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

1 EINLEITUNG

- Motivation
- Ziel der Arbeit
- (Bedeutung von Requirements Engineering (RE) in der Softwareentwicklung und die wachsende Relevanz agiler Ansätze.)
- Zielsetzung: Untersuchung der Rolle von User Stories und Epics in agilen Vorgehensmodellen und deren Einfluss auf Kommunikation und Anpassungsfähigkeit.
- Struktur des Papers: Kurzbeschreibung der Inhalte der einzelnen Kapitel.

2 REQUIREMENTS ENGINEERING ALLGEMEIN (KURZ)

- Definition und Einordnung: RE als Teilbereich des Software Engineerings.
- Arten von Anforderungen: Funktionale und nicht-funktionale Anforderungen, Anforderungsquellen.
- Ziele und Aufgaben: Klärung von Anforderungen, Sicherstellung von Qualität und Kundenzufriedenheit.

3 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.

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.

3.1 Definition "Epic"

TODO

- Was sind User Stories und Epics?
- Unterschied zw. User Stories und klassischen Anforderungen.
- Rolle der Akzeptanzkriterien.
- Definition und Zweck (grobe Anforderungen, die später verfeinert werden).
- Hierarchie (Epic -> User Story -> Task).
- User Stories: Struktur (As a [user], I want to [goal], so that [benefit]).
- User Story und Epic klar voneinander trennen, was unterscheidet sie voneinander? (Beispiel, was später fortgesetzt wird?)
- Was macht gute User Story / gutes Epic aus.

3.2 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].

3.3 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

*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.

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.

3.4 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.

4 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.

4.1 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.

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.

```
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`

```
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`

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.

5 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

5.1 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?

5.2 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?

5.3 FDD

- 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?

6 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

7 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.

C. Ebert. 2008. *Systematisches Requirements Engineering und Management - Anforderungen ermitteln, spezifizieren, analysieren und verwalten* (2nd ed.). dpunkt.verlag.

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

R. Martin. 2014. *Agile Software Development, Principles, Patterns, and Practices* (1st ed.). Pearson.

K. Pohl and C. Rupp. 2015. *Basiswissen Requirements Engineering* (4th ed.). dpunkt.verlag.

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/

H. Wiegers and J. Beatty. 2013. *Software Requirements* (3rd ed.). Microsoft Press.

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!

A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

B ANHANG 2

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.

C ANHANG 3

Damit es im Literaturverzeichnis angezeigt wird:
[Ebert 2008], [Sommerville 2016], [Balzert 2009], [Wiegers and Beatty 2013], [Pohl and Rupp 2015], [Martin 2014]