Requirements Engineering: User Stories und Epics in Vorgehensmodellen

JONAS POHL*, MOSE SCHMIEDEL*, and ANTONIA SWIRIDOFF*, Hochschule für Technik, Wirtschaft und Kultur Leipzig (HTWK Leipzig), Deutschland

In dieser Arbeit wurde der Einsatz und die Wirkung von Epics und User Stories in der agilen Softwareentwicklung untersucht. Dazu wurden mithilfe bestehender Literatur drei agile Vorgehensmodelle und die beim Einsatz von Epics und User Stories auftretenden Herausforderungen betrachtet. Für die Herausforderungen wurden jeweils mögliche Lösungsvorschläge gefunden, welche anhand von kurzen Beispielen in dieser Arbeit beschrieben wurden.

Im Verlauf der Recherche sind die Autoren zu dem Schluss gekommen, dass Epics und User Stories ein hilfreiches Werkzeug für das Requirements Engineering sind. Insbesondere für die agile Softwareentwicklung ist dieses Konzept besser geeignet als traditionelle Methoden.

1 GRUNDLAGEN REQUIREMENTS ENGINEERING

1.1 Definition und Einordnung

Die *Anforderungen* (engl. *requirements*) an ein Softwaresystem beschreiben, welche Eigenschaften von der Software erwartet werden [Sommerville 2016]. Dabei kann es sich um Benutzeranforderungen oder auch Systemanforderungen handeln [Sommerville 2016].

Der Prozess, um solche Anforderungen "zu ermitteln, zu spezifizieren, zu analysieren, zu validieren und daraus eine fachliche Lösung abzuleiten", wird als *Requirements Engineering* (*RE*) bezeichnet [Balzert 2009]. Das Ziel dieses Prozesses ist neben der Erarbeitung einer Anforderungsspezifikation die Qualität der Software und die Kundenzufriedenheit sicherzustellen.

Das Requirements Engineering gilt als besonders kritisch, da es einerseits eine der komplexesten Teilbereiche des Software Engineerings darstellt und andererseits maßgeblich den Erfolg eines Softwareprojekts beeinflusst [Balzert 2009]. Somit gehört das Requirements Engineering zu den größten Herausforderungen in der Softwareentwicklung.

Eine intensive Einbeziehung aller Personen oder Institutionen, die ein Interesse an der Software haben, direkt beteiligt oder von deren Nutzung betroffen sind, ist unerlässlich. Diese Personen oder Institutionen werden *Stakeholder* genannt [Balzert 2009].

1.2 Funktionale und nichtfunktionale Anforderungen

Häufig wird zwischen zwei Arten von Anforderungen unterschieden: Funktionale und nichtfunktionale Anforderungen. In [Sommerville 2016] werden diese folgendermaßen definiert:

Funktionale Anforderungen legen fest, welche Dienste und Funktionen die Software bereitstellen soll. Sie beziehen sich häufig auf konkrete Nutzeranforderungen, wie die Reaktion auf bestimmte Eingaben oder das gewünschte Verhalten in einer konkreten Situation. Ein einfaches Beispiel ist:

> Der Benutzer muss sich mit seiner E-Mail-Adresse und einem Passwort anmelden können.

Nichtfunktionale Anforderungen beziehen sich hingegen meist auf übergreifende Eigenschaften des Softwaresystems oder auf mehrere seiner Funktionen. Dazu zählen Aspekte wie Zuverlässigkeit, Verfügbarkeit, Informationssicherheit und Performance. Besonders bei Echtzeitsystemen spielt bspw. die Antwortzeit eine kritisch Rolle. Darüber hinaus können nichtfunktionale Anforderungen auch Einschränkungen umfassen, z.B. dass für eine Schnittstelle ein bestimmtes Datenformat eingehalten werden muss.

Da sich Anforderungen häufig überschneiden, gegenseitig beeinflussen oder voneinander abgeleitet werden, können nicht alle Anforderungen eindeutig eine der beiden Kategorien zugeordnet werden. Die Anforderung "Der Benutzer muss sich mit seiner E-Mail-Adresse und einem Passwort anmelden können." beschreibt eine konkrete Funktion (Benutzeranmeldung), kann dabei aber zusätzlich nichtfunktionale Anforderungen beinhalten, wie z.B. eine kurze Antwortzeit oder hohe Sicherheitsstandards für die Verarbeitung der Anmeldedaten.

1.3 Requirements Engineering in der agilen Entwicklung

Bei der agile Softwareentwicklung, auf die sich dieser Artikel vorrangig konzentriert, gibt es wesentliche Unterschiede im Vergleich zur traditionellen bzw. plangesteuerten Softwareentwicklung.

Um schneller funktionsfähige Software bereitzustellen und flexibel auf Änderungen der Anforderungen reagieren zu können, wird in der agilen Entwicklung in der Regel auf eine detaillierte, vollständige Anforderungsspezifikation verzichtet. Stattdessen wechseln sich die Phasen des Requirements Engineerings und der Implementierung iterativ ab, sodass Spezifikation und Implementierung direkt ineinander greifen.

Das Requirements Engineering erfolgt daher entwicklungsbegleitend, wobei die Stakeholder fortlaufend einbezogen werden. Auf diese Weise wird eine kontinuierliche Überprüfung und Anpassung der Anforderungen sichergestellt. Dadurch können Änderungen schnell berücksichtigt und potenzielle Missverständnisse frühzeitig erkannt und mit geringerem Aufwand behoben werden.

2 EPICS UND USER STORIES

Wie dem *Agile Manifesto* [Beedle et al. 2001] festgehalten, muss eine agile Softwareentwicklung auf sich verändernde Anforderungen reagieren können. Dies führt dazu, dass viele herkömmliche Methoden zur Anforderungsanalyse für diese Art von Softwareentwicklung nicht mehr geeignet sind.

*Alle Studierenden trugen zu gleichen Teilen zu dieser Arbeit bei.

Diese Arbeit wurde im Rahmen des Mastermoduls „Software Engineering“ (Dozent: Prof. Dr. Andreas Both) an der HTWK Leipzig im Wintersemester 2024/2025 erstellt.

Um mit diesen Veränderungen im Requirements Engineering umzugehen, sind neue Werkzeuge und Konzepte für diese Phase der Softwareentwicklung entstanden, welche die Prinzipien der agilen Softwareentwicklung berücksichtigen und unterstützen.

Das Konzept von *Epics und User Stories* gehört zu diesen neuen Methoden und soll das Entwicklerteam während bei der Anforderungsanalyse unterstützen. Im folgenden Abschnitt werden die zugrundeliegenden Definitionen dieses Konzeptes vorgestellt, anhand eines Beispiels verdeutlicht und die Vor- und Nachteile beleuchtet.

2.1 Definition "User Story"

[Cohn 2004, p. 4] beschreibt "[e]ine *User Story* [als] eine Funktionalität, welche wertvoll für einen Nutzer [...] eines Systems oder Software ist". Diese Funktionalität wird dabei mit ein bis zwei Sätzen in folgender Struktur formuliert:

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

[Balzert 2009, p. 499]

`<Type of user>`, `<goal>` und `<achieved value>` stellen hierbei Platzhalter dar, welche je nach Anforderung ausgefüllt werden müssen. [Jeffries 2001] unterteilt eine User Story in folgende drei Teile:

Card repräsentiert die Anforderung, strukturiert nach oben genannter Vorlage.

Conversation findet als (verbale) Kommunikation der Anforderung zwischen Kunde zu Entwickler statt.

Confirmation besteht aus den Akzeptanztests, welche die notwendigen Eigenschaften der Anforderung festlegen.

In vielen Fällen begegnet ein Entwickler der User Story durch die Card. Diese wird aus Sicht des Kunden formuliert oder sogar von diesem verfasst [Balzert 2009, p. 497].

Auf der Card wird in wenigen Sätzen die in der User Story transportierte Anforderung zusammengefasst. Dabei sei hervorgehoben, dass die Card nicht etwa als vollständige Spezifikation für die geforderte Funktionalität dient, sondern lediglich als Erinnerung und Grundlage für spätere Diskussion. Diese wird in der Conversation durchgeführt und gegebenenfalls dokumentiert. Ziel der Conversation ist es dabei die Details der Funktionalität zu klären und durch Akzeptanztests in der Confirmation als Spezifikation für die Implementierung der Funktionalität festzuhalten [Cohn 2004, p. 4].

2.2 Definition "Epic"

Ein *Epic* stellt eine Vision oder ein übergeordnetes Ziel dar. Es gleicht in der Formulierung der User Story, umfasst aber einen viel größeren Umfang [Cohn 2004, pp. 6, 14]. Das Epic repräsentiert nämlich nicht nur eine einzelne Anforderung, sondern einen Anforderungsbereich. Somit bietet es dem Kunden und den Entwicklern eine Orientierungshilfe beim Formulieren der User Stories, ist aber selber unbrauchbar für die direkte Implementierung von Software-Features.

Um den Zweck einer Vision zu erfüllen, werden Epics dementsprechend grob und allgemein formuliert. Sie lassen einen weiten Interprationsspielraum offen, welcher durch weitere Absprachen zwischen Kunde und Team, bzw. innerhalb des Entwicklerteams ausgefüllt werden müssen.

Die Vorlage für die Formulierung von User Stories kann selbstverständlich auch für Epics verwendet werden. Dies bietet den Vorteil, dass schon die Vision aus der Perspektive des Kunden, bzw. des Stakeholders formuliert wird. Dies vereinfacht gegebenenfalls das spätere Formulieren von User Stories.

2.3 Beispiel

Um das Konzept der *Epics und User Stories* besser zu verdeutlichen, wurden alle Beispiele in diesem Artikel für ein imaginäres Kursverwaltungssystem erstellt. Dieses System soll Mitgliedern einer Hochschule die digitalen Verwaltung von Kursen und deren zugehörigen Informationen und Prüfungen ermöglichen. Außerdem sollen Studierende in der Lage sein, sich zu Kursen, bzw. Prüfungen an- und abzumelden.

Für dieses System wird in diesem Abschnitt am Beispiel eines spezifischen Epics der Umgang mit Epics und User Stories erläutert.

Das Epic ist basierend auf der Vorlage aus dem vorhergehenden Abschnitt formuliert. Dabei sind die Phrasen, welche für die Platzhalter eingesetzt wurden, unterstrichen.

Das gewählte Epic lautet wie folgt:

```
Als Professor möchte ich meine Kurse digital
    verwalten, damit die Studierenden
        unabhängig mit diesen interagieren können.
```

Wie schon in der Definition des Epics angesprochen handelt es sich bei einem Epic um ein umfangreiches Arbeitspaket, welches mehr eine Vision oder größeres Ziel ausdrückt, als eine einzelne Anforderung. Im Beispiel ist dies unschwer durch den Gebrauch von groben Formulierungen zu erkennen. So wird zum Beispiel lediglich festgelegt, dass ein `Professor` seine `Kurse digital verwalten` möchte. Diese Formulierung lässt einen großen Spielraum für die spätere Implementierung und erzwingt gleichzeitig auch die Eingrenzung des Systems durch weitere Anforderungen, damit dieses erfolgreich modelliert und implementiert werden kann.

Einige Fragen, welche das Epic noch offen lässt sind zum Beispiel:

- Was bedeutet digital verwalten?
- Soll eine Web-, Desktop- oder Mobile-Anwendung entwickelt werden?
- Welche Art von Interaktion soll für die Professoren und Studierenden möglich sein?

Die Antworten auf diese Fragen stellen weitere Anforderungen an das System dar. Diese werden wiederum als User Stories formuliert und weiterverarbeit. Selbstverständlich können auch hier nocheinmal User Stories entstehen, welche zu

generell sind und deshalb weitere Verfeinerung benötigen. Tatsächlich handelt es sich dann um ein weiteres Epic, welches wiederum durch den vorher erläuterten Prozess verfeinert werden muss. Hierbei ist es allerdings wichtig, dass User Stories nicht bis in das kleinste Detail aufgeteilt werden dürfen [Cohn 2004, p. 6]. Wie in der Definition erwähnt gehört zu einer User Story neben der *Card*, der Formulierung, auch die *Conversation*, in welcher die Details der User Story geklärt werden. Die Absicht User Stories so spezifisch wie möglich zu formulieren würde hier nur zu überflüssiger Redundanz führen, welche schlussendlich die Effizienz des Entwicklungsteam senkt.

Anhand dieser Überlegungen sind folgende zwei User Stories als Verfeinerung des oben genannten Epics verfasst:

```
Als Professor möchte ich die Kursverwaltung per
    Weboberfläche bedienen können, damit
            ich von unterschiedlichen Geräten
                                darauf zugreifen kann.
```

Diese User Story geht auf die ersten beiden Fragen ein und stellt eine in diesem Punkt eine Spezialisierung der Anforderung dar.

Hierbei sei anzumerken, dass es ausgehend von dem Epic keine eindeutig richtige oder falsche Spezialisierung gibt. In diesem konkreten Fall wären alle drei Möglichkeiten, nämlich eine Web-, Desktop- oder Mobile-Anwendung zu entwickeln, korrekt gewesen. Das Epic lässt die Interpretation von *digital* offen.

Aus diesem Grund wird in vielen Fällen für die Spezialisierung des Epics weitere Kommunikation mit dem Kunden des System von Nöten sein. Meist kann nur dieser die korrekte Interpretation der Anforderung liefern.

Wie und ob diese Kommunikation stattfindet ist allerdings Sache des (agilen) Vorgehensmodells, welches das Entwicklerteam anwendet.

```
Als Professor möchte ich die Kursinformationen
        verändern können, damit sie richtig sind.
```

In dieser zweiten User Story findet eine Spezialisierung der dritten Frage statt. Bei dieser Frage geht es nicht darum, die Mehrdeutigkeit eines verwendeten Begriffs zu klären, sondern konkrete Beispiele aus einem durch das Epic aufgespannten Anforderungsbereich zu formulieren.

So könnten in Zukunft noch weitere User Stories hinzukommen, die das Epic im Bezug auf die dritte Frage spezialisieren. Zum Beispiel könnte eine weitere User Story die eine mögliche Interaktion der Studierenden beschreiben:

```
Als Studierender möchte ich mich für die Teilnahme
  an einem Kurs registrieren ,
     damit mir diese Teilnahme angerechnet wird.
```

2.4 Was sind gute User Stories?

Die Definition von User Stories und Epics beschreibt die Struktur und die Rahmenbedingungen bei der Erstellung einer User Story. In der Praxis bleibt ausgehend von dieser Definition aber die Frage offen, welche Merkmale eine gute User Story aufweist.

[Wake 2003] hat für diesen Zweck sechs Eigenschaften für eine gute User Story unter dem Akronym **INVEST** zusammengefasst. Die Eigenschaften und ihre Bedeutung für das schreiben guter User Stories sei hier kurz genannt. Da Bill Wake die Hintergründe und Details der einzelnen Eigenschaften in seiner Veröffentlichung “INVEST in Good Stories, and SMART Tasks” schon aufführt.

Independent: Eine gute User Story ist nicht von anderen User Stories abhängig.

Negotiable: Eine gute User Story stellt keinen abgeschlossenen Vertrag dar, sondern dient als Diskussionsgrundlage.

Valuable: Eine gute User Story stellt einen Mehrwert für den Kunden dar.

Estimatable: Der Aufwand einer guten User Story ist schätzbar.

Small: Eine gute User Story ist klein und umfasst nur eine einzelne Anforderung, welche in kurzer Zeit implementiert werden kann.

Testable: Die Erfüllung einer guten User Story ist prüfbar sein.

3 AUTOMATISCHE VERARBEITUNG VON USER STORIES AM BEISPIEL VON CUCUMBER

Epics und User Stories werden, ausgehend von der oben genannten Definition, immer in einer bestimmten Struktur formuliert. Diese Struktur unterstützt nicht nur menschliche Leser, sondern ermöglicht auch die computergestützte Weiterverarbeitung der User Story. Ein Beispiel wo dies zum Einsatz kommt ist die Software *Cucumber*.

“Cucumber ist ein Tool um Akzeptanztest, welche in verständlicher Sprache verfasst sind, automatisiert auszuführen” [The Cucumber Open Source Project 2025]. Mit Hilfe von Cucumber ist es möglich sogenannte *Features* in einer von aktuell 80 gesprochenen Sprachen zu verfassen. Dabei muss lediglich die simple Gherkin-Grammatik eingehalten werden.

Basierend auf diesen Features kann Cucumber dann automatische Akzeptanztests für das jeweilige Feature durchführen. Insbesondere können durch diese automatisierte Testerzeugung auch neu erstellte User Stories direkt mit dem bestehenden System überprüft werden. Dadurch kann direkt erkannt werden, an welche Teile einer User Story noch Implementationsbedarf haben und welche Teile möglicherweise schon durch bestehenden Quellcode abgedeckt ist.

3.1 Workflow an einem Beispiel

In diesem Abschnitt wird für eine im vorhergehenden Kapitel formulierte User Story beispielhaft eine Umsetzung mit der Gherkin-Syntax und Cucumber durchgeführt. Die gewählte User Story lautet:

Als Professor möchte ich die Kursinformation verändern können, damit sie richtig sind.

Um diese User Story in der Gherkin-Grammatik auszudrücken, muss lediglich folgendes Schema befolgt werden.

```
Feature: <Titel der User Story>

Scenario: <Card der User Story>
    Given <Vorbedingung für den Test>
    When  <Veränderung, die getestet werden soll>
    Then  <Akzeptanzbedingung des Tests>
```

Wie dieses Schema für die oben genannte User Story umgesetzt werden kann ist in Listing 1 zu sehen.

```
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 correct.
  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"
```

Listing 1. `course_modification.feature`

Um basierend auf solch einer Feature-Datei nun automatisierte Tests durchzuführen, müssen für die einzelnen Testabschnitte `Given`, `When`, `Then` nun noch Template-Testfunktionen implementiert werden, welche bei der Testdurchführung die in der Feature-Datei spezifizierten Parameter, in diesem Fall die Strings `"Software Enginearing"` und `"Software Engineering"`, übergeben bekommen. Der für diesen Zweck im Beispiel verwendete Quellcode ist in Listing 2 zu finden.

Das hier vorgestellte Beispiel und der zugehörige Quellcode sind online auf GitHub in folgenden Repository verfügbar: `https://github.com/Beleg-01-Requirements-Engineering/cucumber-example`. Dort ist ebenfalls eine Anleitung zum Ausführen der Tests zu finden.

```
9  public class StepDefinitions {
10
11     private Course course;
12
13     @Given("a course with name {string}")
14     public void a_course_with_name(String
       ↪  courseName) {
15         course = new Course(courseName);
16     }
17
18     @When("the professor modifies the name
       ↪  of the course to {string}")
19     public void the_professor_modifies_
       ↪  the_name_of_the_course_to(
20         String newName
21     ) {
22         course.setName(newName);
23     }
24
25     @Then("the name of the course is
       ↪  {string}")
26     public void
       ↪  the_name_of_the_course_is(String
       ↪  name) {
27         assertEquals(name,
           ↪  course.getName());
28     }
29 }
```

Listing 2. `StepDefinitions.java`

4 AGILE VORGEHENSMODELLE

- Besonderheiten im agilen Kontext: Herausforderungen und Anpassungen durch das Agile Manifesto und dessen Prinzipien (allgemein auf das agile Manifest nochmal eingehen).
- Kanban / Scrum / FDD für unterschiedliche Teamgrößen

4.1 Kanban

- Wie sind Epics/ User Storys in den Arbeitsprozess eingegliedert?
- Kommunikation im Team: User Stories und Epics als Kommunikationsmittel und zur Schaffung einer einheitlichen Sprache (Einfluss auf Kommunikation mit Kunden)
- Nutzen? geeignet? Vorteile? Risiken?

4.2 Scrum

- Wie sind Epics/ User Storys in den Arbeitsprozess eingegliedert?
- Kommunikation im Team: User Stories und Epics als Kommunikationsmittel und zur Schaffung einer einheitlichen Sprache (Einfluss auf Kommunikation mit Kunden)
-
- Nutzen? geeignet? Vorteile? Risiken?

4.3 Feature-driven Development

Im Folgenden wird erläutert, wie Epics und User Storys im Feature-driven Development (FDD) in den Arbeitsprozess integriert sind.

4.3.1 Überblick über FDD. Das Konzept des Feature-driven Development wurde erstmals in Kapitel 6 von [P. Coad and Lefebvre 1999] beschrieben. Im Zentrum dieses Ansatzes steht

dabei der Begriff des *Feature*, das dort als "small client-valued functionality" definiert wird, also eine Funktionalität, die einen konkreten Nutzen für den Kunden oder Anwender hat.

Ein Feature wird in der folgende Form beschrieben:

```
<Aktion>    <Ergebnis>    <Objekt>
```

Ein Beispiel für ein solches Feature ist:

```
Verwalte die Informationen des Kurses.
```

Diese prägnante Formulierung hat den Vorteil, dass sie die Verständlichkeit für den Kunden und andere Stakeholder erleichtert. Die Entwicklungszeit eines Features ist auf wenige Stunden bis maximal zwei Wochen beschränkt.

FDD eignet sich besonders gut für große Softwareprojekte mit vielen Entwickler, welche dann in verschiedene Teams aufgeteilt werden. Zusätzlich zu den Entwicklungsteams wird ein Planungsteam gebildet, welches aus erfahrenen Softwarearchitekten und Domainexperten besteht.

FDD folgt dabei einem klar strukturierten Prozess, der in fünf Phasen gegliedert ist. Diese Phasen lassen sich in zwei Bereiche einteilen: die Aufgaben des Planungsteams und die der Entwicklungsteams. Das Vorgehen wird in [P. Coad and Lefebvre 1999] folgendermaßen beschrieben:

Zunächst wird ein Objektmodell der Software entwickelt (*Phase 1: Develope an Overall Model*). Anschließend wird eine Liste von Features erstellt (*Phase 2: Build a Feature List*). Darauf aufbauend wird für jedes Feature ein Entwicklungsplan erstellt (*Phase 3: Plan by Feature*). Jedem Entwicklungsteam werden dann ein oder mehrere Features zugeordnet. Die Teams erarbeitet zunächst ein Design-Paket für die jeweiligen Features (*Phase 4: Design by Feature*), anschließend erfolgt die Implementierung (*Phase 5: Build by Feature*).

Dieses strukturierte, inkrementelle Vorgehen soll ermöglichen, regelmäßig sichtbaren Fortschritt zu liefern, ohne die Prozesse dabei unnötig zu verkomplizieren.

4.3.2 Feature vs. User Story. Sowohl Features als auch User Stories dienen dazu, Anforderungen in überschaubare Teilprobleme zu zerlegen, welche einen konkreten Nutzen für den Anwender haben. Beide sind so formuliert, dass sie für den Kunden und andere Stakeholder verständlich bleiben.

Trotz dieser Gemeinsamkeiten gibt es wesentliche Unterschiede zwischen den beiden Konzepten: Bereits die unterschiedliche Formulierung zeigt, dass User Stories stärker aus der Sicht des Kunden beschrieben werden, während Features sich primär auf die funktionale Umsetzung konzentrieren.

Ein weiterer Unterschied ist die Vorgabe der Entwicklungszeit. Bei Features steht die Planbarkeit im Vordergrund, welche durch die Vorgabe von wenigen Stunden bis max. 2 Wochen Entwicklungszeit sichergestellt wird. In [S. Palmer 2002] wird zudem betont, dass ein Feature nicht nur eine triviale Methode sein sollte, die lediglich einen Attributwert zurückgibt. Bei User Stories ist es hingegen durchaus zulässig, auch kleinere Einheiten zu definieren, wie bereits in einigen Beispielen gezeigt wurde.

Weiterhin unterscheidet sich auch die Einbindung in den Entwicklungsprozess. Während User Stories unter anderem dazu dienen, Objekte und Klassen zu identifizieren, werden Features im Kontext von FDD typischerweise aus der bereits bestehenden Softwarearchitektur bzw. dem Ablaufmodell abgeleitet [S. Palmer 2002]. Dadurch sind User Stories im Entwicklungsprozess vor Features einzuordnen.

Features und User Stories lassen sich also nicht miteinander vergleichen. Eine User Story kann ein oder mehrere Features umfassen, ein einzelnen Feature kann aber auch in mehreren User Stories vorkommen[P. Coad and Lefebvre 1999].

4.3.3 Rolle von User Stories und Epics in FDD. Im Gegensatz zu Kanban und Scrum basiert das Konzept des Feature-driven Developments nicht grundlegend auf Epics und User Stories. Dennoch können sie in Phase 1 bei der Modellierung des Gesamtmodells hilfreich sein. In den späteren Phase können sie zudem das Identifizieren von Features unterstützen. Obwohl die Kommunikation in FDD überwiegend auf Features ausgerichtet ist, bieten User Stories eine nützliche Ergänzung bei der Kommunikation zwischen dem Planungsteam und den Entwicklungsteams sowie zwischen dem Planungsteam und dem Kunden. Besonders in der Zusammenarbeit mit Stakeholdern sind User Stories hilfreich, da sie die Perspektive des Anwenders stärker in den Vordergrund rücken. Zudem können sie bspw. in der Dokumentation eines Domain-Walkthroughs eingesetzt werden.

Epics und User Stories übernehmen in FDD üblicherweise nicht die Rolle der Priorisierung und Planung - diese wird durch Features abgedeckt. Stattdessen dienen sie als zusätzliches Hilfsmittel für die Modellierung und zur Verbesserung der Kommunikation. Diese Rolle sollte allerdings nicht unterschätzt werden, da eine klare und effiziente Kommunikation zwischen dem Planungsteam und den Entwicklungsteams in FDD von entscheidender Bedeutung ist.

Zusammenfassend lässt sich festhalten, dass User Stories und Epics zwar keine zentrale Rolle in FDD einnehmen, aber eine wertvolle Ergänzung darstellen können.

5 DISKUSSION

- Vorteile? Nutzen?
- Risiken?
- Probleme bei der Anwendung von User Stories und Epics: unklare Anforderungen, Gefahr der Über- oder Unterpriorisierung.
- aufzeigen wie User Story und Epic helfen um agil arbeiten zu können

6 ZUSAMMENFASSUNG UND AUSBLICK

- Zusammenfassung: User Stories und Epics als Mittel zur Förderung von effektiver Kommunikation und Anpassungsfähigkeit.
- Ausblick:
  - Potenziale durch KI-gestützte Tools zur automatischen Erstellung oder Analyse von User Stories.

REFERENCES

H. Balzert. 2009. *Lehrbuch der Softwaretechnik - Basiskonzepte und Requirements Engineering* (3rd ed.). Spektrum Akademischer Verlag. https://doi.org/10.1007/978-3-8274-2247-7

Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. *Principles behind the Agile Manifesto*. https://agilemanifesto.org/principles.html

M. Cohn. 2004. *User Stories Applied: For Agile Software Development*. Addison Wesley Longman Publishing Co., Inc.

Ron Jeffries. 2001. *Essential XP: Card, Conversation, Confirmation*. https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/

J. De Luca P. Coad and E. Lefebvre. 1999. *Java Modeling in Color with UML: Enterprise Components and Process*. Pearson.

J. Felsing S. Palmer. 2002. *A Practical Guide to Feature-Driven Development*. Prentice Hall.

I. Sommerville. 2016. *Software Engineering Global Edition* (10th ed.). Pearson.

The Cucumber Open Source Project. 2025. *Cucumber*. https://cucumber.io/

Bill Wake. 2003. *INVEST in Good Stories, and SMART Tasks - XP123*. https://xp123.com/invest-in-good-stories-and-smart-tasks/

A ANHANG 1

A.1 Übungsaufgaben

A.1.1 Epics und User Stories. Ein Lebensmittelverwaltungssystem soll Privatpersonen helfen ihre gelagerten Lebensmittel abzurufen, den Lebensmittelverbrauch zu überwachen und bei der Essensplanung unterstützen.

Finden Sie eine geeignete Anforderung für dieses System, welche einem Epic entspricht und formulieren Sie drei zugehörige User Stories mit Hilfe der Gherkin-Grammatik!