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 drei agile Vorgehensmodelle und die beim Einsatz von Epics und User Stories auftretenden Herausforderungen betrachtet. Für die Herausforderungen wurden jeweils mögliche Lösungsvorschläge gefunden und anhand von kurzen Beispielen beschrieben.

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 EINLEITUNG UND MOTIVATION

Moderne Softwareprojekte sind häufig geprägt von großer Komplexität. Zunehmend dynamische Rahmenbedingungen haben dazu geführt, dass traditionelle Entwicklungsansätze oft nicht mehr ausreichen. Insbesondere mangelnde Flexibilität und unklare Spezifikationen können zu erheblichen Verzögerungen oder sogar zum Scheitern eines Projekts führen. Als Reaktion darauf wurden in den letzten Jahrzehnten verschiedene agile Methoden entwickelt, die unter anderem eine schnellere Reaktion auf geänderte Anforderungen ermöglichen sollen.

In der agilen Softwareentwicklung werden häufig Epics und User Stories verwendet, um Anforderungen zu beschreiben und zu strukturieren. Die Verwendung von Epics und User Stories bringt dabei Herausforderungen mit sich, wie z.B. die sinnvolle Abgrenzung von Anforderungen, die richtige Detailtiefe, eine angemessene Priorisierung und die Aufwandsschätzung.

Ziel dieser Arbeit ist es, den Einsatz von Epics und User Stories im agilen Requirements Engineering zu untersuchen. Dabei werden die Wirkung auf das agile Entwicklungsumfeld analysiert, typische Herausforderungen thematisiert und geeignete Lösungsansätze vorgestellt.

Diese Arbeit ist folgendermaßen strukturiert: Kapitel 2 gibt einen Überblick über die Grundlagen des Requirements Engineerings. Anschließend wird in Kapitel 3 das Konzept von User Stories und Epics vorgestellt und durch ein Beispiel veranschaulicht. Kapitel 4 widmet sich konkreten agilen Vorgehensmodellen und deren Umgang mit User Stories und Epics. In Kapitel 5 werden verschiedene Methoden für den Umgang mit User Stories beschrieben, darunter Methoden zum Herunterbrechen, zur Aufwandsschätzung, zur Priorisierung und zur automatischen Verarbeitung mit Hilfe von Cucumber. Kapitel 6 diskutiert, wie Epics und User Stories die Prinzipien des agilen Manifests unterstützen, wobei Vorteile und Risiken behandelt werden. Abschließend werden in Kapitel 7 die wichtigsten Erkenntnisse zusammengefasst.

*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.

2 GRUNDLAGEN REQUIREMENTS ENGINEERING

2.1 Definition und Einordnung

Die *Anforderungen* (engl. *requirements*) an ein Softwaresystem beschreiben, welche Eigenschaften von der Software erwartet werden [Sommerville 2016]. Ein einfaches Beispiel ist:

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

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 dabei 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].

2.2 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.

Nichtfunktionale Anforderungen beziehen sich meist auf übergreifende Eigenschaften der Software oder auf mehrere Funktionen gleichzeitig. Dazu zählen Aspekte wie Zuverlässigkeit, Verfügbarkeit, Informationssicherheit und Antwortzeit. Dabei 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

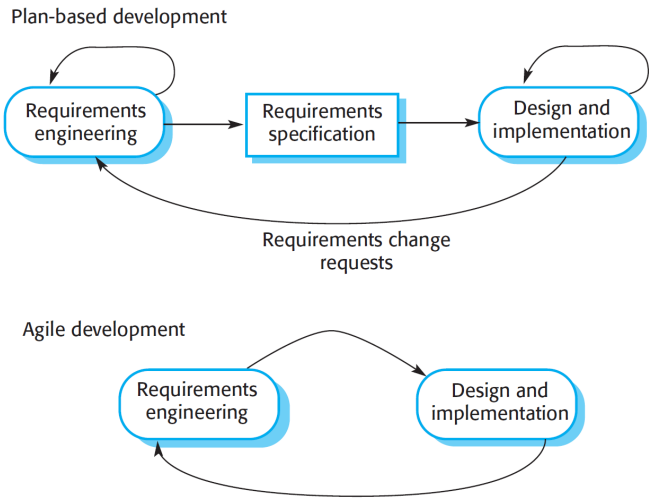


Abb. 1. Traditionelle und agile Softwareentwicklung [Sommerville 2016]

seiner E-Mail-Adresse und einem Passwort anmelden können." beschreibt zwar 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.

2.3 Requirements Engineering in der agilen Entwicklung

Bei der agilen 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 (siehe Abb. 1).

Das Requirements Engineering erfolgt daher entwicklungsbegleitend, wobei die Stakeholder fortlaufend einbezogen werden [Sommerville 2016]. Auf diese Weise wird eine kontinuierliche Überprüfung und Anpassung der Anforderungen sichergestellt. So können Änderungen schnell berücksichtigt und potenzielle Missverständnisse frühzeitig erkannt und mit geringerem Aufwand behoben werden.

3 EPICS UND USER STORIES

Wie in dem *Agile Manifesto* [Beck et al. 2001b] 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 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 "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 und Entwickler oder Entwickler und 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].

Aufgrund der Formulierung der Card der User Story, sind Entwicklerteams häufig mit der Herausforderung konfrontiert, wie nichtfunktionale Anforderungen in diesem Konzept abgebildet werden können. Da nichfunktionale Anforderungen häufig technische Details oder weitreichende Konzepte wie Sicherheit oder Effizienz der Anwendung ansprechen, besteht die Schwierigkeit darin, die Formulierung so zu wählen, dass der Wert der resultierende User Story für den Kunden immer noch ersichtlich ist.

Eine Lösungsmöglichkeit besteht hier darin das Thema der Story wieder so konkret wie möglich zu wählen und die

nichtfunktionale Anforderung in einem Epic zu verpacken. Zum Beispiel kann ein Epic lauten:

```
Als Professor sollen meine Daten geschützt sein,
          damit diese nur von mir zugreifbar sind.
```

Eine daraus resultierende User Story kann dann konkrete Maßnahmen benennen, die unternommen werden um die Daten von Professoren zu schützen.

Eine andere Möglichkeit um nichtfunktionale Anforderungen in User Stories abzubilden besteht darin, Rollen zu finden, welche ein Interesse daran haben, dass die nichtfunktionale Anforderung erfüllt wird. Wenn es zum Beispiel um die Effizienz einer Anwendung geht, kann zum Beispiel der Betreiber der Anwendung als Rolle gewählt werden. Dieser hat ein offensichtliches finanzielles Interesse daran, dass eine Anwendung mit seinen Ressourcen möglichst effizient umgeht.

Zuletzt können nichtfunktionale Anforderungen auch direkt in den Akzeptanzkritieren, also der Confirmation, mit untergebracht werden. So kann ein Entwicklerteam zum Beispiel gewisse Standards für den fertigen Quellcode fordern. Ein anderes weitverbreitetes Beispiel ist die Forderung eines bestimmten Testabdeckunganteils für den neu geschriebenen Code.

3.2 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 Interpretationsspielraum 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.

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 digitale 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 weiterverarbeitet. 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 Entwicklungsteams 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 Systems 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 in Bezug auf die dritte Frage spezialisieren. Zum Beispiel könnte eine weitere User Story 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.

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

Negotiable: Eine gute User Story stellt keine unveränderbare Spezifikation dar, sondern dient als Diskussionsgrundlage. Die Implementationsdetails werden nicht vorher festgelegt, sondern über den Lebenszyklus der User Story als Teil der Conversation erarbeitet.

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.

4 AGILE VORGEHENSMODELLE

RE und die Verwendung von User Stories und Epics wird in verschiedenen Projektmanagementmethoden unterschiedlich gehandhabt. Im Folgenden werden diese Aspekte in den agilen Modellen Kanban, Scrum und Feature-driven Development betrachtet und die Vorgehensweisen bewertet. Dabei wird auch auf die Unterschiede eingegangen. Weiterhin werden für einige Herausforderungen, die bei der Arbeit mit User Stories entstehen, konkrete Lösungsmethoden vorgestellt.

4.1 Requirement Engineering in Kanban

Für das *RE* in Kanban wird dem Team viel Spielraum gelassen, um seine eigenen Methoden und Vorgehensweisen zu entwickeln. Anders als bei Scrum, sind in Kanban keine spezifischen Rollen für einzelne Teammitglieder vorgesehen. Eine Spezialisierung einzelner Mitglieder auf bestimmte Bereiche, wie bspw. das *RE*, ist trotzdem möglich.

In Kanban wird vorgesehen, dass die Requirements und Epics in enger Zusammenarbeit mit den Stakeholdern erarbeitet werden. Dabei ist für den Austausch mit diesen keine spezifische Rolle oder Methodik vorgesehen, sodass meist ein oder mehrere Teammitglieder diese Aufgabe dynamisch übernehmen können. [Baumeister et al. 2017]

Die zu den Epics gehörenden User Stories werden meist in enger Zusammenarbeit im Team erstellt, geschätzt, priorisiert und ihre *Akzeptanzkriterien* festgelegt. Dazu steht dem Team eine Vielzahl von Methoden zur Verfügung. Einige dieser Methoden werden im Verlauf des Kapitels vorgestellt. Es wird dabei oft versucht die entstehenden User Stories so zu konzipieren, dass sie als vertikale Scheibe des Gesamtsystems umgesetzt werden können. Das bedeutet, dass ein eigenständiges, vorzeigbares Produkt entsteht, das sich durch die gesamte Architektur des Systems zieht. Dies ist allerdings vor allem in komplexen Systemen nicht immer möglich oder einfach zu aufwändig. Die technischen Details für die Implementierung der User Stories werden in Teams geklärt, damit die relevanten Teammitglieder sich auf eine Vorgehensweise einigen können. [Baumeister et al. 2017]

Wie größere User Stories in Kanban heruntergebrochen und welche Methoden dafür genutzt werden, bleibt dem Team vollständig überlassen. Eine Methode, die man dafür verwenden kann, wird in Kapitel 5.1 vorgestellt. Der Zeitaufwand für eine User Story sollte am Ende ein paar Tage betragen, aber nicht mehrere Wochen übersteigen, damit die User Stories besser geplant und auf einem Kanban-Board verwendet werden können. Gegebenenfalls kann das Team auch entscheiden eine User Story in Tasks, also noch kleinere Aufgaben, aufzuteilen. Diese können bereits etwas technischer orientiert sein. [Baumeister et al. 2017]

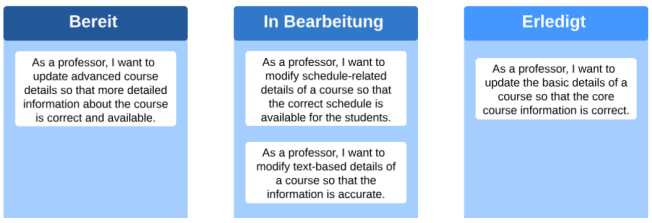


Abb. 2. Kanban Board mit User Stories

In Kanban werden die User Stories auf einem Kanban-Board visualisiert. Auf diesem werden sie je nach Bearbeitungszustand verschiedenen Listen zugeordnet (siehe Abb. 2). Meist werden die User Stories in den einzelnen Zuständen nach ihrer Priorität geordnet. In Kanban kann der Inhalt des Kanban-Boards zu jeder Zeit verändert werden. So können User Stories von dem Board entfernt, angepasst, oder bei Bedarf in kleinere User Stories oder Tasks aufgeteilt werden. So kann das Team sehr schnell auf Änderungen reagieren. [Baumeister et al. 2017]

In Kanban liegt es im Ermessen des Teams, wann aus den fertig umgesetzten User Stories ein neuer Release erstellt wird. So kann das Team flexibel entscheiden, wie lang ein Feedbackloop ist. [Baumeister et al. 2017]

4.1.1 Vorteile und Nachteile von Requirement Engineering in Kanban.

Die Art und Weise, wie das *RE* in Kanban durchgeführt wird, ermöglicht es dem Team sehr flexibel zu arbeiten. Das liegt daran, dass die User Stories jederzeit auf dem Kanban-Board angepasst, entfernt, hinzugefügt oder umgeordnet werden können. Dadurch kann das Team schnell auf Änderungen wie Bugs und Requirementsänderungen reagieren. Das Kanban-Board ermöglicht es auch dem Team und dem Stakeholder leichter den Überblick über den Fortschritt des Projekts zu behalten, und hilft dabei, die Entwicklung zu dokumentieren.

Eine Studie in *"Agile Processes in Software Engineering and Extreme Programming"* [Baumeister et al. 2017] zeigt, dass ein gutes Team mit Kanban konstant und iterativ entwickeln und sich zusätzlich an dynamisch verändernde Anforderungen anpassen kann. So wird das Team dazu angeregt, Lösungen aus eigener Initiative zu entwickeln, ohne an eine feste Ablaufstruktur gebunden zu sein. [Liskin et al. 2014]

Durch das *RE* in Kanban entstehen allerdings auch einige Herausforderungen für das Team. Aufgrund der flexiblen Struktur braucht das Team eine große Selbstverantwortung und Teamdisziplin. So muss es selbständig *RE* in den dynamischen Entwicklungsprozess integrieren. Auch die Kommunikation und den Austausch mit den Stakeholdern muss das Team passend einplanen. Vor allem bei größeren oder unerfahrenen Teams kann dieser zusätzliche Aufwand für die Planung der Ablaufstruktur ein größeres Problem darstellen. [Bundesministerium des Innern und für Heimat 2025]

4.2 Requirement Engineering in Scrum

Anders als bei Kanban, ist in Scrum für das Team eine dedizierte Phase vor einem Sprint für das *RE* bestimmt. In dieser beschäftigt sich das Team mit den User Stories für den nächsten *Sprint Backlog*. Diese Phase wird als *Sprint Planning* bezeichnet. [N. Ramadan and S. Megahed 2016]

Ein weiterer Unterschied zu Kanban besteht darin, dass beim *RE* bestimmte vordefinierte Rollen im Team eine große Bedeutung spielen. So ist z.B. der *Scrum Master* dafür verantwortlich sicherzustellen, dass das Team sich mit den User Stories ordentlich beschäftigen kann. Die wichtigste Rolle für das *RE* in Scrum ist der *Product Owner*. Dieser ist dafür verantwortlich den *Product Backlog* zu verwalten. Das bedeutet, dass er für das Formulieren und Herunterbrechen der Epics und User Stories zuständig ist. Es wird allerdings empfohlen, dass er dafür auch Teile des Teams mit einbezieht. So ist er auch für das Ermitteln der Requirements der Stakeholder zuständig und dient oft auch als direkter Ansprechpartner für diese. [N. Ramadan and S. Megahed 2016]

Weiterhin ist er dafür zuständig, die User Stories im *Product Backlog* nach ihrer Priorität zu ordnen und auf ihre Machbarkeit zu kontrollieren. Eine Methode, mit der man User Stories nach ihrer Priorität ordnen kann, wird in Kapitel 5.3 erklärt. Er soll während des *Sprint Plannings* geeignete User Stories an das Team kommunizieren, um sicherzustellen, dass das Verständnis des Teams mit dem des Stakeholders vereinbar ist. [N. Ramadan and S. Megahed 2016]

Das Team selbst hat hinsichtlich des *RE* die Aufgaben, während des *Sprint Plannings* die einzelnen User Stories für den *Sprint Backlog* auszuwählen und deren Aufwand zu schätzen, sie zu priorisieren und ihre *Akzeptanzkriterien* festzulegen. Eine Methode zum Schätzen des Aufwands, wird in Kapitel 5.2 vorgestellt. Weiterhin muss das Team sich im *Sprint Planning* auf die technischen Details der Implementierung der User Stories einigen. [4]

In Scrum sind Epics ein größeres Themengebiet aus zusammengehörenden Requirements. Sie repräsentieren einen Themenkomplex, der zu groß ist, um ihn in einem Sprint vollständig zu implementieren. [N. Ramadan and S. Megahed 2016]

User Stories in Scrum definieren die Ziele eines Sprints und bilden den Inhalt des *Sprint Backlogs*. Sie helfen dem Team dabei den Fortschritt des Sprints in z.B. einem Burndown Chart zu erfassen. Dabei sollten die User Stories einen kleinen und ähnlichen Umfang haben, sodass sich die Linie für die Abarbeitung der Story Points eines Sprints an eine linear fallende Linie annähert (siehe Abb. 3). So kann das Team leichter erkennen, ob sie den Zeitplan für den aktuellen Sprint einhalten. Sollte erkannt werden, dass der Sprint nicht planmäßig verläuft, können im *Daily Scrum* Gegenmaßnahmen eingeleitet werden. Sind die User Stories zu groß gewählt, fällt der Graph auf dem Burndown Chart gegen Ende des Sprints immer stärker ab, was oft dazu führt, dass zu spät erkannt wird, dass der Zeitplan nicht eingehalten werden kann (siehe Abb. 3). [N. Ramadan and S. Megahed 2016]

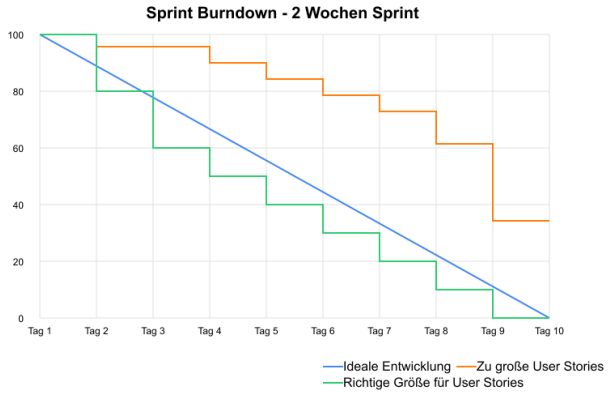


Abb. 3. Burndown Chart für einen 2 Wochen Sprint

Ein grober Richtwert, wie klein die User Stories sein sollten, liefert *"The Humanizing Work Guide to Splitting User Stories"* [R. Lawrence and P. Green 2025]. Hier wird empfohlen, dass die Größe einer User Story in Scrum zwischen $\frac{1}{6}$ und $\frac{1}{10}$ der *Velocity* des Teams liegen soll. Bei der *Velocity* handelt es sich dabei um die Geschwindigkeit der Entwicklung in einem Sprint. Sie wird oft in Form von Story Points ausgedrückt und wird vor Beginn des Sprints festgelegt. [R. Lawrence and P. Green 2025]

Bei dem Zusammenstellen des *Sprint Backlogs* sollte darauf geachtet werden, dass die verwendeten User Stories zusammen ein auslieferbares Produkt bilden, das am Sprint-Ende den relevanten Stakeholdern als neues Release vorgestellt werden kann. Dabei kann sich der Stakeholder am *Sprint Backlog* orientieren, um besser einzuschätzen, ob das Resultat des Sprints seinen Erwartungen entspricht. [N. Ramadan and S. Megahed 2016]

4.2.1 Vorteile und Nachteile von Requirement Engineering in Scrum. Das *RE* in Scrum profitiert von dem klar strukturierten Ablaufplan für jede Iteration. Das *Sprint Planning* ermöglicht es dem Team, sich in einem festgelegten Zeitrahmen ausschließlich darauf zu konzentrieren, welche User Stories im nächsten Sprint bearbeitet werden und wie deren Umsetzung erfolgt. Das erleichtert es vor allem bei unerfahrenen Teams, alle Mitglieder auf die gleiche Verständnisebene bezüglich der Ziele des Sprints zu halten. Das Team muss sich also keine Gedanken darüber machen, wie das *RE* dynamisch in die Entwicklung eingebunden werden soll.

Es ist auch vorteilhaft, dass die Stakeholder in regelmäßigen Abständen über den Fortschritt des Projekts anschaulich informiert werden und dadurch die Kommunikation mit diesen erleichtert und oft klarer wird. Die festen Iterationsintervalle erleichtern es dem Team, die verwendeten Methoden bei Bedarf anzupassen, um folgende Sprints zu verbessern. So kann z.B. die Schätzungsmethode zwischen Sprints einfach gewechselt werden, wenn die aktuelle Methode nicht die geforderte Genauigkeit liefert. Weiterhin wird ein Großteil des *RE* auf den *Product Owner* übertragen, was es dem restlichen Team ermöglicht, sich stärker auf die Implementierung zu fokussieren.

Zwar erleichtert die feste Ablaufstruktur die Planung des *RE* in Scrum, allerdings wird sie dadurch auch unflexibler. So ist es in Scrum nicht vorgesehen den *Sprint Backlog* während der Sprintphase anzupassen, was es dem Team erschwert, auf kurzfristige Änderungen zu reagieren. Dies kann vor allem bei Bugs, integriertem Kundensupport und Customizing zu Problemen mit Stakeholdern führen. Das liegt daran, dass in diesen Kategorien sehr kurzfristig Anforderungen mit hoher Priorität entstehen können.

Weiterhin hängt der Erfolg eines Sprints stark von der Arbeit ab, die während des *Sprint Plannings* geleistet wurde, da die Ergebnisse aus dieser Phase während eines Sprints nicht mehr angepasst werden sollen. Dies kann dazu führen, dass ein schwerer Fehler im *Sprint Planning* zum Abbruch des Sprints führen kann. Ähnliches gilt auch für den *Product Owner*. Dieser nimmt zwar dem restlichen Team einen signifikanten Teil des *RE* ab, hat aber entsprechend viel Verantwortung. Der Erfolg des Projekts hängt davon ab, dass er die Requirements der Stakeholder korrekt analysiert und verständlich an das Team kommuniziert.

4.3 Feature Driven Development

Im Folgenden wird erläutert, wie Epics und User Stories 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 [Coad et al. 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 folgenden Form beschrieben:

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

Ein Beispiel für ein solches Feature ist:

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

Die Entwicklungszeit eines Features ist auf wenige Stunden bis maximal zwei Wochen beschränkt.

FDD eignet sich besonders gut für große Softwareprojekte mit vielen Entwicklern, die 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 (siehe Abb. 4). Diese Phasen lassen sich in zwei Bereiche einteilen: die Aufgaben des Planungsteams und die der Entwicklungsteams. Das Vorgehen wird in [Coad et al. 1999] folgendermaßen beschrieben:

Zunächst wird ein Objektmodell der Software entwickelt (*Phase 1: Develop 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

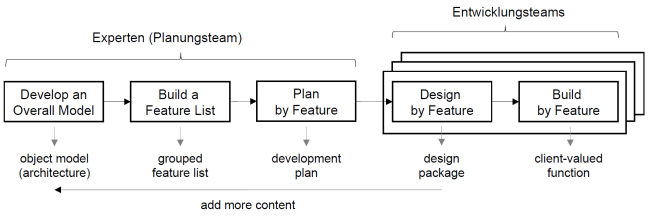


Abb. 4. Feature Driven Development

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 [Hunt 2006].

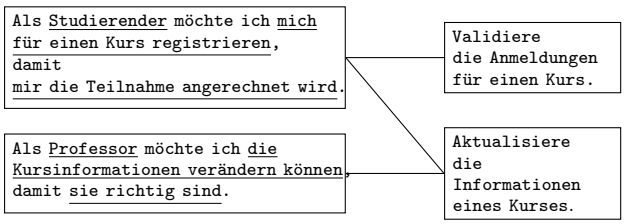
4.3.2 Feature vs. User Story. Sowohl Features als auch User Stories dienen dazu, Anforderungen in überschaubare Teilprobleme zu zerlegen, welche einen konkreten Nutzen für den Anwender haben. Beide sind so formuliert, dass sie für den Kunden und andere Stakeholder verständlich bleiben.

Trotz dieser Gemeinsamkeiten gibt es wesentliche Unterschiede zwischen den beiden Konzepten. Bereits die unterschiedliche Formulierung zeigt, dass User Stories aus der Sicht des Anwenders formuliert 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 [Hunt 2006], welche durch die Vorgabe von wenigen Stunden bis max. 2 Wochen Entwicklungszeit sichergestellt wird. In [Palmer and Felsing 2002] wird zudem betont, dass ein Feature nicht nur eine triviale Methode ist, 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 [Palmer and Felsing 2002].

Features und User Stories lassen sich also nicht miteinander vergleichen. Eine User Story kann ein oder mehrere Features enthalten, ein einzelnes Feature kann aber auch in mehreren User Stories vorkommen [Hunt 2006]. Zur Demonstration soll das folgende Beispiel genügen:



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 Phasen können sie zudem das Identifizieren von Features unterstützen [Hunt 2006]. 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 [Palmer and Felsing 2002].

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, Dokumentation und 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 KONKRETE ANWENDUNGSMETHODEN FÜR DIE KONTROLLE VON USER STORIES

Bei der Betrachtung der agilen Methoden wurde in dieser Arbeit oft darauf hingewiesen, dass die verwendeten User Stories gewisse Kriterien erfüllen sollten. Zu diesen Kriterien gehört unter anderem, dass die User Story weit genug heruntergebrochen, ihr Aufwand geschätzt und sie priorisiert wurde. Im Folgenden wird zu diesen Kriterien jeweils eine konkrete Methodik vorgestellt. Weiterhin wird auch noch erläutert, was eine *Definition of Ready* (*DoR*) und eine *Definition of Done* (*DoD*) im Zusammenhang mit User Stories ist und wofür sie genutzt werden.

5.1 Herunterbrechen von User Stories

Eine Methode, um eine zu große User Story richtig herunterzubrechen, wurde von *humanizing Work* veröffentlicht. Dabei handelt es sich um ein Story Splitting Flowchart, das von Richard Lawrence entwickelt wurde. Diese Methode wurde hauptsächlich für Scrum entwickelt, kann aber auch in anderen agilen Methoden verwendet werden. [R. Lawrence and P. Green 2025]

Bei dieser Methode wird zunächst geprüft, ob die User Story das *INVEST*-Modell erfüllt. Sollte dies nicht der Fall sein, muss die User Story nochmal überarbeitet werden. Wenn der geschätzte Aufwand der User Story über dem vordefinierten Bereich liegt (hier zw. $\frac{1}{6}$ und $\frac{1}{10}$ der *Velocity*), kann mit dem

Herunterbrechen fortgefahren werden. [R. Lawrence and P. Green 2025]

In dem Flowchart werden dafür verschiedene Kategorien erläutert, nach denen man die User Story weiter herunterbrechen kann. Zu diesen Kategorien gehören die verschiedenen Arbeitsschritte, genutzte Operationen, Businessregeln, verwendete Datentypen und Schnittstellen. Weiterhin kann man sie auch nach dem größten Aufwand und der Komplexität unterteilen. Es ist auch möglich, die User Story in Teile aufzuteilen, die bestimmte nichtfunktionale Anforderungen erfüllen müssen, und solche, die ohne diese Anforderungen auskommen. [R. Lawrence and P. Green 2025]

Nach dem Herunterbrechen nach einer der Kriterien, werden die entstandenen User Stories noch evaluiert. Dabei wird geprüft, ob sie unter anderem das *INVEST*-Modell erfüllen, einen ähnlichen Aufwand aufweisen, dieser innerhalb des vorgegebenen Bereichs liegt und keine der User Stories redundant ist. Wenn alle Schritte in dem Flowchart richtig umgesetzt wurden, dann erhält man am Ende gute User Stories mit der gewünschten Größe. [R. Lawrence and P. Green 2025]

5.2 Aufwandschätzung von User Stories über Planning Poker mit Fibonacci-Zahlen

Von allen User Stories muss der Aufwand geschätzt werden, um zu bestimmen, ob ihr Umfang sich für das verwendete Projektmodell eignet. Eine Methode, die man dazu anwenden kann, ist das Planning Poker mit Fibonacci-Zahlen. Dabei wird die Zahlenfolge als der Schätzwert verwendet. [Tamrakar and Jørgensen 2012]

Sie ist besser für das Schätzen geeignet als eine lineare Zahlenfolge, da die Sprünge zwischen den einzelnen Zahlen nach oben immer größer werden. So kann man verhindern, dass versucht wird große User Stories unrealistisch genau zu schätzen. Das ist vorteilhaft, weil aus Beobachtungen hervorgegangen ist, dass die Aufwandschätzung mit zunehmendem Umfang der User Story immer ungenauer wird. [Tamrakar and Jørgensen 2012]

Das Planning Poker folgt einem festen Ablaufplan. Im ersten Schritt wird die User Story zunächst dem Team präsentiert. Daraufhin wird sie im Team diskutiert, damit ein einheitliches Verständnis entsteht. Im dritten Schritt schätzt dann jedes Mitglied den Aufwand mithilfe der Fibonacci-Zahlen. Welcher Aufwand genau geschätzt wird, ist dem Team überlassen. Es können beispielsweise die Story Points oder die Komplexität der User Story geschätzt werden. [Tamrakar and Jørgensen 2012]

Anschließend werden die Ergebnisse der Schätzungen präsentiert und die Teammitglieder mit der größten Differenz stellen die Gründe für ihre Schätzung vor. Die Abweichungen werden dann im Team diskutiert. Dabei wird versucht einen gemeinsamen Kompromiss zu finden. Wenn sich das Team in der Diskussion einigen kann, dann wird die Schätzung finalisiert. Dafür kann man z.B. den Median oder Mittelwert aller Schätzungen bilden. Die verwendete Methode bleibt dabei dem Team überlassen. Sollte sich das Team nicht einigen können, muss der Prozess wiederholt werden. [Tamrakar and Jørgensen 2012]

5.3 Priorisieren von User Stories mit MoSCoW-Methode

User Stories müssen priorisiert werden, damit sie richtig zugewiesen und eingeplant werden können. Dabei müssen die Abhängigkeiten von User Stories untereinander beachtet werden, wenn diese vorhanden sind. Für die Priorisierung kann man z.B. die *MoSCoW*-Methode verwenden. Dabei werden die User Stories in vier Kategorien eingeteilt. [Miranda 2022]

Dazu gehören als erstes die *Muss*-Kategorie (*Must*). In diese werden User Stories eingeordnet, die absolut essentiell für die Grundfunktionalität sind und ohne die das Projekt nicht auskommt. In die *Soll*-Kategorie (*Should*) kommen User Stories, die noch durchaus viel Wert für die Stakeholder aufweisen, aber nicht unmittelbar zur Grundfunktionalität des Produkts gehören und somit nicht unbedingt direkt in der ersten Version umgesetzt werden müssen. Als drittes gibt es die *Kann*-Kategorie (*Could*). Die User Stories in dieser Kategorie sind oft nützlich, haben aber für den Stakeholder einen eher geringeren Wert. Sie werden oft auch als *"Nice to have"* bezeichnet. Die letzte Kategorie ist die *Nicht*-Kategorie (*Will not*). In diese fallen die User Stories, die außerhalb des Scopes liegen und höchstens in zukünftigen Versionen von Bedeutung sein könnten. [Miranda 2022]

Die erfolgreiche Kategorisierung der User Stories ist bei dieser Methode stark von einer präzisen Aufwandschätzung abhängig. [Miranda 2022]

5.4 Definition of Ready und Definition of Done

Damit das Team erkennt, wann eine User Story bereit für die Bearbeitung, oder fertig implementiert ist, definieren die meisten Unternehmen eine *DoR* und eine *DoD*. Bei einer *DoR* handelt es sich dabei um eine Liste von Kategorien, die alle User Stories erfüllen müssen, bevor sie für die Umsetzung freigegeben wird. Der genaue Inhalt ist bei der *DoR* und der *DoD* dem Unternehmen überlassen. In dem *DoR* könnten durchaus folgende Kriterien enthalten sein:

- das INVEST-Modell ist erfüllt
- der Aufwand wurde geschätzt und ist innerhalb des vordefinierten Bereichs
- die Priorisierung wurde durchgeführt
- die *Akzeptanzkriterien* sind festgelegt
- die technischen Details sind geklärt

Die *Akzeptanzkriterien* (auch *Confirmation* genannt) sind zwischen den verschiedenen User Stories durchaus unterschiedlich. Sie werden oft von Stakeholdern vorgegeben, können aber auch nichtfunktionale Anforderungen enthalten, oder von dem erfolgreichen Abschluss vordefinierter Tests (z.B. Unit-Tests) abhängen.

In der *DoD* werden die Kriterien aufgelistet, die alle User Stories erfüllen müssen, damit sie als vollständig umgesetzt

gelten. Zu diesen Kategorien können unter anderem folgende gehören:

- das Erfüllen der *Akzeptanzkriterien*
- das Dokumentieren der Änderungen
- der erfolgreiche Abschluss aller Tests
- die Abnahme durch den *Product Owner* oder Projektleiter

5.5 Automatische Verarbeitung von User Stories am Beispiel von Cucumber

Epics und User Stories werden, ausgehend von der Definition in 3, 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 Akzeptanztests, 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 welchen Teilen einer User Story noch Implementationsbedarf besteht und welche Teile möglicherweise schon durch bestehenden Quellcode abgedeckt sind.

5.5.1 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.

6 DISKUSSION

Epics und User Stories sind eine Methode, welche für Requirements Engineering im Sinn des Agilen Manifests [Beck et al. 2001a] geeignet sind. Dies kann durch folgende drei Grundsätze des Agilen Manifests verdeutlicht werden.

6.1 Individuals and interactions over processes and tools

Epics und User Stories tragen zu einer klaren Kommunikation sowohl im Team (intern) als auch zwischen Kunde und Entwicklerteam (extern) bei. Dies zeigt sich auch häufig darin, dass durch den Einsatz von User Stories die Perspektive des Kunden besser verstanden werden kann, da dieser tief in den Prozess der Anforderungsanalyse involviert ist.

6.2 Customer collaboration over contract negotiation

Wie schon im vorhergehenden Punkt angesprochen ist der Kunde beim Erstellen und Bearbeiten von User Stories fest mit eingebunden. In traditionellen Methoden der Anforderungsanalyse werden häufig Anforderungen einmalig erstellt und dann vertraglich festgesetzt. Im Gegenteil dazu werden bei User Stories Anforderungen auch nach ihrer Erstellung noch bearbeitet. Ganz nach dem Qualitätsmerkmal "Negotiable" der INVEST-Kriterien, werden mit User Stories immer wieder durch Kommunikation mit dem Kunden Details erarbeitet oder Teile der User Story angepasst, da sich die zugrundeliegenden Anforderungen von Seiten des Kunden verändert haben.

6.3 Responding to change over following a plan

Zuletzt befähigen Epics und User Stories auch agile Entwicklungsteams auf sich verändernde Anforderungen zu reagieren. Im letzten Abschnitt wurde schon die Kommunikation zum Kunden als wichtiges Mittel zur Reaktion auf Veränderungen erwähnt. Darüber hinaus bieten User Stories aber auch durch ihre Größe eine geeignete Struktur für ein Entwicklungsteam, um einerseits Anforderungen für die Implementation auszuwählen, aber auch in regelmäßigen Zeitabständen den Kunden über den Stand des Projektes zu informieren und auf dessen Rückmeldung zu reagieren.

Trotz der wesentlichen Vorteile von Epics und User Stories sollen hier auch nocheinmal einige Herausforderungen aufgeführt werden, auf welche Entwicklungsteams bei ihrem Einsatz stoßen werden.

Wie schon bei der Einführung von Epics und User Stories erwähnt, ist das Formulieren von nichtfunktionalen Anforderungen als User Stories schwierig. Es gibt Techniken, die dieses Problem lösen, doch erfordert deren Einsatz ein Bewusstsein für diese Herausforderung und ein Verständnis dieser Techniken.

Außerdem besteht die Gefahr der Überspezialisierung auf einen bestimmten Kunden. Wenn im Rahmen der Anforderungsanalyse ein Großteil der User Stories nur von einem einzelnen Kunden formuliert wird, können dadurch Anforderungen, welche nur dieser eine Kunde hat, überbewertet und Anforderungen von anderen Kunden unterbewertet werden. Auch die Anforderungen, welche möglicherweise durch andere Stakeholder der Software (Maintenance, Deployment etc.) an die Software gestellt werden können in diesem Prozess untergehen.

Der effiziente Einsatz von Epics und User Stories erfordert an vielen Stellen Erfahrung. Dies kann insbesondere beim Schätzen und Herunterbrechen von User Stories zu Effizienzeinbußen in unerfahrenen Teams führen. Ein einfaches Mittel, dem entgegenzuwirken, ist Teammitglieder mit unterschiedlichen Erfahrungsstufen zusammenzustellen und einen aktiven Wissenstransfer anzuregen.

7 ZUSAMMENFASSUNG

Im Rahmen dieser Arbeit wurden Epics und User Stories als Methode des agilen Requirements Engineering vorgestellt.

Der korrekte Umgang mit User Stories wurde theoretisch anhand des INVEST-Modells erörtert und exemplarisch an den drei verschiedenen agilen Vorgehensmodellen Kanban, Scrum und Feature Driven Development dargestellt. Dabei wurden auch mit dem Einsatz von Epics und User Stories verbundene Herausforderungen angesprochen und mögliche Lösungsmethoden vorgeschlagen.

Zuletzt wurden die Erkenntnisse dieser Arbeit ausgewertet und die Bedeutung von Epics und User Stories für die agile Softwareentwicklung diskutiert. Basierend auf dieser Diskussion kommen wir zu dem Schluss, dass Epics und User Stories ein geeignetes Mittel für das Requirements Engineering in agilen Softwareprojekten sind und wesentlich zu deren Erfolg beitragen.

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

H. Baumeister, H. Lichter, and M. Riebisch. 2017. *Agile Processes in Software Engineering and Extreme Programming*. SpringerOpen.

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, K. Schwaber, J. Sutherland, and D. Thomas. 2001a. *Agile Manifesto*. https://agilemanifesto.org

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, K. Schwaber, J. Sutherland, and D. Thomas. 2001b. *Principles behind the Agile Manifesto*. https://agilemanifesto.org/principles.html

Bundesministerium des Innern und für Heimat. 2025. *Kanban*. https://www.orghandbuch.de/Webs/OHB/DE/OrganisationshandbuchNEU/4_MethodenUndTechniken/Methoden_A_bis_Z/Kanban/Kanban_node.html

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

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

J. Hunt. 2006. *Agile Software Construction*. Springer.

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

O. Liskin, K. Schneider, F. Fagerholm, and J. Münch. 2014. Understanding the Role of Requirements Artifacts in Kanban. veröffentlicht auf ResearchGate. https://www.researchgate.net/publication/260370694_Understanding_the_Role_of_Requirements_Artifacts_in_Kanban

E. Miranda. 2022. Moscow Rules: A Quantitative Exposé. In *Agile Processes in Software Engineering and Extreme Programming* (Cham, 2022), Viktoria Stray, Klaas-Jan Stol, Maria Paasivaara, and Philippe Kruchten (Eds.). Springer International Publishing, 19–34.

N. Ramadan and S. Megahed. 2016. Requirements Engineering in Scrum Framework. 149, 8 (15 09 2016), 24–29. https://doi.org/10.5120/ijca2016911530

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

R. Lawrence and P. Green. 2025. *The Humanizing Work Guide to Splitting User Stories*. https://www.humanizingwork.com/the-

humanizing-work-guide-to-splitting-user-stories/
I. Sommerville. 2016. *Software Engineering Global Edition* (10th ed.). Pearson.
R. Tamrakar and M. Jørgensen. 2012. Does the use of Fibonacci numbers in Planning Poker affect effort estimates?. In *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)* (Ciudad Real, Spain, 2012). IET, 228–232. https://doi.org/10.1049/ic.2012.0030
The Cucumber Open Source Project. 2025. *Cucumber*. https://cucumber.io/
W. C. 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!

A.1.2 FDD und nichtfunktionale Anforderungen. Formulieren Sie für jede Ihrer User Stories aus Aufgabe A.1.1 ein oder mehrere Features in der für FDD typischen Form, die für deren Umsetzung erforderlich sind. Die Wiederverwendung von Features ist erlaubt und sogar erwünscht. Falls sich für eine User Story kein passendes Feature formulieren lässt, begründen Sie dies.

Überlegen Sie außerdem, welche nichtfunktionalen Anforderungen bei der Implementierung der Features berücksichtigt werden sollten.

A.1.3 User Story herunterbrechen. Die folgende User Story wurde für die Entwicklung eines typischen Online-Shops erstellt. Nutzen Sie die in Kapitel 5.1 definierte Methode, um diese User Story in kleinere herunterzubrechen. Verwenden Sie dazu eine passende Kategorie aus dem Flowchart aus "*The Humanizing Work Guide to Splitting User Stories*" (https://www.humanizingwork.com/the-humanizing-work-guide-to-splitting-user-stories/) [R. Lawrence and P. Green 2025]. Achten Sie darauf, dass die entstehenden User Stories einen relativ ähnlichen Umfang haben sollen.

User Story: "Als Benutzer muss ich in der Lage sein, Produkte zu suchen, auszuwählen, in den Warenkorb zu legen und den Kauf abzuschließen, damit ich sie online einkaufen kann."