten

- Anforderungen und Ziele sind nicht immer klar
- Anforderungen können sich schnell ändern
- Anforderungen können sich spät im Projekt ändern
- Technologien ändern sich schnell
- Geringe Wiederverwendbarkeit zwischen Projekten
- Software-Entwicklung liegt zwischen Handwerk und Kunst

# Traditionelles Projekt-Management

- Top-Down-Ansatz („command and control principle“)
- Projektmanager sind für das Projekt verantwortlich
- Der Kunde liefert die Anforderungen
- Push-Prinzip: Der Projektmanager weist die Tätigkeiten den Teammitgliedern zu
- Tests und QS werden erst spät im Projekt tätig
- Das Produkt wird erst zu einem späten Zeitpunkt an den Kunden geliefert
- Der Projektmanager versucht Unsicherheiten zu reduzieren und gleichzeitig die Vorhersagbarkeit zu erhöhen
- Oftmals organisatorische Trennung zwischen Entwicklung, Test und Betrieb
- Prozesse und Dokumentation dienen im Problemfall als Verteidigung

# Manifest für Agile Softwareentwicklung

2001 von einer Gruppe renommierter Persönlichkeiten aus dem Software Engineering-Bereich definiert.

Es definiert zentrale Werte für die Softwareentwicklung:

- Individuen und Interaktionen mehr als Prozesse und Werkzeuge
- Funktionierende Software mehr als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
- Reagieren auf Veränderung mehr als das Befolgen eines Plans

„Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein."

Quelle: https://agilemanifesto.org/iso/de/manifesto.html

# 12 Prinzipien des Agilen Manifests 1 / 2

- Kundenzufriedenheit durch die frühe und kontinuierliche Auslieferung von wertschaffender Software
- Änderungen an den Anforderungen sind selbst spät in der Entwicklung willkommen
- Funktionierende Software wird regelmäßig innerhalb von Wochen oder Monaten ausgeliefert, wobei kürzere Intervalle zu bevorzugen sind
- Fachexperten und Entwickler müssen täglich zusammenarbeiten
- Das Team benötigt eine entsprechende Umgebung und Unterstützung, sowie Vertrauen, dass sie die Arbeit erledigen
- Die effizienteste und effektivste Methode für die Weitergabe von Informationen ist von Angesicht zu Angesicht

# 12 Prinzipien des Agilen Manifests 2 / 2

- Der Projektfortschritt wird anhand der funktionierenden Software gemessen
- Durch eine konstante Arbeitsgeschwindigkeit soll eine nachhaltige Entwicklung erreicht werden
- Durch Einfachheit wird die Menge nicht-getaner Arbeit maximiert
- Technische Exzellenz und gutes Design erfordern kontinuierliche Aufmerksamkeit
- Selbstorganisierte Teams liefern die besten Anforderungen, Architekturen und Entwürfe
- Das Team reflektiert in regelmäßigen Abständen, um effektiver zu werden

# Agile Vorgehensmodelle

# Agile Vorgehensmodelle

## Vorgehensmodelle für einzelne Teams

- Extreme Programming
- Scrum
- Kanban
- Crystal Family
- Lean Software Development

## Vorgehensmodelle für mehrere Teams

- Scrum of Scrums
- Nexus Framework
- Scaled Agile Framework (SAFe)
- Large Scale Scrum (LeSS)
- Scaling Agile @ Spotify

# Extreme Programming (XP)

- Ende der 90er von Kent Beck, Ward Cunningham und Ron Jeffries entwickelt
- Definiert 5 Werte und leitet davon 14 Prinzipien ab, die wiederum die Basis für 23 Hauptpraktiken und 11 davon abgeleiteten Praktiken bilden
- Praktiken haben sich teilweise bereits vorher bewährt
- Wesentlicher Einfluss auf das agile Manifest

# Praktiken von Extreme Programming (Auszug)

- Einbeziehung des Kunden
- Iterative Entwicklung (Wöchentliche & quartalsmäßige Zyklen)
- Requirements als Stories
- Einzelne Code-Basis
- Pair Programming
- Inkrementelles Design
- Test-First Programming
- Continuous Integration

# Scrum

- Bezeichnung „Scrum“: Methode, um in Rugby das Spiel erneut zu starten
- Entwickelt von Ken Schwaber & Jeff Sutherland und erstmalig auf der OOPSLA 1995 präsentiert
- Ab 2014 Veröffentlichung des Scrum Guides auf https://scrumguides.org
- Iterativer Prozess unterteilt in Sprints mit fixer Zeitdauer („timeboxed“)
- Mittels „Definition of Done“ wird festgelegt, welches Maß an Qualität benötigt wird, um etwas abzuschließen

# Rollen in Scrum

## Product Owner

- Schnittstelle zwischen Kunden & Team
- Sammelt und priorisiert Anforderungen (User Stories)
- Fungiert als Ansprechpartner für fachliche Fragen

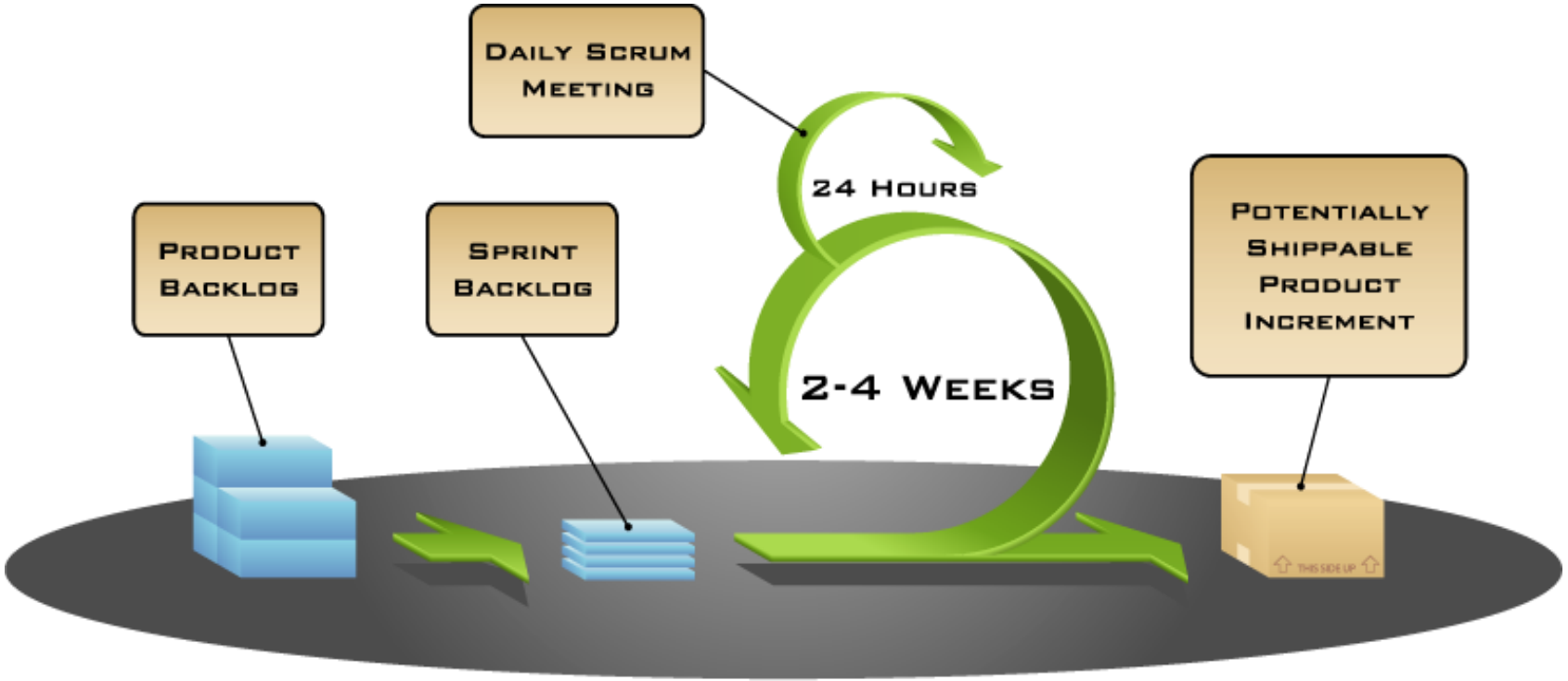
## Scrum Master

- Sorgt für die Einhaltung des Prozesses
- Moderiert die Meetings, trifft aber keine Entscheidungen
- Beseitigt Blockaden (eng. Impediments) für das Team

## Team Member

- Schätzt Anforderungen
- Arbeitet am Projekt

# Scrum – Prozessmodell




Quelle: https://www.mountaingoatsoftware.com/agile/scrum/resources/overview

# Scrum – Sprints

- Fixe Timebox von 1 – 4 Wochen mit definiertem Umfang
- Beginnt mit Sprint Planning Meeting, bei dem die umzusetzenden Items geschätzt und festgelegt werden
- Daily Scrum Meeting zur internen Abstimmung
- Endet mit
  - Sprint Review Meeting: Ergebnis des Sprints wird Stakeholdern präsentiert
  - Sprint Retrospective: Zur Verbesserung der Qualität und Effektivität
- Sprints können nur abgebrochen, aber nicht verlängert werden
- Nicht fertiggestellte Items werden in den nächsten Sprint verschoben

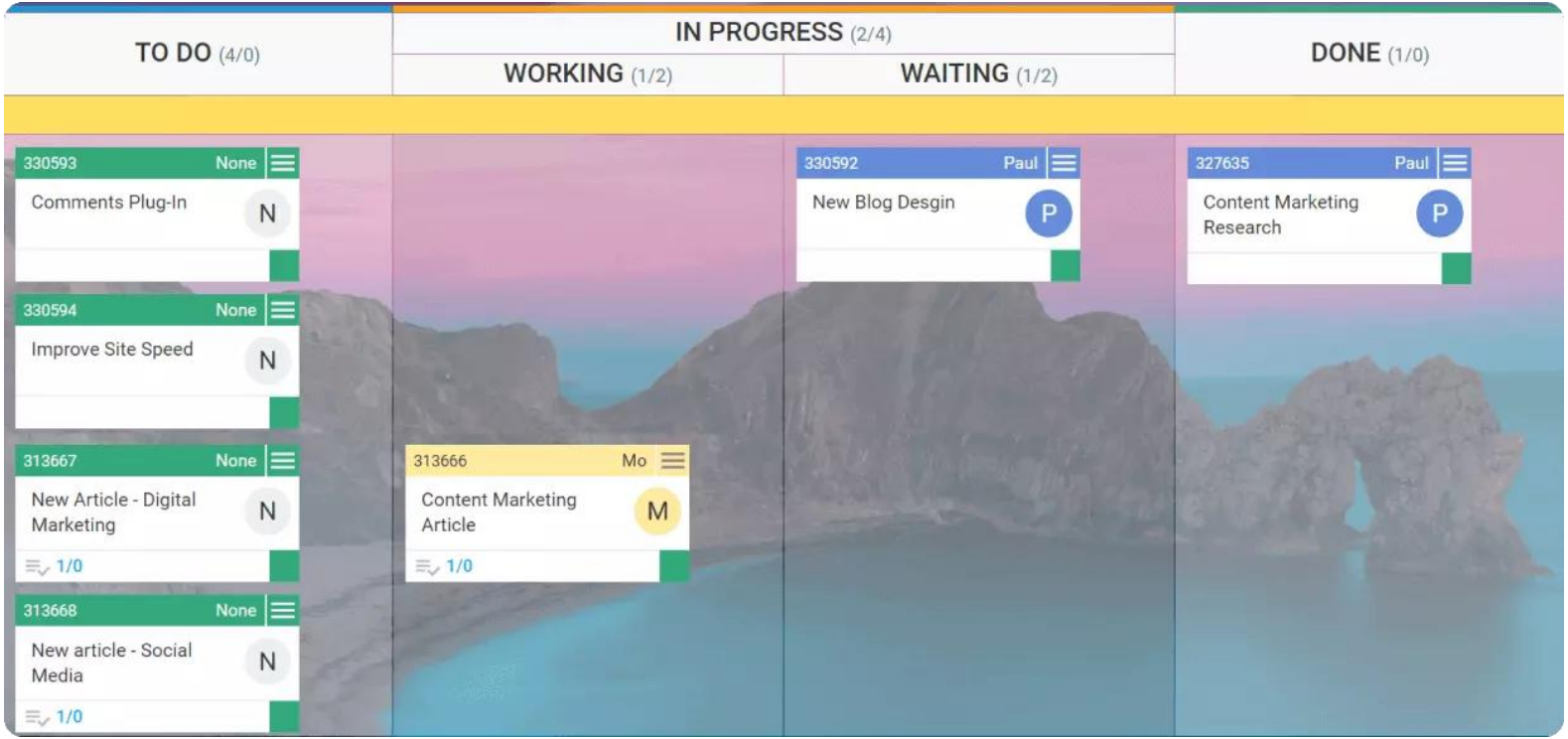
# Kanban

- Aus dem Japanischen für Karte oder Schild
- Basiert auf der gleichnamigen Methode zur Produktionssteuerung aus dem Toyota Production System (TPS)
- Von David J. Anderson für die Software-Entwicklung adaptiert
- Pull-System für konstante Arbeitsgeschwindigkeit
- Leichtgewichtiges System, um Widerstand bei der Einführung zu vermeiden
- Oftmals in Verbindung mit Scrum verwendet => Scrumban

# Prinzipien von Kanban

- Visualisierung des Workflows
- Limitierung des Work-in-Progress
- Verwaltung des Flows
- Regeln des Prozesses
- Gemeinsame Verbesserung

# Kanban-Board 1 / 2




Quelle: https://businessmap.io/de/kanban-ressourcen/kanban-software-de/kanban-boardbeispiele

# Kanban-Board 2 / 2

- Bietet Überblick über alle offenen Aufgaben und deren aktuellen Stage
- Stages definieren in Form von Spalten den Workflow
  - Stages können zusätzlich unterteilt werden
- Optionale Swimlanes entlang einer Zeile können zur Gruppierung von Aufgaben dienen (z.B. nach Team-Mitglied, Typ der Aufgabe)
- Können physisch (z.B. als Whiteboard) oder in Form einer Software (z.B. Trello, Jira) verwendet werden

# Fallstricke

- Auch agile Entwicklung ist keine Silver-Bullet
- Agile Entwicklung wird oft mit Scrum gleichgesetzt
- Dark Scrum: Missbrauch der Ideen von Scrum
  - Keine Änderung der Kultur, sondern nur Umbenennung, z.B. Projekt-Manager wird zu Scrum Master
  - Keine Selbstorganisation des Teams
  - Micromanagement, z.B. Daily wird als Status-Meeting missbraucht
  - Zombie Scrum: Stures Befolgen der Regeln ohne Agilität zu leben

# Diskussion

*Diskussion – Agile Vorgehensmodelle*

*Haben Sie bereits Erfahrung mit agilen Vorgehensmodellen?*

*Welche positiven oder negativen Aspekte sind Ihnen ausgefallen?*

# Agiles Testen

# Agiles Testen?

- Wo ist in Scrum der Test?
- Wo kommt der Tester ins Spiel?
- Muss ich jetzt alles neu lernen?
- Gibt es spezielle Testmethoden?

# Agiles Testen

- Überbegriff für qualitätssichernde Maßnahmen in agilen Projekten
- Baut auf bestehendem Stand der Technik auf
- Folgt den Prinzipien der agilen Software-Entwicklung, u.a.
  - Fokus auf Zusammenarbeit und Kommunikation
  - Qualitätssicherung von Beginn des Projekts an
  - Iterative Entwicklung
- Innovation und Weiterentwicklung des Testens hauptsächlich im Bereich des agilen Testens

# Die Rolle des agilen Testers

- Im ganzen Lebenszyklus involviert
- Bringt Wissen und Erfahrung aus dem Testen ins Projekt
- Sorgt für Einführung & Einhaltung der qualitätssichernden Maßnahmen im Projekt
  - Teststrategie
  - Auswahl und Unterstützung bei Verwendung der Testmethoden
  - Aufbereitung Testdaten
  - Erstellen von Reports und Berichten
- Identifiziert Risiken im Produkt
- Unterstützt Analysten bei der Definition von User Stories & Akzeptanztests
- Unterstützt Entwickler bei Entwicklung von automatisierten Tests

# Whole-Team-Approach

- Interdisziplinäres („cross functional“), selbstorganisiertes Team, das Fachbereich bzw. Domänen-Experten, Entwickler, Tester und weitere Spezialisten einbezieht => „Power of Three“
- Gesamtes Team ist für die benötigte Qualität verantwortlich
- Auch Entwickler benötigen Wissen über Testen (z.B. für Entwicklung von Unit-Tests)
- „Two-Pizza Team Rule“ von Amazon: Teams sollten mit zwei Pizzen versorgt werden können

# Agile Testmethoden

## Test-Driven Development (TDD)

- dt. Testgetriebene Entwicklung
- Entwickler-fokussiert

## Behaviour-Driven Development (BDD)

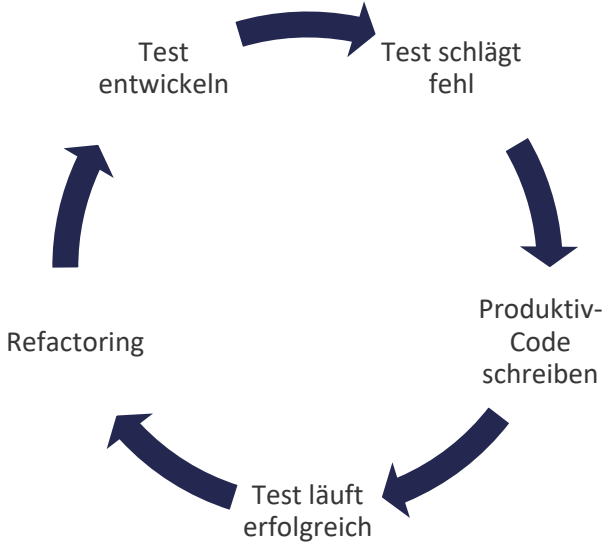
- dt. Verhaltensgetriebene Entwicklung
- Kunden-fokussiert

## Acceptance Test-Driven Development (ATDD)

- dt. Abnahmetestgetriebene Entwicklung
- Kunden-fokussiert

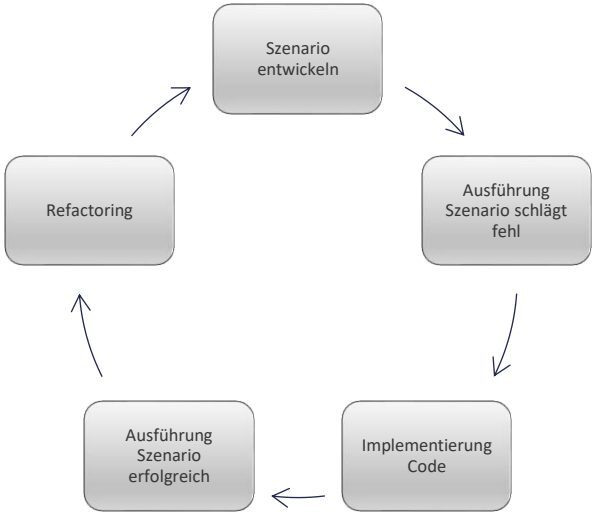
# Test-Driven Development (TDD) - Recap

- Verfahren, bei dem automatisierte Tests vor dem Produktiv-Code entwickelt werden
- Führen zu einem besseren Design des Produktiv-Code
- Sicherheitsnetz für Entwickler bei Erweiterungen, Refactorings und Bugfixes
- xUnit-Frameworks für viele Programmiersprachen verfügbar



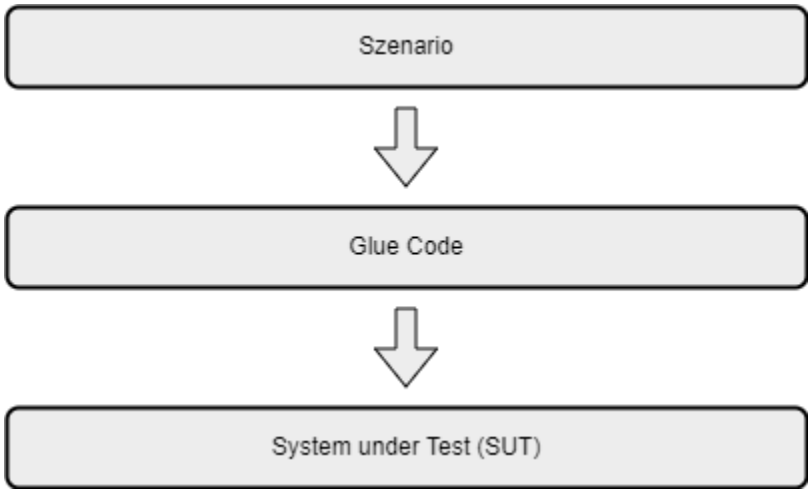
# Behaviour-Driven Development (BDD)

- 2006 von Dan North vorgestelltes kunden-fokussiertes Verfahren
- Product Owner bzw. Analyst, Entwickler und Tester entwickeln gemeinsam Beispiele (= Szenarien) wie das System zu funktionieren hat
- Szenarien werden möglichst natürlichsprachlich formuliert. Werden auch als „Living Documentation“ bezeichnet
- Entwicklung einer gemeinsamen Sprache zwischen Beteiligten



# Behaviour-Driven Development (BDD)

- Szenario: Formulierung in eigener Sprache
- Glue Code: Übersetzung des Szenarios in Aufrufe an das zu testende System
- System under Test: System bzw. Funktionalität, die getestet werden soll

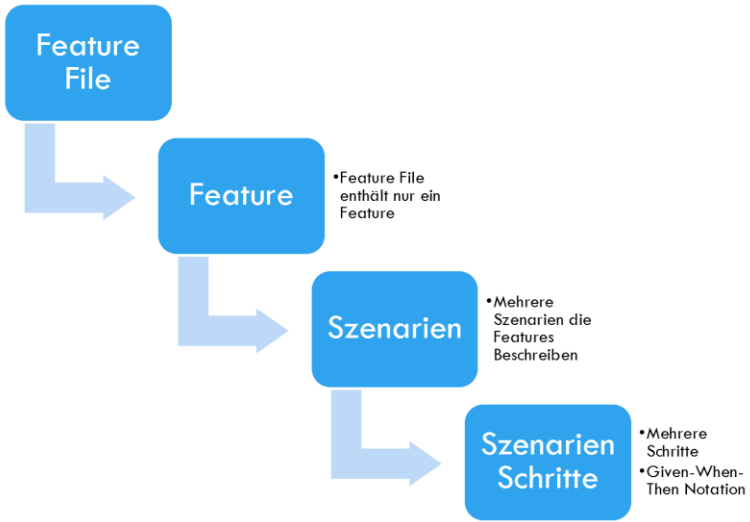


# Cucumber


- BDD-Framework unter MIT-Lizenz
  - Website: https://cucumber.io/
  - Git-Repository: https://github.com/cucumber/
- Implementierungen in verschiedenen Sprachen
  - intern: Ruby, JVM (Java, Kotlin), JavaScript, Go
  - Externe: C#, Behave, Rust, PHP
- Cucumber-JVM: Anbindung an Test-Frameworks (JUnit, TestNG) & DI-Frameworks (z.B. Spring, CDI)
- Definiert und verwendet Gherkin als Beschreibungssprache

# Gherkin

- Strukturierte Beschreibungssprache für Szenarien
- Entwickelt im Rahmen von Cucumber; kann und wird von anderen BDD-Frameworks verwendet
- Bestehend aus 11 Keywords




Quelle: https://www.testautomatisierung.org/lexikon/gherkin/

# Gherkin - Beispiel

- Den Kern bildet das Given/When/Then-Triplet

```
Feature: Authors can create article and submit them for review

  Scenario: An Author can submit an article to editors
    Given user Mike is logged in
    And has the role author
    And user Bob has the role editor
    When he submits an article
    Then user Bob sees the article in his review list
```

# Gherkin - Lokalisierung

- Lokalisierung für über 70 Sprachen
- Sprache wird am Beginn der Feature-Dateien in einem Kommentar angegeben werden (z.B. # language: de)
- Verwendung, wenn andere Sprache bevorzugt wird (z.B. oftmals im Behördenumfeld)

| englisch | deutsch |
|---|---|
| feature | Funktionalität, Funktion |
| scenario | Beispiel, Szenario |
| given | Angenommen, Gegeben sei, Gegeben seien |
| when | Wenn |
| then | Dann |
| and | Und |

# Cucumber – Glue Code 1 / 2

- Übersetzt die Ausdrücke der Szenarien in Aufrufe an das zu testende System
- Ausdrücke werden anhand von regulären Ausdrücken oder Cucumber Expressions ausgewertet
  - Eigene Syntax für variable Teile, die als Parameter an die StepDefinitions übergeben werden
- Cucumber-JVM:
  - StepDefinitions können in zwei Formen angegeben werden
    - Methoden mit Cucumber-Annotationen (z.B. @Given)
    - Aufruf von Cucumber-Methoden, denen Lambda-Expressions übergeben werden
  - Innerhalb der StepDefinitions können assert-Methoden von Test-Frameworks verwendet werden, um den richtigen Zustand sicherzustellen

# Cucumber – Glue Code 2 / 2

```java
@Given("user {word} has the role {word}")
public void user_has_the_role(String user, String role) {
    testHelper.ensureUserWithRoles(user, role);
}

@When("he submits an article")
public void he_submits_an_article() {
    testHelper.createArticle(testHelper.getCurrentUser());
}

@Then("user {word} sees the article in his review list")
public void editor_sees_the_article_in_his_review_list(String user) {
    User editor = userService.getUser(user);

    List<Article> articles = articleService.getArticlesForReview(editor);

    assertFalse(articles.isEmpty());
    assertTrue(articles.contains(testHelper.getArticle()));
}
```

# Acceptance Test-Driven Development (ATDD)

- Von Product Owner bzw. Analyst, Entwickler und Tester werden zu den User Stories Akzeptanztests definiert
- Tests werden möglichst natürlichsprachlich formuliert, so dass sie auch von Nicht-Technikern geschrieben werden können
- Akzeptanztests können, aber müssen nicht automatisiert werden
- Ähnlich zu BDD, aber höherer Abstraktionslevel

# Testautomatisierung

- Gemeint sind meist UI-Tests
- Auch oft unter dem Begriff e2e-Tests („end-to-end“)

- Herausforderungen
  - UI schwierig zu testen
  - Hohe Volatilität von UI-Technologien
  - Langsamere Ausführung im Vergleich zu anderen automatisierten Tests
  - Leicht zu brechen, z.B. Umbenennung eines Buttons
  - Cross-Browser-Kompatibilität

# Testautomatisierung – Gestern & Heute

## Gestern

- Defacto keine Unterstützung durch UI-Technologien
- Tests eher unzuverlässig => Viele False-Positives
- Hauptsächlich kommerzielle Produkte

## Heute

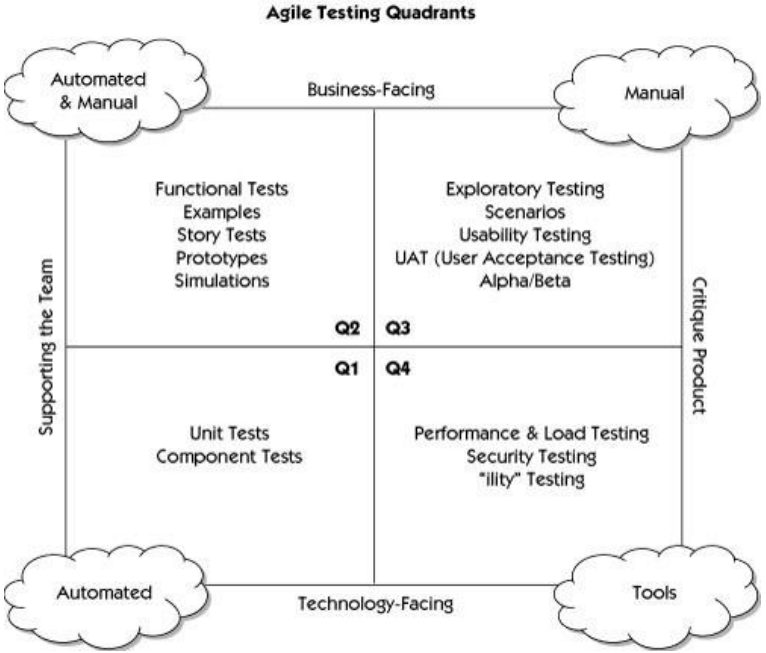
- (Bessere) Unterstützung durch UI-Technologien (z.B. W3C WebDriver)
- False Positives durch bessere Tools weniger, aber immer noch möglich
- Open Source-Frameworks (ggf. mit kommerziellem Support)

# Testautomatisierung – Best Practices

- Erfordert Abstimmung bzw. Aufstellen von Regeln zwischen Entwicklern und Testern (z.B. Definition eigener Test-Attribute in HTML-Elementen)
- Testen des Positiv-Falls
- Für Setup Abkürzungen verwenden, um Laufzeit zu verringern (z.B. direkter Aufruf von REST-Services statt durch UI)
- Auf sauberen Code und gute Strukturierung achten
- Ergebnis von Record-and-Play-Funktionalität nicht 1:1, sondern nur als Ausgangsbasis für Implementierung verwenden

# Agile Testquadranten nach Crispin & Gregory

- Einteilung der verschiedenen Test-Typen in unterschiedliche Quadranten
- Keine Priorisierung oder Reihung der einzelnen Quadranten => es werden alle Quadranten benötigt




| | |
|---|---|
| Q2 | Q3 |
| Q1 | Q4 |

Quelle: Lisa Crispin, Janet Gregory: Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley Professional, 2008

# Q1: Technische Tests, die das Team unterstützen

| Q2 | Q3 |
|---|---|
| Q1 | Q4 |

- Entwicklung von Unit-Tests z.B. mittels Test-Driven Development
- Regelmäßige Integration und Ausführen der Tests
- Zeigen die interne Qualität des Produkts
- Verwendete Werkzeuge
  - Versionskontrollsysteme (z.B. Git)
  - IDEs für Refactoring-Support
  - Build-Tools (z.B. Apache Maven, Gradle)
  - Continuous Integration Server (z.B. Jenkins, GitLab CI/CD, GitHub Actions)
  - Unit-Test-Frameworks (z.B. JUnit, TestNG)

# Q2: Geschäftsorientierte Tests, die das Team unterstützen

| Q2 | Q3 |
|---|---|
| Q1 | Q4 |

- Automatisierte Akzeptanztests, die auch von Nicht-Entwicklern verstanden werden können
- Höheres Abstraktionslevel als Tests von Q1, jedoch Überschneidung zwischen Tests von Q1 und Q2 möglich
- Zeigen die externe Qualität des Produkts und festzustellen, ob man "done" ist
- Unterstützen gemeinsame Sprache zwischen Domäne und Entwicklung zu finden
- Werkzeuge:
  - manuell: Checklisten, Mind Maps, Tabellen, Mock-Ups, Diagramme
  - automatisiert: BDD-Frameworks, Werkzeuge zur Testautomatisierung

# Q3: Geschäftsorientierte Tests, die das Produkt kritisieren

| Q2 | Q3 |
|---|---|
| Q1 | Q4 |

- Hauptsächlich manuelle Tests
- Test-Methoden:
  - Szenario-basierte Tests
  - Explorative Tests
  - Session-basierte Tests
  - Usability-Tests
- Testen von Schnittstellen (z.B. Web Services, RESTful-Services)

# Q4: Technische Tests, die das Produkt kritisieren

| Q2 | Q3 |
|---|---|
| Q1 | Q4 |

- Testen der nicht-funktionalen Anforderungen
- Benötigen oftmals Unterstützung durch Spezialisten
- *ilities-Tests
  - Security
  - Maintainability
  - Reliability
  - Installability
- Performance-Tests
- Last-Tests
  - Unterschiedliche Arten mit unterschiedlichen Zielen: Load-, Stress-, Spike-, Capacity-Testing
  - Aufwändiges Test-Setup
  - Benötigen ähnliche Ausstattung wie Produktionssystem

# Zusammenfassung

- Agiles Manifest bildet die Grundlage für agile Software-Entwicklung und agiles Testen
- Scrum & Kanban sind die vorherrschenden Vorgehensmodelle in der agilen Software-Entwicklung
- Tester sind Teil des Teams und unterstützen Projekt vom Beginn bis zum Ende
- Spezielle Testmethoden im agilen Bereich: TDD, BDD, ATDD
- UI-Tests werden besser unterstützt und haben an Bedeutung gewonnen
- Agile Testquadranten geben Hilfestellung welche Tests benötigt werden, um Qualität sicherstellen zu können

# Referenzen


- Thomas Grechenig, Mario Bernhart, Roland Breiteneder, Karin Kappel: Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten, Pearson Studium, 2009
- Ken Schwaber, Jeff Sutherland: The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game, 2020
- Kent Beck: Extreme Programming Explained: Embrace Change, 2nd Edition, Addison-Wesley Professional, 2004
- ISTQB Certified Tester: Foundation Level Extension Syllabus Agile Tester, dt. Ausgabe, 2017
- Lisa Crispin, Janet Gregory: Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley Professional, 2008
- Manfred Baumgartner, Martin Klonk, Christian Mastnak, Richard Seidl: Agile Testing - Der agile Weg zur Qualität, 3. Auflage, Hanser, 2024
- Matt Wynne, Aslak Hellesøy: The Cucumber Book - Behaviour-Driven Development for Testers and Developers, 2nd Edition, The Pragmatic Bookshelf, 2017