Vorlesung:
Requirements Engineering
Agiles Requirements Engineering

Prof. Dr. Gerd Beneken

Lernziele

- Sie verstehen, was **„agil“ für das Requirements Engineering bedeutet** und welche Rolle der Requirements Engineer spielt.
- Sie kennen die **Vorteile und die Grenzen**, wenn sie Anforderungen **just in time** detaillieren und sofort in Software umsetzen.
- Sie können selbstständig **User Storys** formulieren und zu umfangreiche **Storys in kleinere zerlegen**. Sie haben verstanden, was eine gute User Story ausmacht.
- Die „**Spezifikation über Beispiele**“ können Sie praktisch anwenden.
- Sie haben ein grobes Verständnis, wie das agile Vorgehensmodell **Scrum** funktioniert, und Sie kennen dort die besondere Verantwortung des **Product Owner**.
- Sie verstehen, wie mit Anforderungen in Scrum umgegangen wird, und können unter Anleitung als Product Owner arbeiten.

Themen

- **Was bedeutet agil?**
- Anforderungsermittlung in agilen Projekten
- Funktionale Anforderungen mit User Stories und Spezifikation durch Beispiele
- Nichtfunktionale Anforderungen als Constraints
- Scrum als agiles Framework und Anforderungsverwaltung im Product Backlog

Was bedeutet „agil“?

Agiles Manifest als Leitlinie

1. **Individuen und Interaktionen** mehr als Prozesse und Werkzeuge
2. **Funktionierende Software** mehr als umfassende Dokumentation
3. **Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung
4. **Reagieren auf Veränderung** mehr als das Befolgen eines Plans

- Achtung: „mehr als“ bedeutet nicht, dass der rechte Teil weggelassen werden kann.

[Quelle: http://agilemanifesto.org/]

Themen

- Was bedeutet agil?
- **Anforderungsermittlung in agilen Projekten**
- Funktionale Anforderungen mit User Stories und Spezifikation durch Beispiele
- Nichtfunktionale Anforderungen als Constraints
- **Scrum** als agiles Framework

  und Anforderungsverwaltung: Product Backlog

Agile Anforderungsermittlung **Just in Time** Requirements

- Anforderungen werden zunächst nur grob erhoben
  - Eher als grober Stichpunkt, als „Merker“ für ausstehendes Gespräch
  - Eher nicht als Dokument, das alle Details enthält
- Anforderungen werden erst dann detailliert, wenn Sie **kurz vor der Implementierung** stehen (just in time)
  - Erste Verfeinerung (über Gespräche) in Teilanforderungen und Akzeptanzkriterien, vor der Planungsrunde der nächsten Iteration
    Ziel: Anforderung **klein genug**, dass diese **schätzbar** und in der nächsten Iteration **umsetzbar**
  - Weitere Verfeinerung dann im Gespräch zwischen Entwickler und Anforderungsquelle (z.B. Kunde vor Ort)
  - Schnelles Feedback: Kunde sieht schnell, ob die richtigen Anforderungen richtig umgesetzt wurden, da er schnell testen kann
- Wichtig: Es wird nur soviel wie gerade eben nötig dokumentiert (eventuell entsteht sogar keine dauerhafte Dokumentation)

Agile Anforderungsermittlung
Fachexperte / Entscheider vor Ort

- Zwei Rollen müssen möglichst nahe an den Entwicklern sein (vor Ort!)
  1. Der **Fachexperte**, der die Details der Anforderungen kennt
  2. Der **Entscheider (genau einer!)**, der sagt, welche Anforderungen wann umgesetzt werden (kann die gleiche Person wie der Fachexperte sein)
- Ziel: Möglichst **kurze Feedbackschleife und schneller Informationsaustausch** zwischen Fachexperte und Entwickler
  - Fachexperte sieht sofort, ob Entwickler die Anforderungen richtig verstanden hat. – Er kann die Anforderung sofort testen.
  - Entwickler kann bei Unklarheiten sehr schnell Nachfragen und Hypothesen über die Anforderungen mit Fachexperten klären
- Ziel: Entscheidungen werden zeitnah getroffen, Priorisierung der Anforderungen erfolgt zeitnah
  - Halbfertige Anforderungen liegen nicht auf Vorrat irgendwo herum, sondern wandern schnell nach ihrer detaillierten Entdeckung in Produktion (vgl. „Value Stream Mapping“)
  - Keine Unklarheiten wegen Politischer Wetterlage oder weil das „CCB“ noch nicht tagen konnte

Themen

- Was bedeutet agil?
- Anforderungsermittlung in agilen Projekten
- **Funktionale Anforderungen mit User Stories und Spezifikation durch Beispiele**
- Nichtfunktionale Anforderungen als Constraints
- **Scrum** als agiles Framework
  und Anforderungsverwaltung: Product Backlog

User Storys

A **User Story** is a story about how the system is supposed to **solve a problem or support a business process**. Each User Story is written on a Story Card, and represents a chunk of functionality that is coherent in some way to the customer.

[Ward Cunnigham]

Eine User Story ist eine kurze Beschreibung einer Funktion des Systems aus der **Perspektive des Benutzers**. Die User Story unterstützt den Benutzer bei der Lösung eines Problems oder der **Erreichung eines geschäftlichen Ziels**. Sie ist Gesprächsgrundlage für die Klärung der Details zu der Funktion.

[Arbeitsdefinition]

User Storys
Beispiele für Campus-Informationsystem

- Als Student will ich mich online zu einer Vorlesung anmelden, so dass ich einen Platz in der Übung erhalte
- Als Professor will ich eine Anmeldestatistik abrufen, damit ich meine Vorlesung planen kann
- Als Student will ich meinen Stundenplan online abrufen um mich in der Hochschule zu orientieren

Typische Form der Beschreibung:

As a [**person in a role**] I want to [**perform some activity**] so that [**some goal is achieved**].

User Storys
Vorteile der Beschreibungsform

As a [**person in a role**] I want to [**perform some activity**] so that [**some goal is achieved**].

- **Rolle des Benutzers explizit**: nicht einfach nur „User“ sondern z.B. Administrator, Sachbearbeiter, Kunde, … oder noch konkreter als Persona (siehe unten)
- **Konkrete Interaktion** mit dem System muss angegeben werden
- **Geschäftswert explizit**: Jede Story muss dem Benutzer helfen ein Ziel zu erreichen oder ein Problem zu lösen
  = Vermeidung der „Goldenen Henkel“

User Stories: Details mündlich, so spät wie möglich

The Three C

- **Card**
  - Karteikarte mit groben Informationen, als Erinnerung
  - Verwendet für Planung und Risikoabschätzung
- **Conversation**
  - Kunde vor Ort erzählt die tatsächliche Anforderung direkt den Entwicklern
  - Verbale, direkte Kommunikation über die User Story bei Planung und Umsetzung (Nennung der Details), Diskussion über die US
  - Ggf. Unterstützt durch Dokumente, Screenshots, Spreadsheets, …
- **Confirmation**
  - Bestätigung der Umsetzung der UserStory durch Akzeptanztests
  - Kunde erzählt, wie er testen will, ob die US richtig umgesetzt ist
  - Akzeptanztests werden (zum Teil) von den Programmierern umgesetzt

Akzeptanzkriterien

- *Beispiel für eine User Story*

  Als Student will ich mich online zu einer Vorlesung anmelden, sodass ich einen Platz in der Übung erhalte.

- Beispiele für dazu passende Akzeptanzkriterien
  - Wenn sich ein Student einer anderen Fakultät anmeldet, muss eine Fehlermeldung erscheinen.
  - Wenn der Studierende bereits angemeldet ist, muss eine Fehlermeldung erscheinen.
  - Wenn bereits alle Plätze in einer Vorlesung mit begrenzter Teilnehmerzahl vergeben sind, kann sich der Studierende nicht mehr anmelden.
  - Der Studierende kann sich bei einer Vorlesung mit begrenzter Teilnehmerzahl anmelden, wenn noch nicht alle Plätze vergeben sind.

Gute User Stories sind **INVEST**

I ndependent:	möglichst unabhängig voneinander
N egotiable:	zerlegbar, komponierbar, änderbar, verhandelbar
V aluable:	hat wirtschaftlichen Wert
E stimatable:	so klar, dass es vom Team geschätzt werden kann
S mall:	klein genug, um in einem Sprint entwickelt werden zu können
T estable:	Klare Akzeptanzkriterien

Vgl. Cohn: User Stories Applied, Addison Wesley 2004

Independent - Details

- Zwei User Stories sind abhängig:
  - *„Als Personalsuchender will ich eine Stellenanzeige im Hochschul-Informationssystem mit einer Visa Karte bezahlen, um …“* und
  - *„Als Personalsuchender will ich eine Stellenanzeige im Hochschul-Informationssystem auf Rechnung bezahlen, um …“*
- Lösung: anderer Schnitt der Stories
  - *„Als Personalsuchender will ich die Zahlungsart für eine Stellenanzeige im Hochschul-Informationssystem wählen können, um …“* sowie
  - *„Als Personalsuchender will ich die Zahlung auf Rechnung durchführen, um …“* und
  - *„Als Personalsuchender will ich die Zahlung per Kreditkarte durchführen, um …“*.

Valuable - Details

- Geschäftswert auch durch Einhaltung von Standards
  - *„Als Team will ich Dokumentation nach Norm ISO 9001 erzeugen, damit die Firma das ISO 9001 Zertifikat erhält.“*
- Gegenbeispiel
  - *„Als Entwickler will ich mit dem Lehrevaluationssystem über SQL.NET auf eine Oracle-Datenbank zugreifen, damit ich die Studenten lesen kann“.*

Schrittweiser Ausbau von User Stories
Straßenbau Metapher

- **Feldweg**: Funktion ist mit einiger manueller Arbeit und Workarounds verfügbar
- **Pflastersteinstraße**: Einfache, minimale Implementierung der Funktion
- **Asphaltierte Straße**: Implementierung mit Sonderfällen und Zusatzfeatures.
- **Autobahn**: Vollausgebaute Implementierung mit allen Sonderfällen, Konfigurierbarkeit und Komfortfunktionen.

Schrittweiser Ausbau von User Stories
Straßenbau Metapher

- **Feldweg**: Als Buchhalter will ich die Rechnungsdaten aus dem System kopieren und eine Rechnung selber mit einer Textverarbeitung erstellen und versenden, um die geleisteten Arbeiten abzurechnen.
- **Pflasterstraße**: Als Buchhalter will ich eine minimale Rechnung (Rechnungsdatum, Kunde, MWSt, Gesamtsumme) automatisch erstellen und selbst versenden, um …
- **Asphaltierte Straße**: Als Buchhalter will ich eine ausführliche Rechnung mit allen Rechnungspositionen und Teileinformationen automatisch erstellen und selbst versenden, um …
- **Autobahn**: Als Buchhalter will ich eine ausführliche und für den Kunden personalisierte Rechnung mit allen Zusatzinformationen und Werbung automatisch an den Kunden versenden, um …

Schneiden von User Stories

- Konzepte für das Aufteilen von User Stories in kleinere Einheiten:
  - Nach Daten
  - Nach Usern / Rollen
  - Nach Schritten im Prozess
- Beispiel aus Campus-Informationssystem

„Als Benutzer will ich Informationen zu Vorlesungen verwalten können, um ...“

*) Weitere Verfahren und mehr Details in Leffingwell: Agile Software Requirements , Addison-Wesley, 2011

Schneiden von User Stories
Schnitt nach Daten

„Als Benutzer will ich Informationen zu Vorlesungen verwalten können, um …“

- Als Benutzer will ich ***Titel und Beschreibung*** einer Vorlesung verwalten können, um …
- Als Benutzer will ich ***Skripte und Übungsblätter*** zu einer Vorlesung verwalten, um …
- Als Benutzer will ich die ***Teilnehmer einer Vorlesung*** verwalten können, um ….
- Als Benutzer will ich ***Termine*** zu einer Vorlesung verwalten können, um ….

Schneiden von User Stories
Schnitt nach Schritten im Workflow, z.B. CRUD

„Als Benutzer will ich Informationen zu Vorlesungen verwalten können, um …“

- Als Benutzer will ich Informationen zu einer Vorlesung ***anlegen*** können, um …
- Als Benutzer will ich (neue) Informationen zu einer Vorlesung ***validieren*** können, um …
- Als Benutzer will ich Informationen zu einer Vorlesung ***suchen*** können, um …
- Als Benutzer will ich Informationen zu einer Vorlesung ***löschen*** können, um ...

Schneiden von User Stories
Schnitt nach Benutzer Rollen / Personas

„Als Benutzer will ich Informationen zu Vorlesungen verwalten können, um …“

- Als ***Professor*** will ich Informationen zu einer Vorlesung verwalten können (= Skripte, Termine), um …
- Als ***Student*** will ich Informationen zu einer Vorlesung verwalten können (= Notizen, Nachrichten, Klausuranmeldung, Evaluation, …), um …
- Als ***Prüfungsamt*** will ich Informationen zu einer Vorlesung verwalten können (= Klausurtermin, Raum für die Klausur, …), um …

Personas in User Stories

- Idee: **Persona** ist der User (As a …) in der User Story

- **Persona** = Idee aus dem Usability Engineering
  (Cooper: „The Inmates are running the asylum“, SAMS, 2004)
- = prototypische Benutzer
  - Modelliert Ziele und Verhaltensweisen einer bestimmten Benutzergruppe
  - Sowie gemeinsame „demografische“ Eigenschaften (Alter, Erziehung, …)
- Ziel: System an die **tatsächlichen Bedürfnisse** einer Benutzergruppe anpassen
  - 75 Jähriger Rentner hat andere Bedürfnisse wie ein 6 Jahre altes Kind oder ein 40 Jahre alter Manager
  - Erfahrener Benutzer vs. Gelegenheitsbenutzer
- Aufwendiger Prozess zum Finden/Schneiden von Personas, über Interviews und Beobachtung

Grobteilige User Stories

- = **Epics (Epen)**
  - Sehr grobe, allgemeine User Stories
  - Kaum Schätzbar (> 13 Story Points)
  - Nicht in einem Sprint umsetzbar
  - Umsetzung noch in entfernter Zukunft
- = **Themes (Themen)**
  - Strukturieren den Product Backlog
  - Sehr grobe Epen, eher Überschriften
  - Auch verwendet zur Release Planung

User Stories: Eigenschaften

- Vorteile
  - In der **Sprache der Nutzer**, der Kunden
  - Fokussiert auf konkreten **Mehrwert für Kunden**
  - **Leichtgewichtig**, leicht teilbar, greifbar (Karteikarte)
  - Kein Formalismus notwendig
  - Werkzeug für Planung, Kommunikation mit dem Kunden
- Nachteile
  - Fehlende Details: z.B. Um Konzernbilanzierung zu beschreiben, muss man wissen wie Bilanzierung funktioniert
  - Kunde muss für Detailinformationen **verfügbar**, **kompetent** und **entscheidungsbefugt** sein
  - Keine Dokumentation der Anforderungen, z.B. für das Wartungs-Team, das die Software irgendwann übernimmt

User Storys: Kritik

- **Viele Garantien des Requirements Management gehen verloren**
  - Nachverfolgbarkeit von Anforderung zu Code zu Testfall fehlt
  - Wartungsdokumentation fehlt (was die Software tut steht höchstens im Code, ist aber nicht zwingend dokumentiert)
  - User Stories sind nicht im Team Review-fähig, da Details fehlen (ein persönliches Gespräch kann man nicht reviewen)
- Blick fürs ganze geht evtl. verloren
  vgl. Cockburn Bild: „Elefanten Carpaccio“
  - Wichtig: Überblick über alle Anforderungen verschaffen
  - Wichtig: Überblick fachlich dokumentieren (< 10 Seiten)
- **User Storys richtig zu definieren ist schwieriger, als es aussieht**, auch wegen der „Schneide-Techniken“
- Alternative: Use Cases

Akzeptanztests einmal anders …

- Ideen
  - Requirements Engineer, Entwickler und Tester (**Three Amigos**) spezifizieren gemeinsam (-> weniger Missverständnisse)
  - Und zwar so: **Akzeptanzkriterien als Beispiele formulieren** (= Konkrete Daten, konkretes Vorgehen, exakt definiert)
  - Beispiel ist Spezifikation und automatisierter Testfall in einem
  - Damit: Beispiel = Testfall = Spezifikation -> „**Living Documentation**“
- Techniken
  - Behaviour Driven Development (BDD)
  - Acceptance Test Driven Development (A-TDD)
  - Specification By Example

Akzeptanztests einmal anders …
Living Documentation

- Kunde vor Ort muss Systemverhalten festlegen (im Rahmen der Anforderungsanalyse)
  - Als Anforderung
  - Als Akzeptanzkriterium / Akzeptanztest
- Beispiele des Systemverhaltens sind dafür besonders geeignet
  - Beide Seiten entwickeln eine konkrete Vorstellung
  - Risiko von Missverständnissen (wg. Begriffsdefinitionen, wg. Abstraktion) sinkt
- Nutzung der Beispiele als Spezifikation und für autom.Tests (= Ausführbare Spezifikation)
  - Spezifikation ist zwingend aktuell (sonst scheitern die Akzeptanztests)
  - Erzwingt Präzision (sonst nicht automatisierbar)
  - Gemeinsame Sprache der Tester, Entwickler und Kunden
  - Erleichtert die Pflege, da Beispiele im Code stehen
- Damit: Testfall == Dokumentation == Spezifikation

Akzeptanztests einmal anders … Behaviour Driven Development

- = Weiterentwicklung des Test Driven Development
- = Formulierung von Akzeptanzkriterien zur Test -Automatisierung
- = Ausführbare Spezifikation

User Story (Formuliert durch den „Kunden“)

```
As a        <Rolle>
I want      <Feature>
So that     <Geschäftswert erreicht wird>
```

Szenario (Formuliert durch den „Kunden“)

```
Given       <Voraussetzungen>
When        <Feature wird ausgeführt>
Then        <Erwartetes Ergebnis>
```

Akzeptanztests einmal anders …
BDD Beispiel

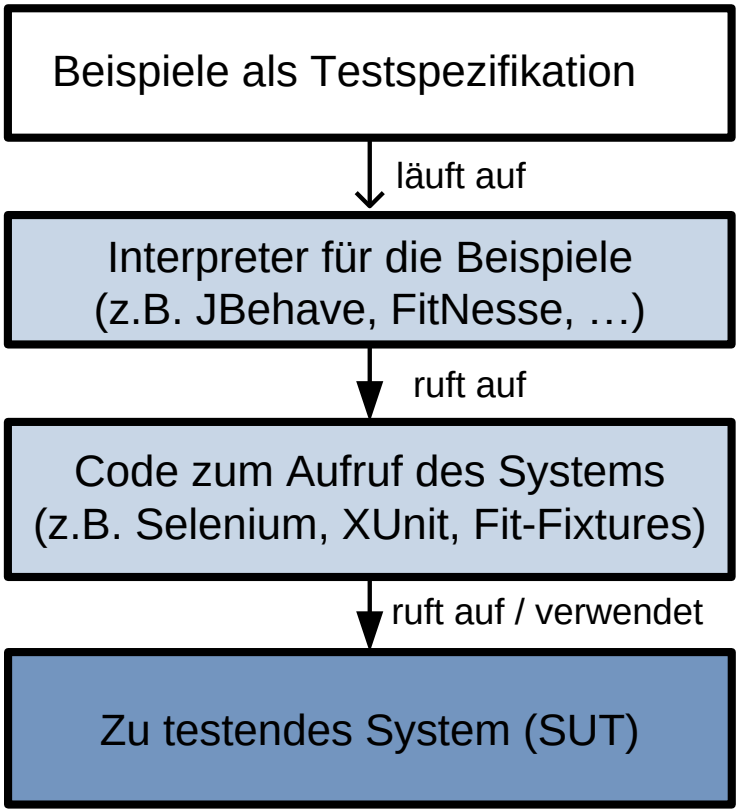
- Verhaltensbeschreibung - Allgemein

```
Given <Initialisierung, Testvoraussetzung>
When <ein Ereignis tritt ein>
Then <sichere ein bestimmtes Ergebnis zu>
```

- Ausprägung für einen Geldautomaten

Scenario: Konto hat Guthaben
Given das Konto hat Guthaben größer als 300 Euro
And die Karte ist valide
And der Automat hat Bargeld mehr als 300 Euro
When der Kunde fordert 300 Euro an
Then sichere zu das Konto ist mit 300 Euro belastet
And sichere zu 300 Euro sind ausgegeben
And sichere zu die Karte ist ausgegeben

BDD zur Testautomatisierung
Testspezifikation = Given..When..Then



Themen

- Was bedeutet agil?
- Anforderungsermittlung in agilen Projekten
- Funktionale Anforderungen mit User Stories und Spezifikation durch Beispiele
- Nichtfunktionale Anforderungen als Constraints
- **Scrum als agiles Framework und Anforderungsverwaltung: Product Backlog**

Scrum
Ein agiles Prozess Framework

- Erfinder = Sutherland, Schwaber ca. 1995
- Scrum = Figur im Rugby: Planung nächster Spielzug
- Scrum = **Prozess Framework** / Rahmenprozess
  - Team organisiert sich während Iteration selbst
  - Iteration = ca. **7 - 30 Tage = Sprint**
  - **Details zur Entwicklung nicht vorgegeben**, lediglich Organisationsrahmen, daher kombinierbar mit XP
  - Anforderungen **beispielsweise** als User Stories, diese werden priorisiert und verfolgt
- **Beliebteste Agile Methode** auch unter Managern
- Literatur: http://www.scrum.org/Scrum-Guides

Rollen in Scrum: **Product Owner**

- **= verantwortlich für das Produkt**
  - = Wirtschaftlicher Erfolg des Produktes
- Produktkonzept
  - Schnittstelle zu dem/den Kunden / Stakeholdern
  - Erstellt Anforderungen
  - Verfügbar für Fragen des Entwicklungsteams
  - Priorisiert Anforderungen aus Backlog, definiert zusammen mit Entwicklungsteam Sprint Inhalte
  - Nimmt implementierte Anforderungen ab (Akzeptanztest)
- Misst Produkt-Fortschritt
- Erstellt Release Plan für das Produkt

Rollen in Scrum: **Scrum Master**

- = **Verantwortlich für Prozess**
- Ziel: Selbst organisierenden Prozess / Team erzeugen
- Überwacht und kontrolliert Scrum Prozess
  - Erkennt und klärt Probleme
  - Bildet Team in Scrum aus
  - Fordert die Einhaltung der Regeln ein
- Prozessverbesserung über Retrospektiven
- Hilft dem Entwicklungsteam, Probleme zu lösen

Rollen in Scrum: **Entwicklungsteam**

- Liefert fertiges Produkt an Product Owner
- = sichert zu, die ausgewählten Anforderungen in passender Qualität zu liefern
- **Selbstorganisierend**
  - Teammitglied sucht sich seine Aufgaben (Pull-Prinzip) = Analyse, Design, Coding, Test, Dokumentation
  - **Teammitglieder bestimmen Vorgehen selbst**
  - Jeden Tag Abstimmung: **Daily Scrum**

Artefakte in Scrum: **Product Backlog**

Das Product Backlog ist eine **geordnete Liste mit allem, was in dem Produkt benötigt werden könnte**. Es ist die **einzige Quelle** von Anforderungen für jedwede Änderungen an dem Produkt. Der Product Owner ist für das Product Backlog verantwortlich, inklusive dessen Inhalte, Bereitstellung und Reihenfolge.

Sutherland/Schwaber

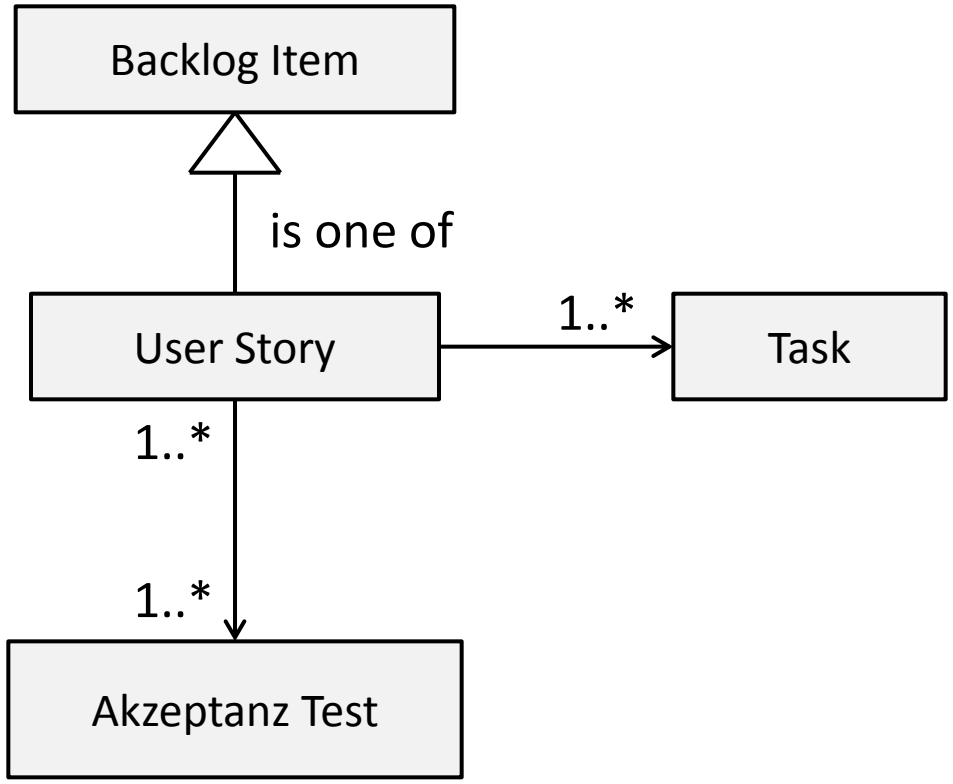
Inhalte:

- Anforderungen (z.B. als User Story)
- Unkritische Fehler (Kritische wandern in den Sprint Backlog)
- Forschungsaufträge (Spikes)

Product Backlog - Beispiel

Id.	Beschreibung	Priorität	Aufwand
Release 1.0: Minimale Evaluationssoftware nur für Vorlesungen			
…	…	…	…
Sprint #3: Ziel: Evaluationsbogen ausfüllen			
2	Als Student will ich einen Evaluationsbogen absenden können, um mein Feedback dem Prof. mitzuteilen	Hoch	3
4	Als Student will ich eine Ja/Nein Frage im Evaluationsbogen beantworten können, um den Evaluationsbogen auszufüllen.	Hoch	8
17	Als Student will ich eine Freitext- Frage im Evaluationsbogen beantworten können, um den Evaluationsbogen auszufüllen.	Hoch	5
23	Als Student will ich eine Auswahl-Frage mit bis zu 7 möglichen Antworten im Evaluationsbogen beantworten können, um den Evaluationsbogen auszufüllen.	Hoch	2
…	…	…	…

User Storys im Product Backlog Zusammenhänge



User Storys in Scrum

- User Story ist im Backlog (neben Spikes und Bugs)
- Mehrere Akzeptanztests für jede Story (häufig auf Rückseite der Karte)
- Sprintplanung (= Sprint Backlog): Story wird in Tasks zerlegt

*) Grafik aus: Leffingwell: Agile Software Requirements , Addison-Wesley, 2011

Ein guter Backlog ist **DEEP**

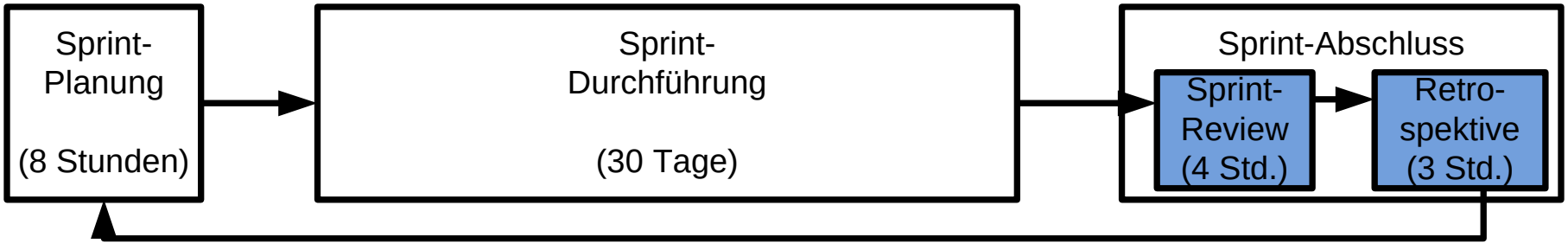
- **D**etailed Apropriately (angemessen Detailliert)
  - Direkt vor dem Sprint so detailliert, dass umsetzbar
  - Sonst: Eher vage als grobe UserStory oder Epic
- **E**stimated (Geschätzt)
  - Team schätzt Aufwand zur Umsetzung einer UserStory
  - Schätzung dient als Grundlage zur Sprintplanung
- **E**mergent (Entwickelt sich, wird gepflegt)
  - Team pflegt zusammen mit dem PO regelmäßig den Product Backlog (= ***GROOMING***)
  - Aufwand: ca. 10% des Projektbudgets
- **P**rioritized (Priorisiert)
  - Jede UserStory muss Priorität haben, um zu entscheiden, ob diese in den nächsten Sprint soll (= weiter detailliert werden muss, …)

Vgl. Roman Pichler:
Agiles Produktmanagement
mit Scrum, Addison-Wesley, 2010

Artefakte in Scrum: **Sprint Backlog (= Plan)**

- **= Liste der Aufgaben in einem Sprint**
- Inhalte
  - Anforderungen für den nächsten Sprint
  - Tasks für das Entwicklungsteam (->Anforderungen)
  - Fehler die Schnell behoben werden müssen
- Guter Sprint Backlog
  - = Öffentlich Sichtbar (Information Radiator)
  - Enthält **alle** Tasks für den nächsten Sprint

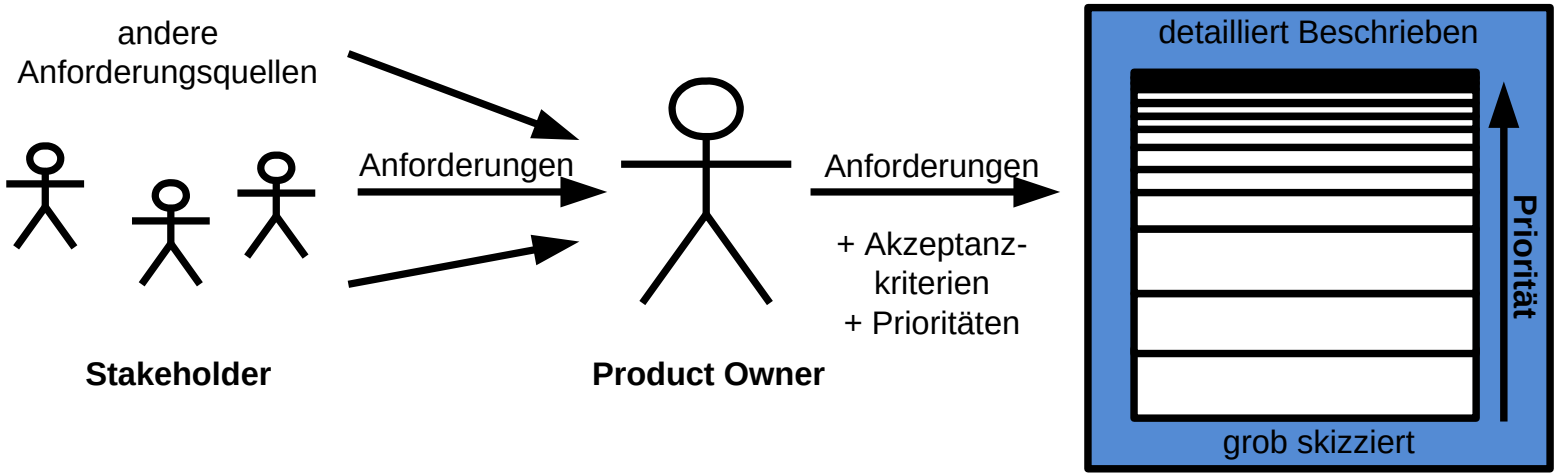
Phasen in Scrum



- **Timeboxed**: Abhängig von der Länge der Sprint-Durchführung verkürzen sich alle anderen Meetings / Phasen proportional
- Bei 30 Tagen: 8 Stunden Sprintplanung und ca. 7 Stunden Sprint Abschluss,
- Bei kürzeren Sprints: verkürzen sich auch die Meetings

Vor dem Sprint
Anforderungs**analyse**

- Product Owner **sammelt Anforderungen**
- PO **priorisiert** Anforderungen
- Wichtige Anforderungen werde **detailliert** (PO+Team)
- Entwicklungsteam stellt Verständnis-Fragen, **hilft** PO
- Entwicklungsteam **schätzt Aufwand** für Umsetzung
- Pflege des Product Backlog geschieht fortlaufend (Grooming)



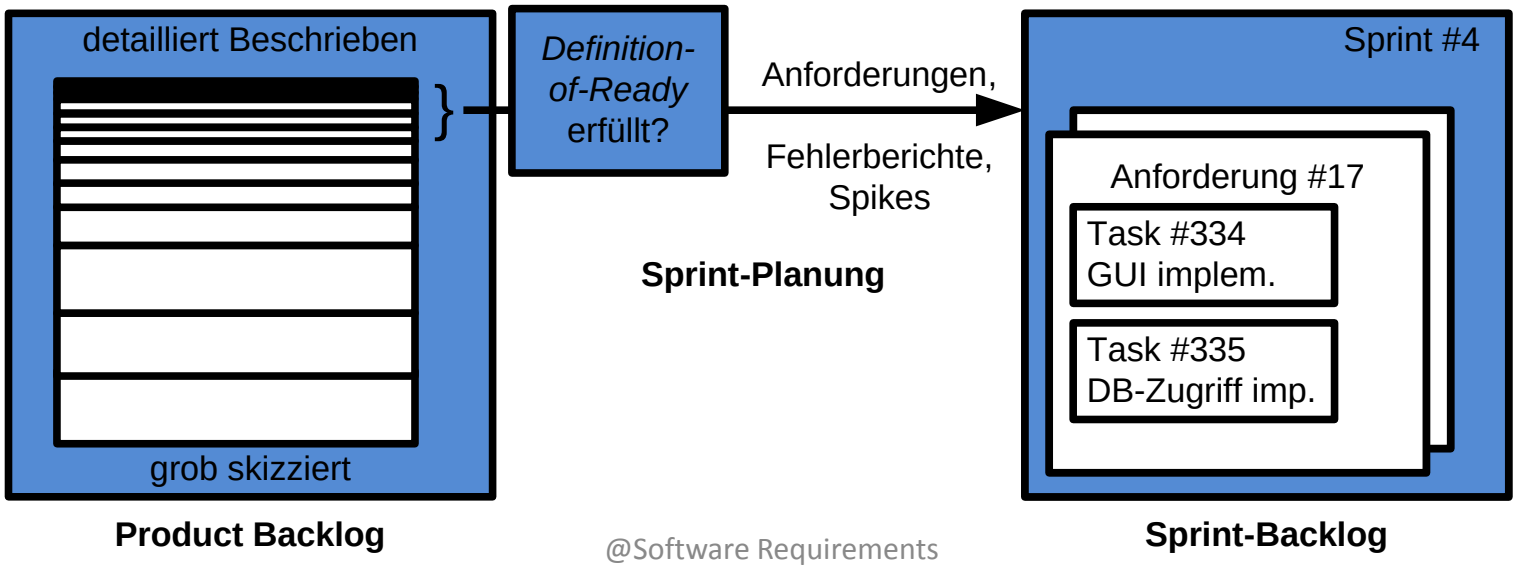
Sprint-Planung (4 + 4 Stunden)

1. Hälfte (4 Stunden)

- PO stellt Sprint-Ziel und wichtige Anforderungen vor
- Entwicklungsteam wählt Anforderungen zusammen mit PO

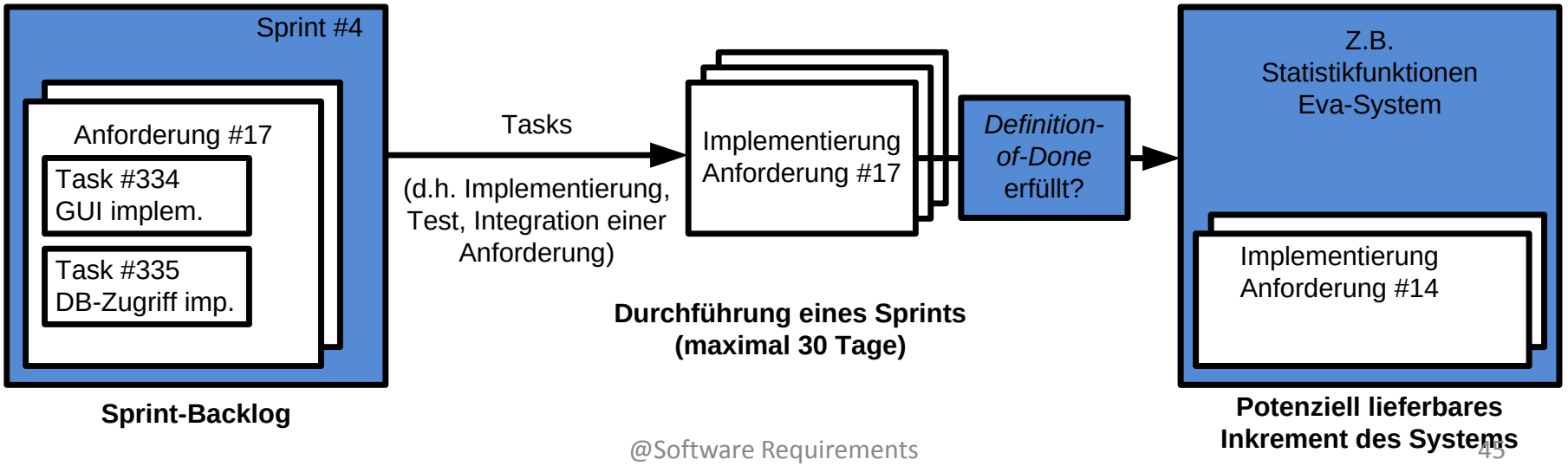
2. Hälfte (4 Stunden)

- Entwicklungsteam definiert für jede Anforderung Tasks für Sprint Backlog
- D.h. Team plant die Tasks für den Sprint



Sprint Durchführung

- Entwicklungsteam arbeitet Tasks aus Sprint Backlog ab
- PO nimmt „fertige“ Anforderungen ab
  (= Definition of Done erfüllt)
- Fertige Anforderungen werden auf Burn-Down Chart vermerkt (zur Fortschrittskontrolle)



Definition of Ready und *Definition of Done*

- Beide „Definitions“ im Team (PO, SM, Entw.) festgelegt
- **Definition of Ready**
  - = Anforderung so gut verstanden und detailliert, dass sie eingeplant werden kann
  - Eigenschaften = Detailliert genug, dass sie im nächsten Sprint umgesetzt werden kann, Hohe Priorität, Abgeschätzt
- **Definition of Done**
  - = Anforderung ist vollständig umgesetzt
  - Eigenschaften = Integriert, Getestet, Dokumentiert (!), Code-Review bestanden, Akzeptanztests bestanden, …

Sprint Durchführung

- Teammitglieder
  - Bestimmen selbst was sie tun
    = Auswahl der Tasks / User Storys
  - Bestimmen selbst Methodik / Vorgehen
- Sprint Backlog wird erweitert
  - Neue Tasks entstehen durch besseres Problemverständnis
  - Entsprechende Tasks werden ergänzt
- Täglich: Daily Scrum – Meeting
  - = Fortschrittskontrolle (wie viele Story Points sind fertig?)
  - = Synchronisation im Team
  - = Beheben von (org.) Problemen durch den Scrum Master

Daily Scrum

Hier Bild:
http://www.martinfowler.com/articles/itsNotJustStandingUp.html

- Entwicklungsteam + Scrum Master + ggf. PO trifft sich ***morgens*** (z.B. 09:00), ***Maximal 15 Minuten***
- Jeder hat 2-5 Minuten und stellt Arbeitsfortschritt und Probleme dar
  - **Was habe ich gestern getan?**
  - **Was werde ich heute tun?**
  - **Was hat mich aufgehalten?**
- Im Stehen, ohne Meeting-Kekse oder Kaffee. Alternative: **Daily Skype**
- Ziele
  - Tägliche Koordination des Teams: „Selbstorganisierendes Team“
  - Schnelles beseitigen von Problemen
  - „Gruppendruck“: Übernommene Aufgaben tatsächlich fertigstellen
  - Team-Building

Details:
http://www.martinfowler.com/articles/itsNotJustStandingUp.html

Sprint-Abschluss

- **Sprint-Review Meeting**
  - Öffentliche Veranstaltung, auch für Kunden
  - Fertige Anforderungen werden dem Product Owner vorgeführt
  - Eventuell: Neue Anforderungen
- **Sprint Retrospektive**
  - Wichtig: Kontinuierliche Verbesserung des Prozesses
  - Gutes beibehalten
  - Probleme beheben und Prozess anpassen
  - Klärung der Verantwortlichkeiten

Literatur

- Gojko Adzic: Specification By Example, Manning, 2011
- Mike Cohn: User Stories Applied: For Agile Software Development, Addison Wesley, 2004
- Mike Cohn: Agile Estimating and Planning, Prentice Hall, 2005
- Dean Leffingwell: Agile Software Requirements , Addison-Wesley, 2011
- Roman Pichler: Scrum, dpunkt, 2007
- Roman Pichler: Agiles Produktmanagement mit Scrum, Addison-Wesley, 2010
- Ken Schaber, Mike Beedle: Agile Software Development with Scrum, Prentice Hall, 2002 (erstes Buch zu Scrum)