HTWK Hochschule für Technik, Wirtschaft und Kultur Leipzig

Requirements Engineering: User Stories und Epics in agilen Vorgehensmodellen

Modul "Software Engineering" (Prof. Dr. Andreas Both, Wintersemester 2024/2025) an der HTWK Leipzig

Jonas Pohl, Mose Schmiedel, Antonie Swiridoff

08.01.25

Zielsetzung

- **Wie werden Epics und User Sories in agilen Modellen eingesetzt?**
- **Welchen Effekt haben User Stories auf das agile Umfeld?**

Vorgehen:

- Grundlagen Requirements Engineering (Wdh.)
- Def. Epics und User Stories
- Praktische Anwendungsbeispiele
  - Kanban
  - Scrum
  - Feature Driven Developement

Wiederholung Requirements Engineering (RE)

RE als Teilbereich des Software Engineerings

Anforderungen *(requirements)*: Welche Eigenschaften werden vom Softwaresystem erwartet?

Requirements Engineering: Prozess des Herausfindens, Spezifizierens, Analysierens, Dokumentierens und Überprüfens der Anforderungen

Ziele: Klärung von Anforderungen, Sicherstellung von Qualität und Kundenzufriedenheit.

*„Die Anforderungen an ein neues Softwareprodukt zu ermitteln, zu spezifizieren, zu analysieren, zu validieren und daraus eine fachliche Lösung abzuleiten […], **gehört mit zu den anspruchsvollsten Aufgaben** innerhalb der Softwaretechnik.“*

(Balzert, Lehrbuch der Softwaretechnik)

*„Die Bedeutung des Requirements Engineering erkennt man daran, **wie stark der Erfolg** einer Softwareentwicklung **davon abhängt**.“*

(Balzert, Lehrbuch der Softwaretechnik)

Literatur: Sommerville: Software Engineering, 2016
Balzert: Lehrbuch der Softwaretechnik, 2009

Wiederholung Requirements Engineering (RE)

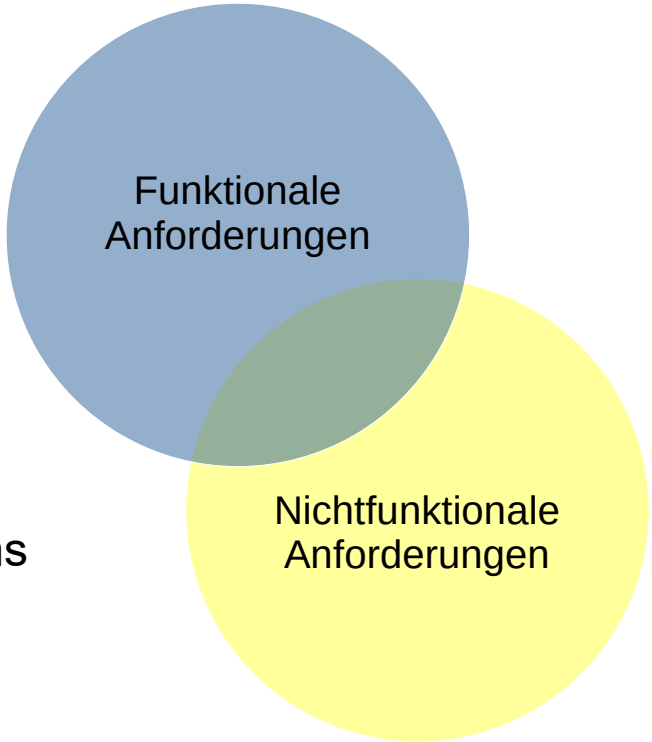
Arten von Anforderungen

Funktionale Anforderungen

- Was? Dienste und Verhalten des Softwaresystems.
- Kernpunkte:
  - Dienste / Funktionen, die bereitgestellt werden sollten
  - Reaktionen auf bestimmte Eingaben
  - Verhalten in bestimmten Situationen

Nichtfunktionale Anforderungen

- Was? Eigenschaften oder Beschränkungen des *gesamten* Softwaresystems
- Beispiele:
  - Zuverlässigkeit, Verfügbarkeit, Informationssicherheit
  - Antwortzeit
  - Standards für Schnittstellen oder Datenformate

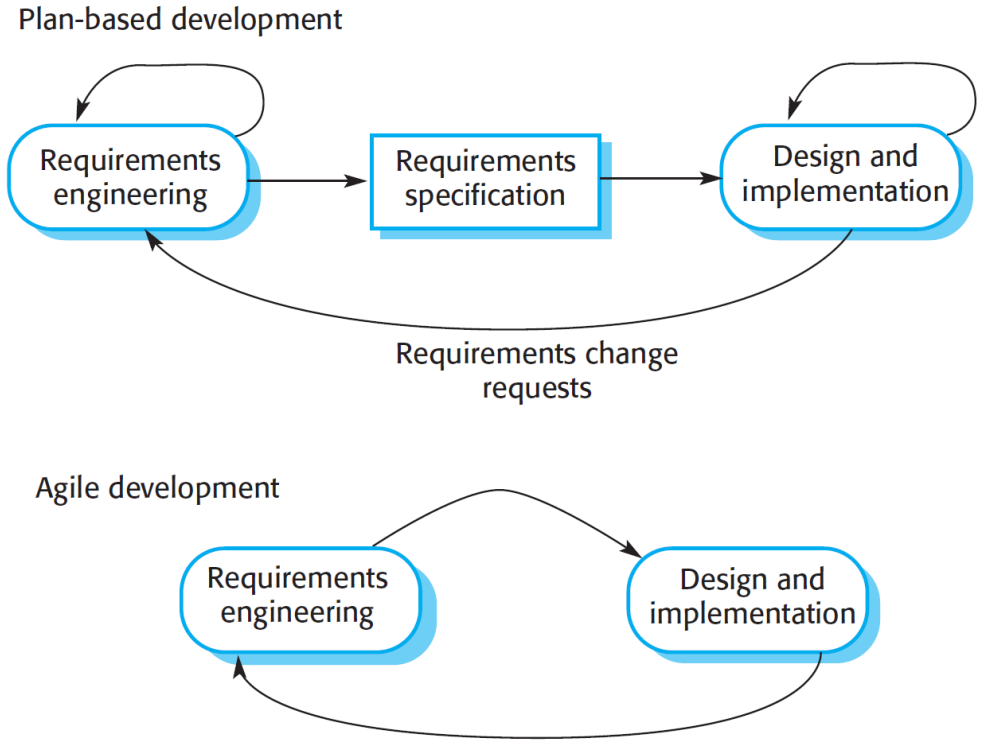


Wiederholung Requirements Engineering (RE)

Besonderheiten bei der agilen Entwicklung

Hier: Fokus auf agile Entwicklung, d.h. …

- keine abgeschlossene Phase des RE vor der Implementierung
- Iteration über alle Phasen hinweg (Spezifikation und Implementierung greifen unmittelbar ineinander)
- Anforderungen und Design werden gemeinsam entwickelt
- kontinuierliche Einbindung der Stakeholder in Spezifikation und Bewertung



(Sommerville: Software Engineering, 2016)

Epics und User Stories

Definition “Epic”

- Großes Arbeitspaket
  - großer Teamaufwand
  - lange Umsetzungsdauer
- Muss in User Stories verfeinert werden
- Gibt ein übergeordnetes Ziel/Vision vor

- Beispiel:

„As a *professor*, I want to *manage my course digitally* so that *students can autonomously register for them*.“

Literatur: Cohn, User Stories Applied: For Agile Software Development, 2004

Epics und User Stories

Definition “User Story”

- Kurze Beschreibung einer Funktionalität, welche für Nutzer der Anwendung wertvoll ist
- Aus Kundensicht, d.h. in Geschäftsterminologie des Kunden
- Card:

  “As a *<type of user>*, I want to *<goal>* so that *<achieved value>*.“

  “Als *<Typ des Nutzers>* möchte ich *<Ziel>,* so dass *<erreichter Wert>*.”\

- Conversation: Diskussion über Details
- Confirmation: Akzeptanzkriterien

- Beispiel:

  “As a *professor*, I want to *modify a course’s information* so that *it is valid*.”

Literatur: Balzert, Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering, 2008
Cohn, User Stories Applied: For Agile Software Development, 2004

Epics und User Stories

Was ist eine “gute” User Story?

- **INVEST**-Modell (nach B. Wake 2003)
  - **Independent** – unabhängig von anderen Anforderungen
  - **Negotiable** – Details beim Implementierungszeitpunkt absprechen
  - **Valuable** – Wert für Kunde soll ersichtlich sein
  - **Estimatable** – Implementierungsaufwand soll schätzbar sein
  - **Small** – Anforderung soll klein sein (Aufwand ca. 2-3 Mitarbeiterwochen)
  - **Testable** – Anforderungsformulierung soll Testbarkeit ermöglichen (klare Akzeptanzkriterien)

Literatur: B. Wake: https://xp123.com/invest-in-good-stories-and-smart-tasks/ (05.01.2025), 2003

Wann ist eine User Story bereit und abgeschlossen?

Definition of Ready (DoR):

Beispiele:
- Erfüllt INVEST
- Priorisiert und Akzeptanzkriterien festgelegt
- Aufwand geschätzt
- Technische Umsetzung geklärt

Definition of Done (DoD):

Beispiele:
- Akzeptanzkriterien erfüllt
- Fehlerfrei
- Dokumentiert
- Vom Product Owner abgenommen

➔ Werden von jedem Unternehmen selbst definiert

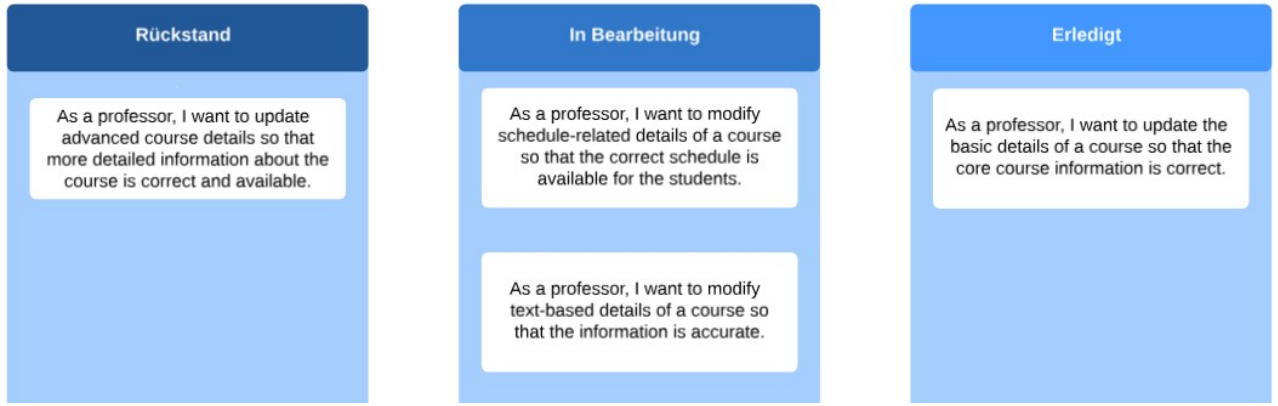
Kanban

Epics

- In Zusammenarbeit mit den Stakeholders erstellt

User Stories

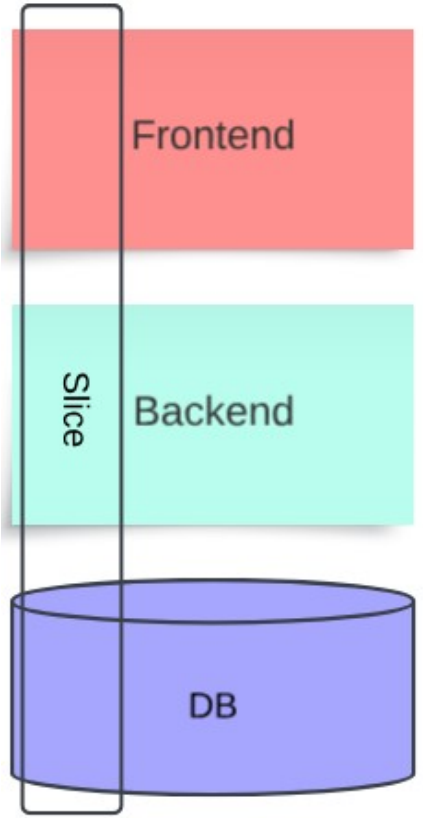
- Ganzes Team an Erstellen, Aufwandsschätzung, Priorisierung und Festlegen der Akzeptanzkriterien beteiligt
- Maximaler Umfang nicht festgelegt (Richtwert: Tage bis ein paar Wochen)



Kanban-Board mit User Stories

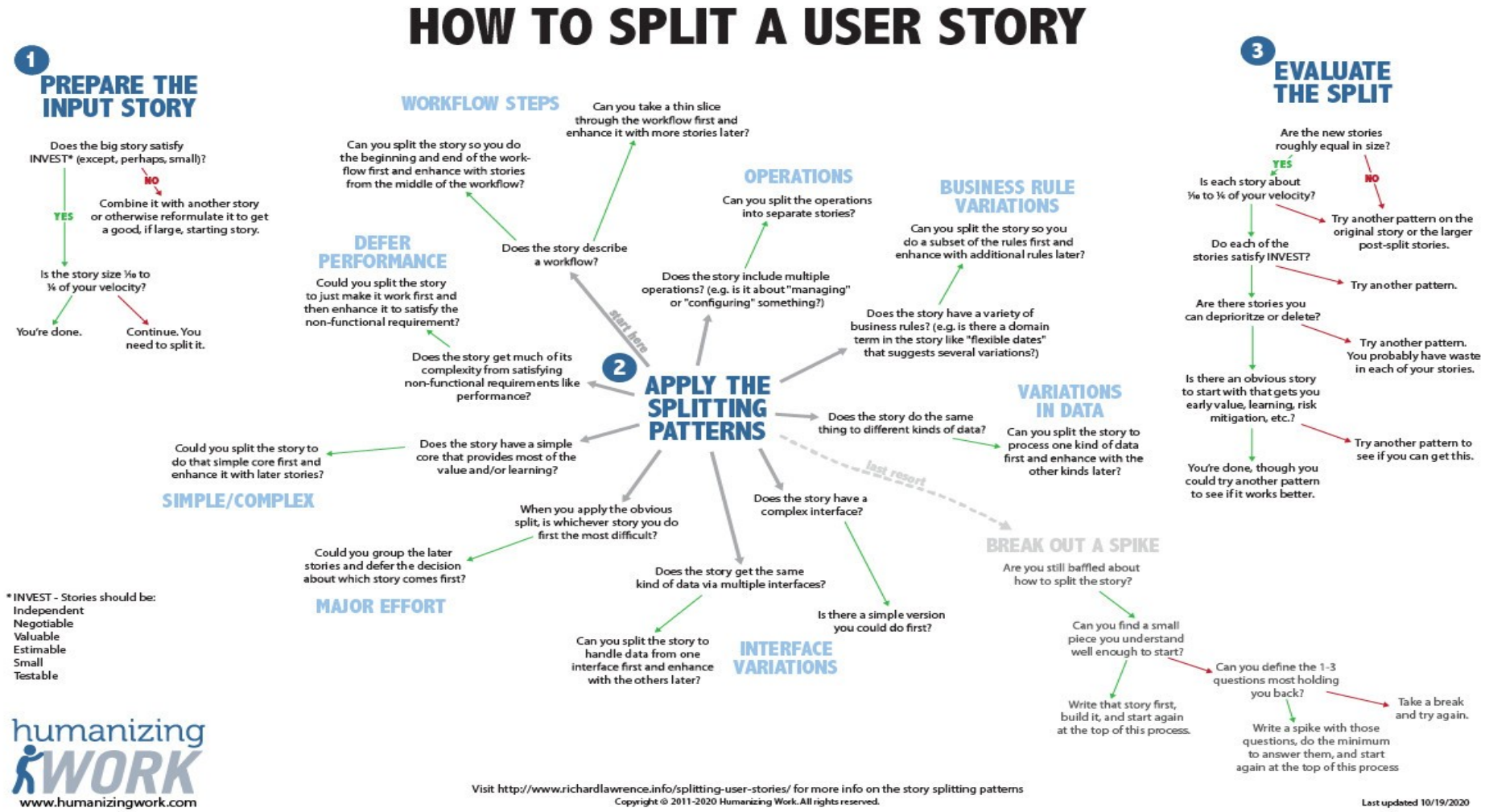
Kanban: Arbeiten mit User Stories

- Techn. Details Implementierung im Team selbstständig ausgearbeitet
- Vertikale Umsetzung wird stark empfohlen
- User Stories kontinuierlich im Team zugeteilt
- User Stories auf Kanban Board zugeordnet und nach Priorität sortiert
- User Stories können jederzeit entfernt, geändert oder weiter heruntergebrochen werden
- Synchronisierungspunkte mit dem Kunden werden vom Team dynamisch festgelegt



Vertikaler Slice eines Systems

User Stories herunterbrechen: Methode nach “humanizing Work”



[1]

User Stories herunterbrechen: Beispiel

Start: “As a professor, I want to modify a course's information so that it is valid.“

➔ Entspricht größtenteils INVEST, aber könnte kleiner sein

Nach Datenarten:

- „As a professor, I want to modify text-based details of a course so that the information is accurate.“
- „As a professor, I want to modify schedule-related details of a course so that the correct schedule is available for the students.“

Nach Komplexität:

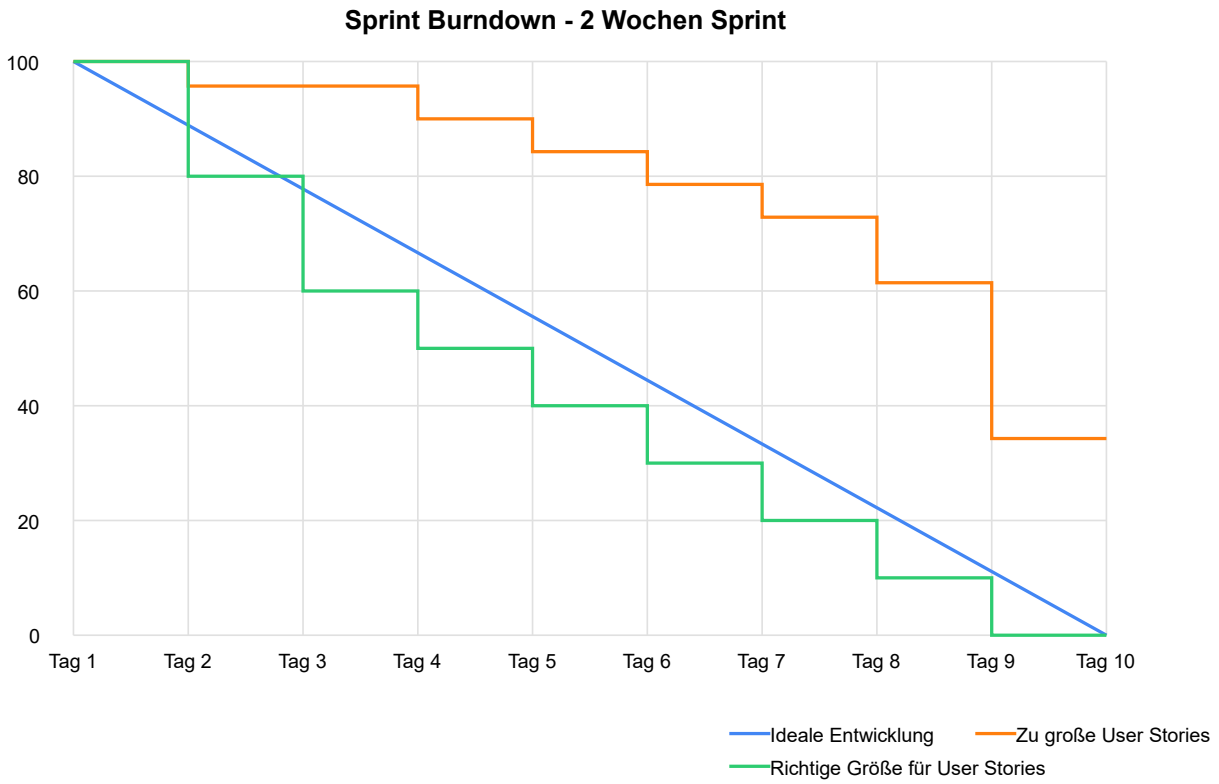
- „As a professor, I want to update the basic details of a course so that the core course information is correct.“
- „As a professor, I want to update advanced course details so that more detailed information about the course is correct and available.“

➔ Nächster Schritt: User Stories auf Größe, INVEST und ähnlichen Umfang untersuchen.

Scrum

Epics
- Zu groß für einen einzigen Sprint
- Sollten mit dem Stakeholder erarbeitet werden

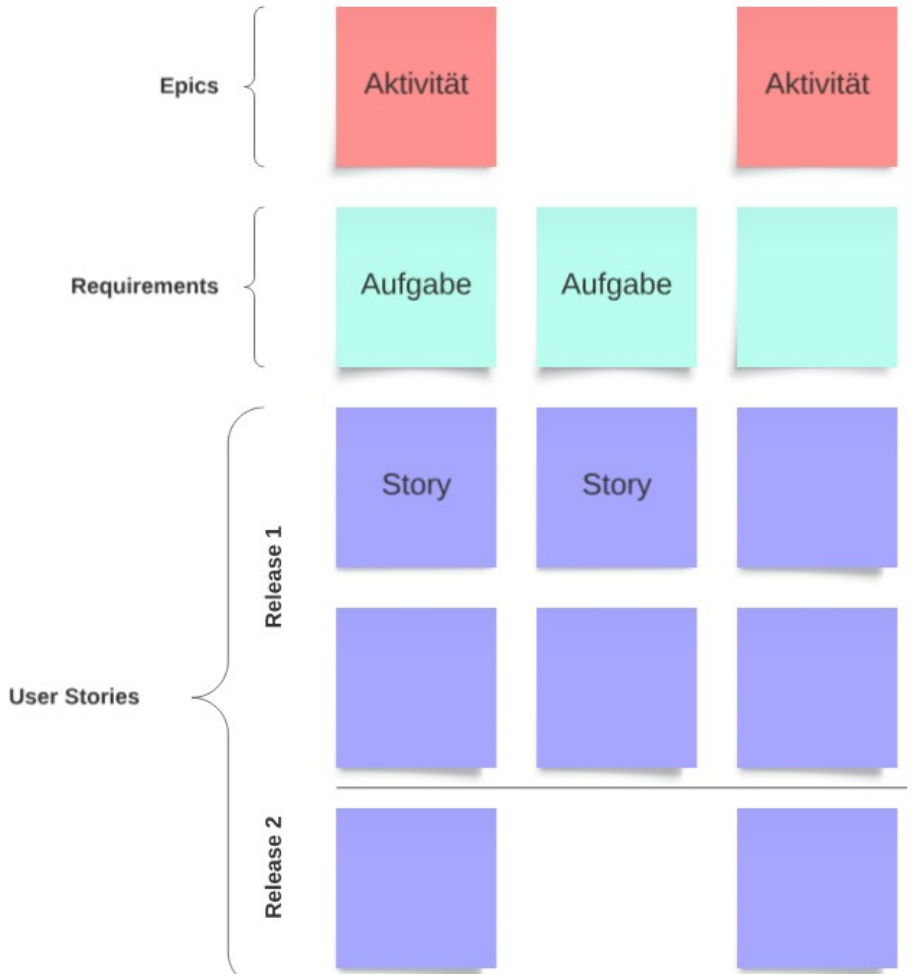
User Stories
- Sind Ziele für einen Sprint
- Umfang einer User Story sollte nach „The Humanizing Work Guide to Splitting User Stories“ zw. 1/6 und 1/10 der Velocity liegen



Burndownchart für einen Sprint

Scrum: Arbeiten mit User Stories

- Sprint Backlog soll ein lieferbares Produkt bilden
- Details der Implementierung wird im Sprint Planning erarbeitet
- User Stories müssen geschätzt und priorisiert werden
- Fortschritt in z.B. Burndownchart und Schwierigkeiten in daily Scrums erfasst
- Am Ende Sprint kann Stakeholder anhand der User Stories beurteilen, ob Ergebnis seinen Requirements entspricht

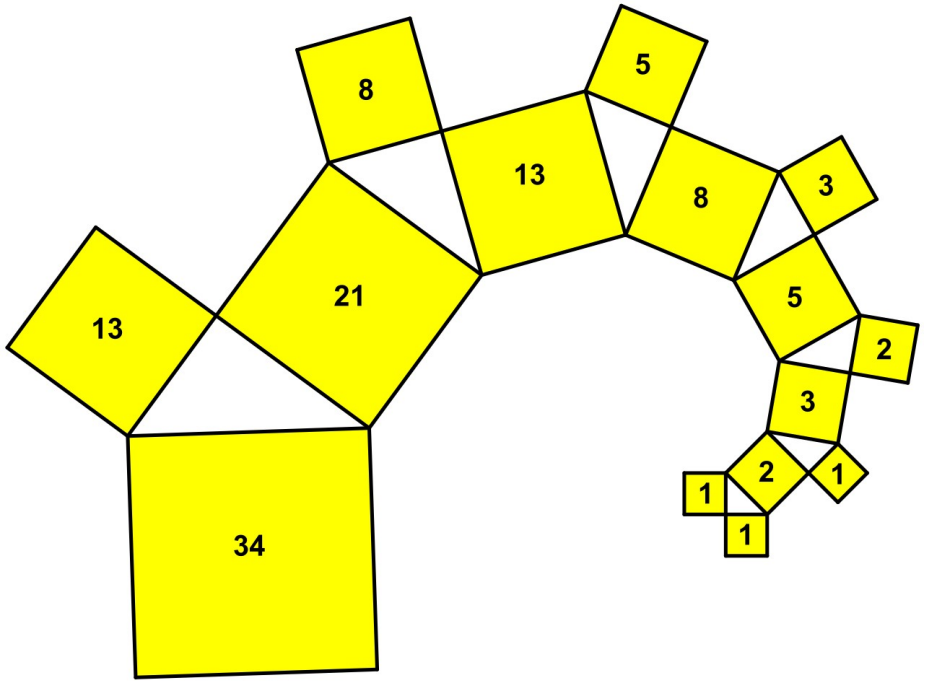


Bündeln von User Stories zu fertigen Produkten

Aufwandsschätzung von User Stories: Beispielmethode Planning Poker mit Fibonacci-Zahlen

1) Präsentation der User Story im Team
2) Diskussion zur Entwicklung gemeinsamen Verständnis
3) Individuelle Schätzung des Aufwands über die Fibonacci-Folge
4) Präsentation der Schätzungen
5) Diskussion über größte abweichende Schätzungen
6) Wiederholung ab 2) bei keiner Einigung
7) Finalisierte Schätzung (z.B. Median oder Mittelwert)

➔ Fibonacci-Folge für die Schätzwert, da so unrealistisch genaue Schätzung großer User Stories unterdrückt



Fibonacci-Folge [2]

Priorisieren von User Stories: Beispielmethode MoSCoW

[3]

Must Have

- Nicht verhandelbare
- für Produkt und Stakeholder unverzichtbar

Should Have

- Wichtig, aber nicht essenziell
- Oft noch nicht in erster Version

Could Have

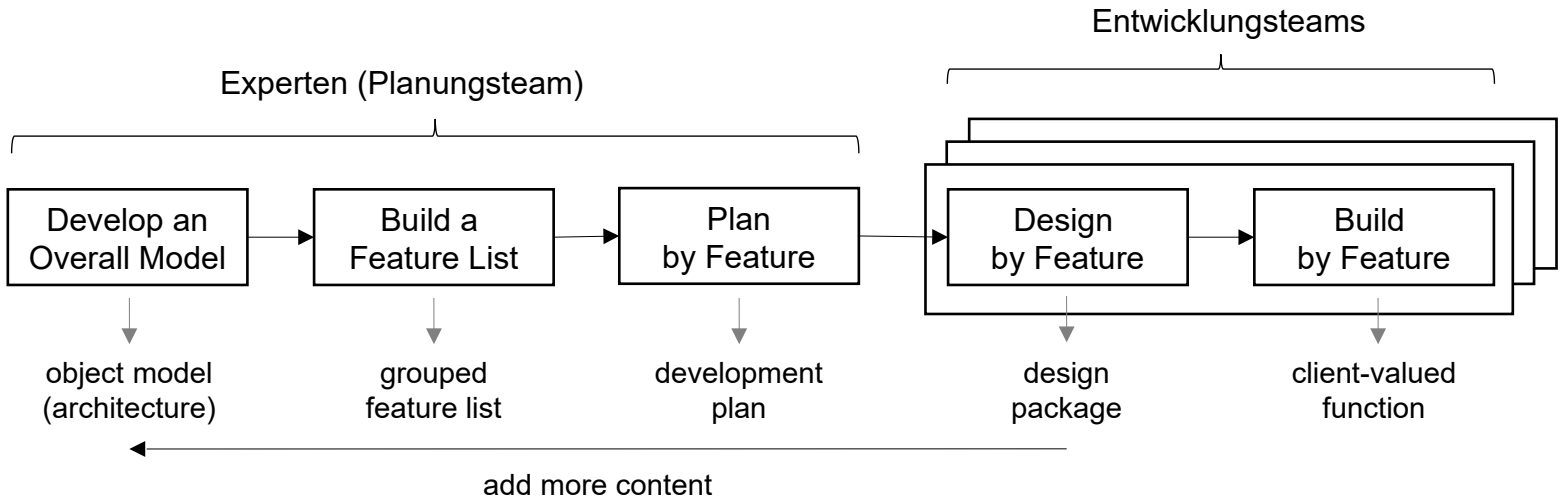
- Hilfreich und „Nice to have“
- Geringerer Nutzwert für Stakeholder

Will Not Have

- Nicht im Scope
- Vielleicht in zukünftigen Versionen aufgenommen

Feature-driven Development (FDD)
Wiederholung

- Feature ≙ „small client-valued functionality” (Entwicklungszeit: max. 2 Wochen)
  - Form: <Aktion> <Ergebnis> <Objekt>
  - Bsp.: „**Ändere** die **Informationen** des **Kurses**.“



- Hierarchie: Projektleiter → Experten (Planungsteam) → Entwicklungsteams
- geeignet für große, erfahrene Teams (es muss Experten geben)

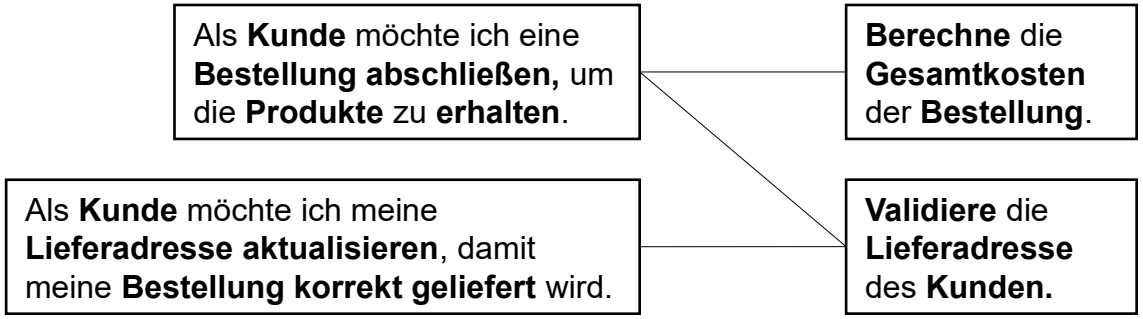
Literatur: Palmer, Felsing: A Practical Guide to Feature-Driven Development, 2002
Coad, De Luca, Lefebvre: Java Modeling in Color with UML, 1999

Feature-driven Development (FDD)

Feature 🕒 User Story

- Gemeinsamkeiten
  - Zerlegung von abstrakten Anforderungen in überschaubare Teilprobleme
  - Anforderung, die einen Wert für den Kunden / Anwender hat
- Unterschiede
  - Form, Abstraktionsebene, Entwicklungszeit
  - User Story: Objekte / Klassen werden aus User Stories identifiziert
  - Features: Features werden aus der Architektur- / dem Ablaufmodell abgeleitet

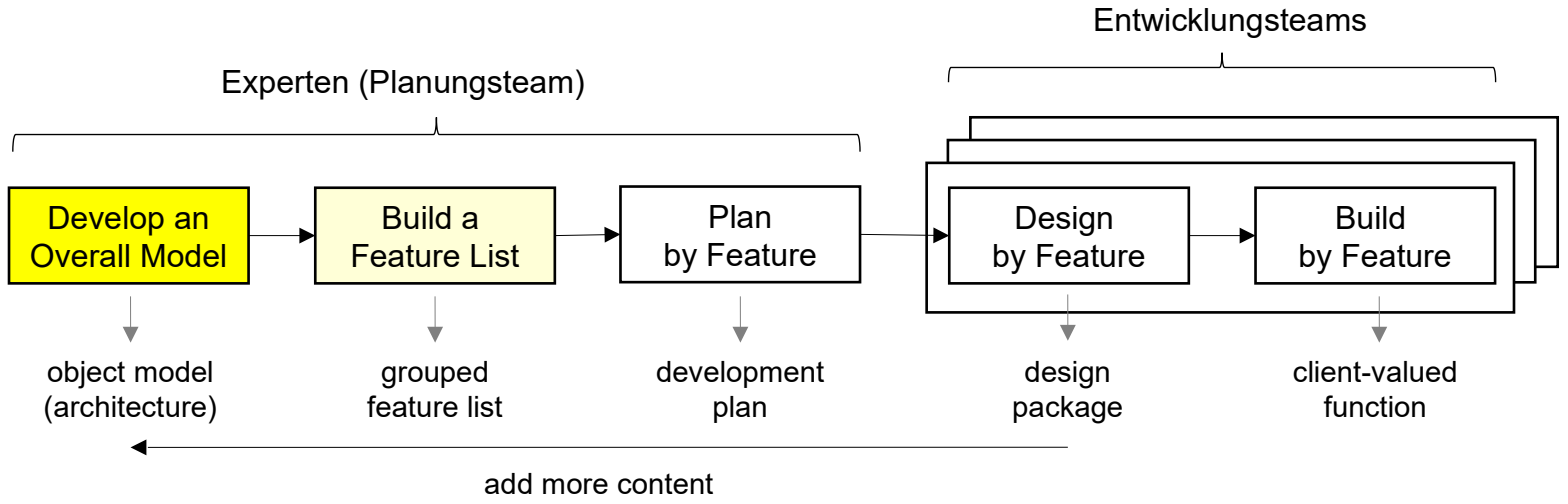
Eine User Story kann ein oder mehrere Features umfassen, gleichzeitig kann ein Feature in einer oder mehreren User Stories vorkommen:



Literatur: Palmer, Felsing: A Practical Guide to Feature-Driven Development, 2002
Hunt: Agile Software Construction, 2005

Feature-driven Development (FDD)
Verwendung von Epics und User Stories

- Verwendung hauptsächlich in Phase 1 (Modellierung)
- können außerdem dabei helfen, Features zu identifizieren



- dienen als zusätzliches Kommunikationsmittel (neben Features) zwischen
  - Planungsteam und Entwicklungsteams
  - Planungsteam und Kunden (stärker aus der Sicht des Kunden als Features)

Literatur: Palmer, Felsing: A Practical Guide to Feature-Driven Development, 2002
Hunt: Agile Software Construction, 2005

Feature-driven Development (FDD)
Zusammenfassung

- FDD baut nicht grundlegend auf Epics und User Stories auf
- Epics und User Stories werden im Allgemeinen nicht zur Priorisierung und Planung eingesetzt (diese Rolle übernehmen hier Features)
- Verwendung in erster Linie für die Modellierung und Kommunikation
- Kommunikation zw. Planungsteam und Entwicklungsteams bei FDD essenziell und kritisch, Epics und User Stories können hier helfen
- Beachte: User Stories dürfen nicht mit Features verwechselt werden!

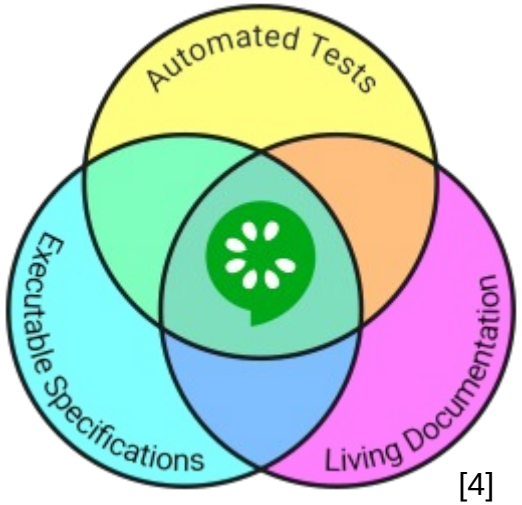
Insgesamt: User Stories und Epics sind eine nützliche Ergänzung für FDD.

Literatur: Palmer, Felsing: A Practical Guide to Feature-Driven Development, 2002
Hunt: Agile Software Construction, 2005

Automatische Verarbeitung von User Stories

Cucumber

- Testsystem um definierte User Stories (Features) automatisiert zu validieren
- Entwickelt für Behaviour-Driven-Development (BDD)
- Gherkin-Grammatik
  - Spezifikation
  - Automatisierte Tests
  - Dokumentation des tatsächlichen Systemverhaltens



[4]

Automatische Verarbeitung von User Stories

Cucumber – Beispiel

```
Feature: As a professor, I want to modify a course.

Scenario: As a professor, I want to modify a course's information so that it is valid.
  Given a course with name "Software Enginearing"
  When  the professor modifies the name of the course to "Software Engineering"
  Then  the name of the course is "Software Engineering"
```

Automatische Verarbeitung von User Stories

Cucumber – Beispiel

```java
public class StepDefinitions {

    private Course course;

    @Given("a course with name {string}")
    public void a_course_with_name(String courseName) {
        course = new Course(courseName);
    }

    @When("the professor modifies the name of the course to {string}")
    public void the_professor_modifies_the_name_of_the_course_to(
        String newName
    ) {
        course.setName(newName);
    }

    @Then("the name of the course is {string}")
    public void the_name_of_the_course_is(String name) {
        assertEquals(name, course.getName());
    }
}
```

Automatische Verarbeitung von User Stories

Cucumber – Beispiel

```
Feature: As a professor, I want to modify a course.

Scenario: As a professor, I want to modify a course's information so that it is valid.
  Given a course with name "Software Enginearing"
  When  the professor modifies the name of the course to "Software Engineering"
  Then  the name of the course is "Software Engineering"
```

```
1  private Course course;
2
3  @Given("a course with name {string}")
4  public void a_course_with_name(String courseName) {
5      course = new Course(courseName);
6  }
```

Automatische Verarbeitung von User Stories

Cucumber – Beispiel

```
Feature: As a professor, I want to modify a course.

Scenario: As a professor, I want to modify a course's information so that it is valid.
  Given a course with name "Software Enginearing"
  When  the professor modifies the name of the course to "Software Engineering"
  Then  the name of the course is "Software Engineering"
```

```
1  @When("the professor modifies the name of the course to {string}")
2  public void the_professor_modifies_the_name_of_the_course_to(
3      String newName
4  ) {
5      course.setName(newName);
6  }
```

Automatische Verarbeitung von User Stories

Cucumber – Beispiel

```
Feature: As a professor, I want to modify a course.

Scenario: As a professor, I want to modify a course's information so that it is valid.
  Given a course with name "Software Enginearing"
  When  the professor modifies the name of the course to "Software Engineering"
  Then  the name of the course is "Software Engineering"
```

```
1  @Then("the name of the course is {string}")
2  public void the_name_of_the_course_is(String name) {
3      assertEquals(name, course.getName());
4  }
```

Diskussion

Effekt User Stories:

- sorgen für klare Kommunikation mit dem Kunden
- Ermöglichen Strukturieren und Priorisieren der Projekt-Requirements
- Verständnis der Perspektive des Stakeholders
- Kleine Arbeitspakete für die Umsetzung in agilen Modellen

Beachte:

- Nichtfunktionale Anforderungen sind schwierig als User Story abzubilden
- Gefahr der Überspezialisierung auf einen spezifischen Stakeholder (oft Kunden)
- Kompatibilität der einzelnen User Stories untereinander

Bei Beachtung der INVEST-Richtlinien tragen User Stories maßgeblich zum Erfolg von agilen Projekten bei

Zusammenfassung

- Definition und Qualitätsmerkmale von Epics und User Stories
  - INVEST-Modell
  - Akzeptanzkriterien
  - DoR und DoD
- Anwendung von User Stories in verschiedenen agilen Modellen und damit einhergehenden Herausforderungen
  - Kanban
  - Scrum
  - FDD
- Einige Anwendungsmethoden und Tools für den richtigen Umgang mit User Stories
  - „How to Split a User Story“: humanizing Work
  - Planning Poker mit Fibonacci-Zahlen
  - MoSCoW-Methode
  - Cucumber

Literatur

– I. Sommerville: Software Engineering, Global Edition, 10. Auflage, Pearson, 2016
– H. Balzert: Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering, 3. Auflage, Spektrum Akademischer Verlag, 2009
– S. Palmer, J. Felsing: A Practical Guide to Feature-Driven Development, Prentice Hall, 2002
– P. Coad, J. De Luca, E. Lefebvre: Java Modeling in Color with UML, Pearson, 1999
– J. Hunt: Agile Software Construction, Springer, 2005
– M. Cohn, User Stories Applied: For Agile Software Development, 1.Auflage, Addison Wesley Longman Publishing Co., Inc., 2004
– B. Wake: https://xp123.com/invest-in-good-stories-and-smart-tasks/ (05.01.2025), 2003

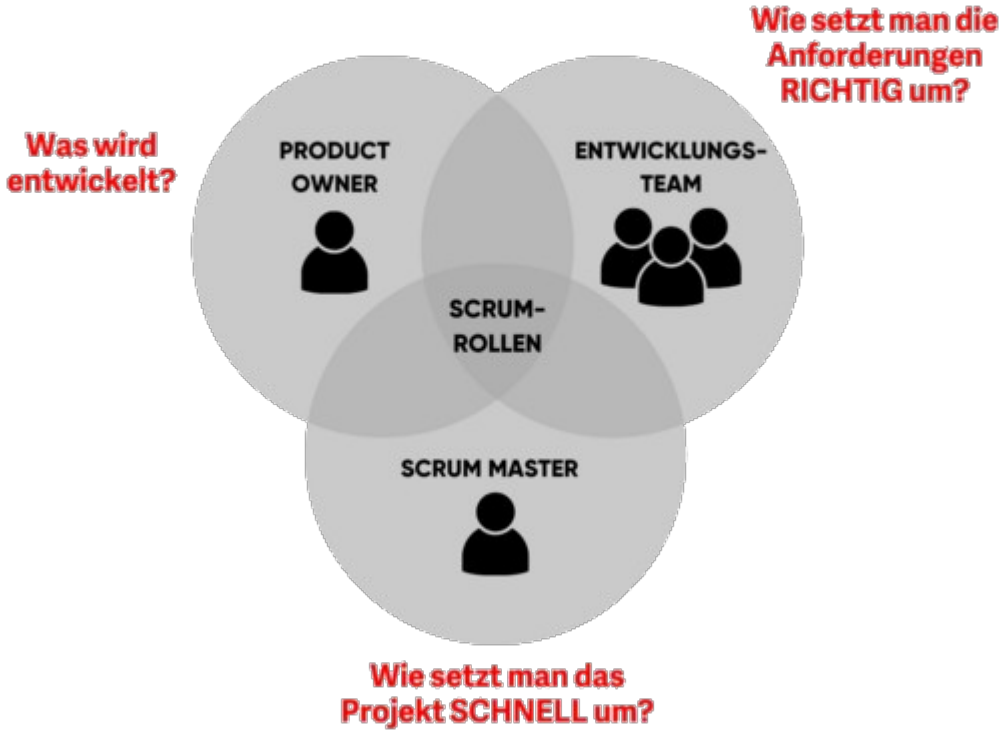
Bildquellen

- [1] Agile Clinic: Wer stellt ein Scrum-Team zusammen?, Agile Clinic, veröffentlicht am 17. September 2023, [online], verfügbar unter: https://www.agile-clinic.de/news/detail/wer-stellt-ein-scrum-team-zusammen, zuletzt abgerufen am 7. Januar 2025
- [2] Wikipedia: Fibonacci-Folge, Wikipedia, [online], verfügbar unter: https://de.wikipedia.org/wiki/Fibonacci-Folge, zuletzt abgerufen am 7. Januar 2025
- [3] J. Lawless: The MoSCoW Technique, Purple Griffon, zuletzt bearbeitet am 7. August 2024, [online], verfügbar unter: https://purplegriffon.com/blog/moscow-technique, zuletzt abgerufen am 7. Januar 2025
- [4] The Cucumber Open Source Project, [online], verfügbar unter: https://cucumber.io/assets/images/single-source-of-truth-7d8888ca3bbeea3a4554766064b39a6f.png, zuletzt abgerufen am 7. Januar 2025
- [5] R. Lawrence, P. Green: The Humanizing Work Guide to Splitting User Stories, Humanizing Work, [online], verfügbar unter: https://www.humanizingwork.com/the-humanizing-work-guide-to-splitting-user-stories/, zuletzt abgerufen am 7. Januar 2025

Scrum

Rollen im RE

- Product Owner:
  - Verwaltet Product Backlog: ordnet nach Priorität
  - Ansprechpartner für den Stakeholder
  - Kommuniziert Anforderungen zum Team
  - Formuliert Epics und User Stories (evtl. mit Team)
- Team:
  - Erarbeiten im Sprint Planing den Sprint Backlog
  - Klärt die Aufteilung und Implementierung der User Stories im Sprint Backlog



Rollen in Scrum [5]