EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT

Eingebettete Systeme

Agile Softwareentwicklung mit Scrum

Eine praxisnahe Einführung

04.05.2016, Sebastian Burg

Gliederung

- Einführung
- Agile Entwicklung
- Scrum
  - Stories
  - Komponenten
  - Rollen
  - Zyklen

Warum setzen wir Methoden der agilen Softwareentwicklung ein?

EINFÜHRUNG

Klassische Vorgehensmodelle zur Softwareentwicklung

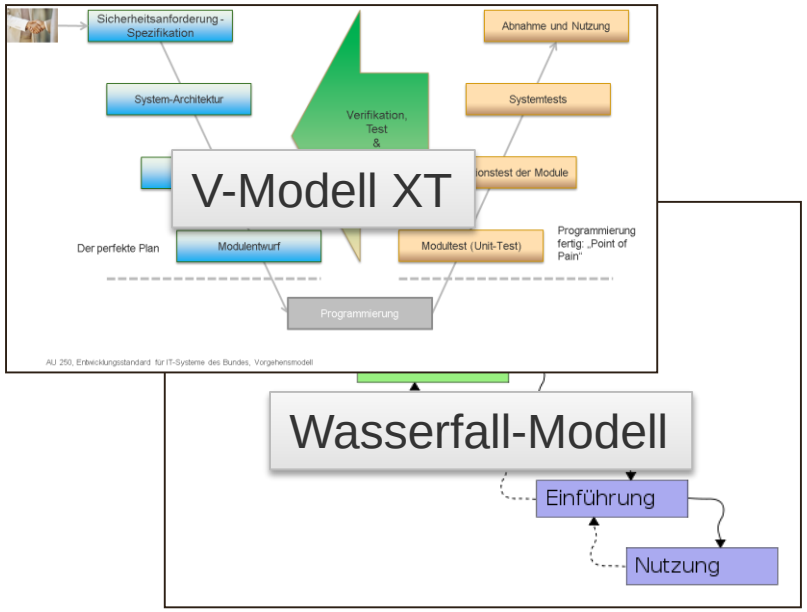
Chaos,
One-Man-Show

- Schnell erste Ergebnisse
- Kaum Weiterbildung des Entwicklers
- Kein Wissenstransfer zwischen den einzelnen (unabhängig arbeitenden) Entwicklern
- Keine Qualitätskontrolle
- „Bin ich auf dem richtigen Weg?“
- Kommen die richtigen Technologien zum Einsatz?
- Lange Zeit bis zur Fertigstellung des Produkts

Klassische Vorgehensmodelle zur Softwareentwicklung

- Definierte Schnittstellen zwischen den Entwicklern
- Qualitätskontrolle gut durchführbar
- Vollständiger Prozess selten von vorn bis hinten planbar
- „Was schief gehen kann, wird schief gehen“, Murphys Gesetz
- Reaktion auf ungeplante Ereignisse träge
- Hoher Verwaltungsaufwand

Plan-getriebene Entwicklung



Klassische Vorgehensmodelle zur Softwareentwicklung

Chaos,
One-Man-Show

Plan-getriebene
Entwicklung

Sowohl One-Man-Show als auch Plan-getriebene Entwicklung sind für die (relativ) kleinen Programmierprojekte ungünstig.

Keine Qualitätskontrolle, wenig Lerneffekt
vs.
Großer Aufwand, träge Reaktionsfähigkeit, geringe Motivation

Klassische Vorgehensmodelle zur Softwareentwicklung

Chaos,
One-Man-Show

Plan-getriebene
Entwicklung

Agile Softwareentwickung

AGILE ENTWICKLUNG

Was bedeutet eigentlich agil?

- Agil sein bedeutet, dass jemand körperlich oder geistig wendig und/oder flink ist
- Bezug auf Softwareentwicklung:
  - Angepasst (wendig) vorgehen: Flexibel Mittel wählen, mit denen Ziel erreicht werden kann
  - Schnell (flink) vorzeigbare Ergebnisse erzielen: Oft und schnell Laufende Software ausliefern

Das agile Manifest

- Unterzeichnet 2001 von vielen Vertretern aus der Software-Branche:
  - Ken Schwaber
  - Jeff Sutherland
  - Mike Beedle
  - u.v.a.
- Unter anderem von den Erfindern von Scrum, Crystal, Feature Driven Development & eXtreme Programming

Das agile Manifest: Einleitung

- 4 Säulen, die durch folgenden Satz eingeleitet werden:

Wir entdecken bessere Wege, Software zu entwickeln, indem wir Software entwickeln und anderen dabei helfen.

Das agile Manifest: Die 4 Säulen

Durch diese Arbeit bewerten wir

Individuen und Interaktionen höher als Prozesse und Werkzeuge,

Laufende Software höher als ausgedehnte Dokumentation,

Zusammenarbeit mit dem Kunden höher als Vertragsverhandlungen,

Reaktion auf Veränderung höher als Planverfolgung.

Agile Werte

- **Rückkopplung**
  - Kommunikation zwischen Teammitgliedern
  - Feedback vom Kunden durch frühe Softwareauslieferung
- **Mut**
  - Offenheit und Ehrlichkeit (besonders bei der Kommunikation von negativen Dingen)
- **Respekt**
  - Nur wer sich respektiert fühlt kann offen sprechen

Ziele agiler Entwicklung

schnell lauffähiges System erstellen

Sicherheitsanforderung - Spezifikation

System-Architektur

Systementwurf

Modulentwurf

Programmierung

Modultest (Unit-Test)

Integrationstest der Module

Systemtests

Abnahme und Nutzung

Verifikation, Test & Validierung

verkürzen

Programmierung fertig: „Point of Pain“ verhindern

Weiterbildung der Entwickler (vgl. 1-Personen-Projekt)

AU 250, Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell

Agile Methoden

- Unter einer agilen Methode verstehet man eine konkrete benannte Zusammenstellung von agilen Praktiken
  - Crystal
  - FDD
  - eXtreme Programming
  - Scrum

Scrum als Methodik zur Umsetzung der agilen Softwareentwicklung

SCRUM

Scrum: Gedränge

- Scrum: Spielkonstellation aus dem Rugby-Sport

Scrum

- Keine Entwicklungsmethode im eigentlichen Sinne, sondern Managementrahmen für beliebige Projekte
- Einfach zu erlernen
- Prozessablauf grob festgelegt, aber bietet genügend Spielraum zur Reaktion auf ungeplante Veränderungen
- Scrum setzt sich zusammen aus:
  - Rollen
  - Komponenten
  - Meetings und Zyklen

- Anforderungen an das Produkt werden in Form von **Stories** festgehalten

Beedle, M; Schwaber, K.: Agile Software Development with Scrum. Prentice Hall, 2001
Pichler, R.: Agiles Projektmanagement erfolgreich einsetzen. Heidelberg, dpunkt.verlag, 2008
http://www.scrumalliance.com

STORIES, AUFWANDSSCHÄTZUNG, PRODUKTIVITÄT

Was sind Stories (kurzer Überblick)?

- Story: Anforderung
  - Aspekt, den eine Software erfüllen soll
  - Textuell oder formal aufgeschrieben
  - Story soll prüfbar sein (System soll schnell sein: **nicht prüfbar**; System soll innerhalb 1 sec antworten: **prüfbar**)

- Aus agiler Sicht: Schnell Software erstellen
  - Das impliziert: so wenig wie möglich aufschreiben
  - Gerade so genau, dass Aufwand geschätzt werden kann

- **Erspart das Lastenheft!**

Welches Medium? Welcher Inhalt?

- Gerne mittels Karteikarten (handschriftlich) oder in System
- Aufbau
  - Nummer, Datum, Beschreibung
  - Autor/Ansprechpartner, Schätzung, Priorität
  - Status:
    - RTD-F (Definition of Done)
    - ***Running***: Zugehörige Software(-Komponente) ist lauffähig
    - ***Tested***: Zugehörige Komponente hat Komponenten-Tests bestanden
    - ***Deployed***: Komponente hat Integrationstest bestanden und wurde ausgeliefert
    - Solange RTD nicht erfüllt ist, ist die Story in Bearbeitung

Aufwandsschätzung

- Das Schätzen von Aufwänden ist keine exakte Wissenschaft
- Nur relativ zu Rahmenbedingungen
- Möglichst festes Team
- Vergleich zu bereits umgesetzten Anforderungen aus anderen Projekten (“**Wetter von gestern**”)

- Die Aufwände zur Umsetzung werden vom Entwicklerteam erbracht
- Also schätzt dieses Team auch die Aufwände (**Verantwortung**)
- Unterschiedliche Entwickler sind unterschiedlich schnell: Teamaufwand wird geschätzt
- Alle Teammitglieder sind eingebunden (**demokratisches Schätzen**)

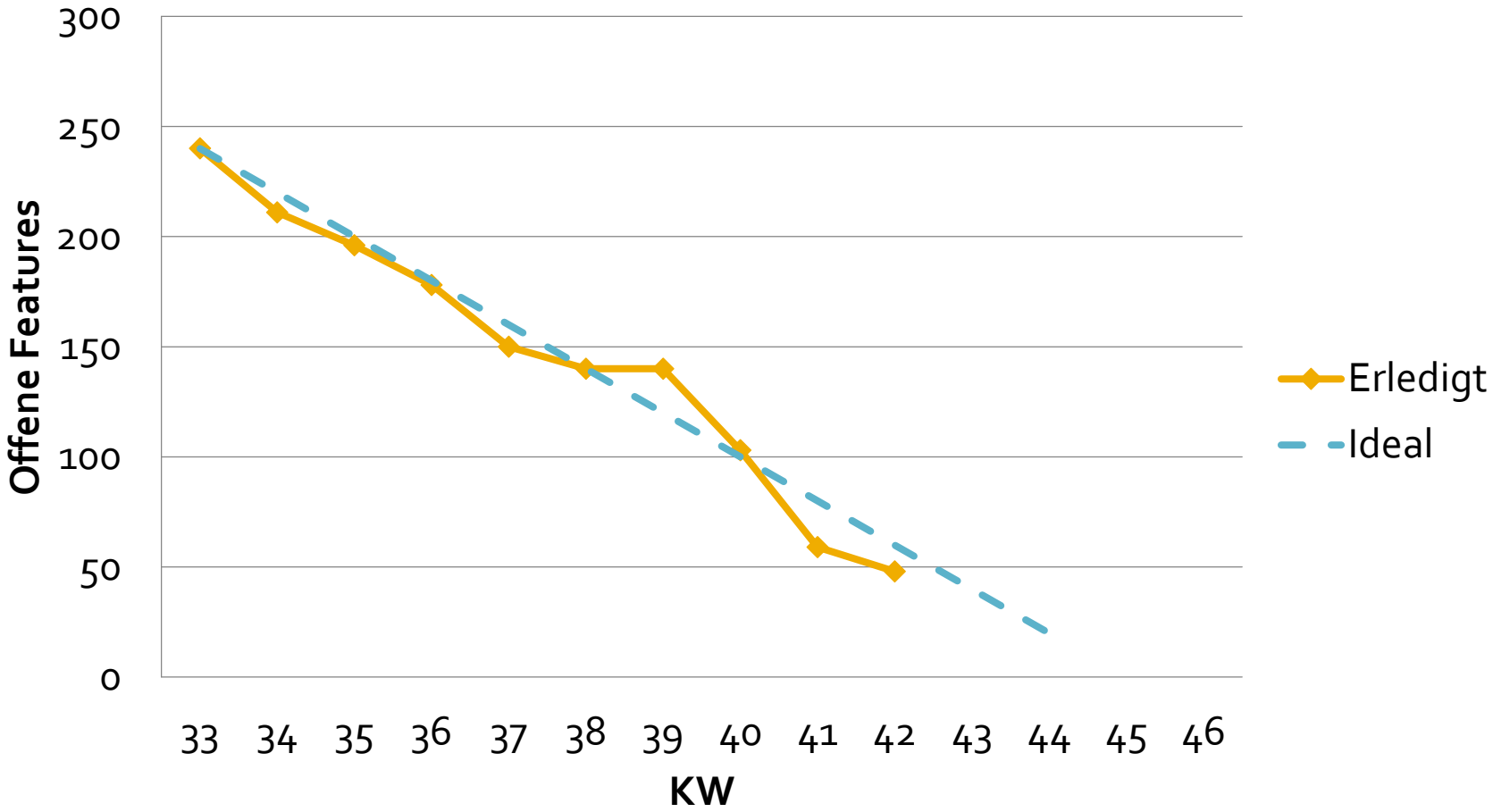
Schätzmaße

- Möglichst **abstraktes Schätzmaß**
  - Besser als konkretes Maß, da allgemeingültiger für gesamtes Team
  - Man weiß ja nicht wer was machen wird
- **Umrechnungsfaktor** von abstrakten Schätzmaß zu realem Aufwand bilden (**Erfahrungswert**)
- Wieviel (abstrakte) Einheiten werden pro Iteration abgearbeitet (**Erfahrungswert**)

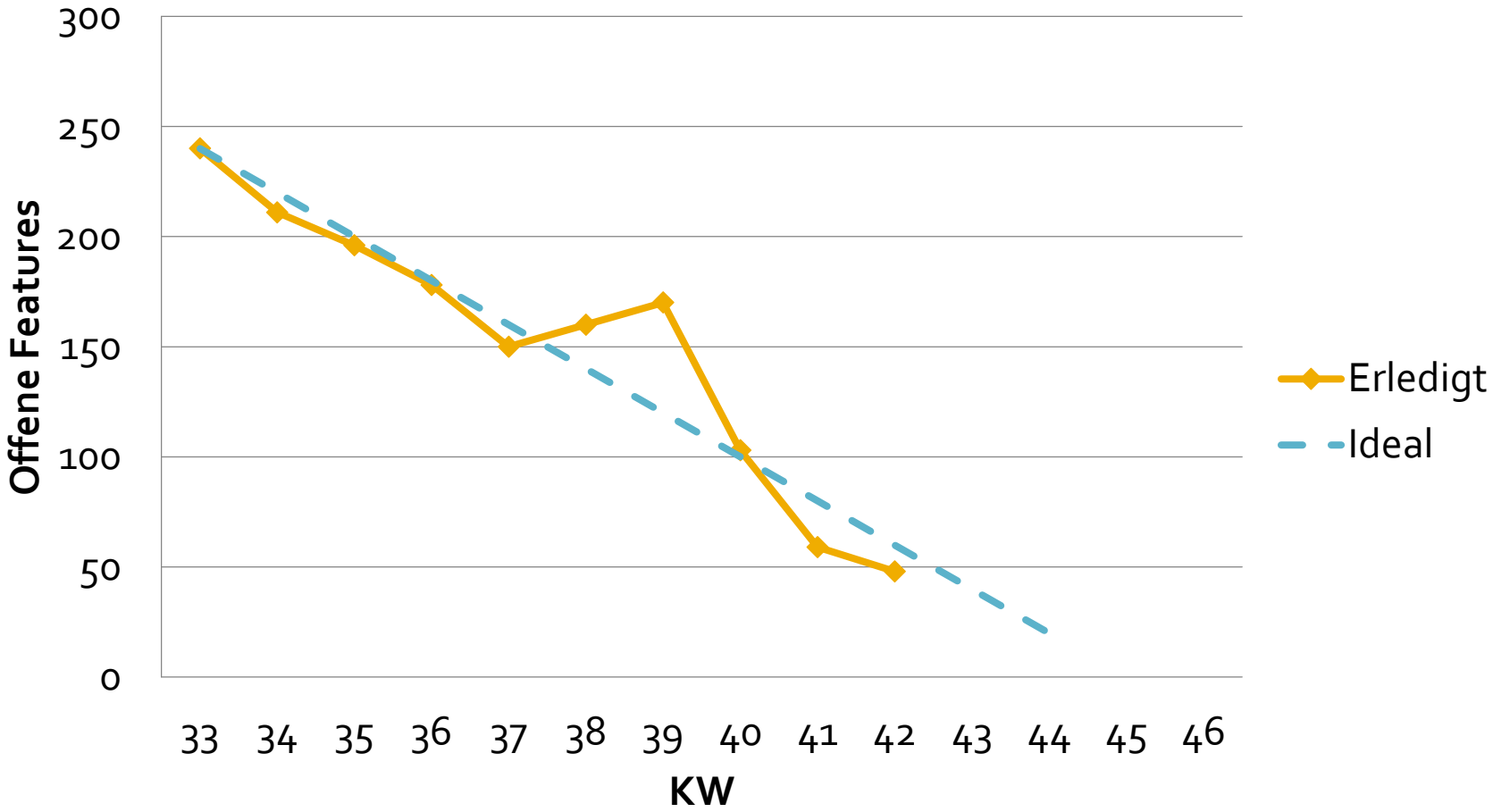
Sching-Schang-Schong-Schätzen

- Sching-Schang-Schong (Anlehnung an *Schere, Stein, Papier*)
- Per **Handzeichen** werden 1, 2, 3, 4, oder 5 relative Aufwandspunkte vergeben
- Bei großen Abweichungen: Diskussion
  - Technische Probleme?
  - Story zu groß? → Story teilen!

Feature-Burndown-Graph



Feature-Burndown-Graph



Scrum

- Keine Entwicklungsmethode im eigentlichen Sinne, sondern Managementrahmen für beliebige Projekte
- Einfach zu erlernen
- Prozessablauf grob festgelegt, aber bietet genügend Spielraum zur Reaktion auf ungeplante Veränderungen
- Scrum setzt sich zusammen aus:
  - Rollen
  - Komponenten
  - Meetings und Zyklen

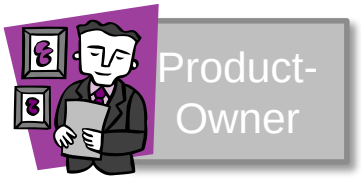
- Anforderungen an das Produkt werden in Form von **Stories** festgehalten

Beedle, M; Schwaber, K.: Agile Software Development with Scrum. Prentice Hall, 2001
Pichler, R.: Agiles Projektmanagement erfolgreich einsetzen. Heidelberg, dpunkt.verlag, 2008
http://www.scrumalliance.com

Beteiligte Personen am Scrum-Prozess

ROLLEN

Beteiligte Rollen



- Product Owner
  - Vertritt den Kunden und Anwender
  - Anforderungen in Form von Stories: Zuständig für
    - Formulierung
    - Auswahl
    - Priorisierung

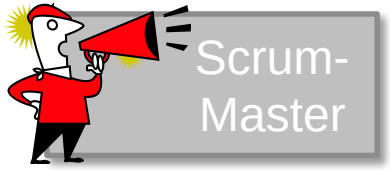
- **In den Programmierprojekten sind die Betreuer die Product Owner!**

Beteiligte Rollen

- Team
  - Softwareentwickler
  - Eigenverantwortlich für die Umsetzung der Anforderungen
- Handelt mit Product Owner aus, welche Anforderungen in welcher Zeit umgesetzt werden

- **In den Programmierprojekten bilden die Studierenden das Team!**

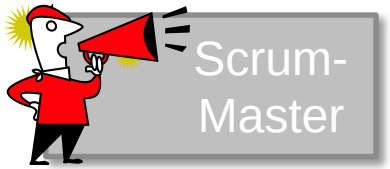
Beteiligte Rollen



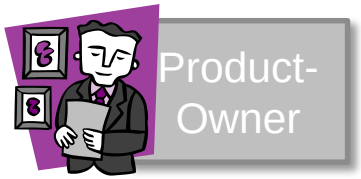
- Scrum-Master
  - Unterstützt Product Owner und Team bei Durchführung des Projekts
  - Achtet auf Einhaltung der Prozesse
  - Hilft bei konkreten Problemen
  - Hat eine Vermittlerrolle zwischen Team und Product Owner

- **In den Programmierprojekten sind die Scrum-Master 1 oder 2 Personen aus dem Team (wechselt nach jedem Sprint)!**

Beteiligte Rollen



1-2 Studierende aus Team (wechselnd)
Überwacht die Scrum-Prozesse und
vermitteln zwischen Team und Product Owner.
Stellt am Ende eines Sprints die Ergebnisse vor.



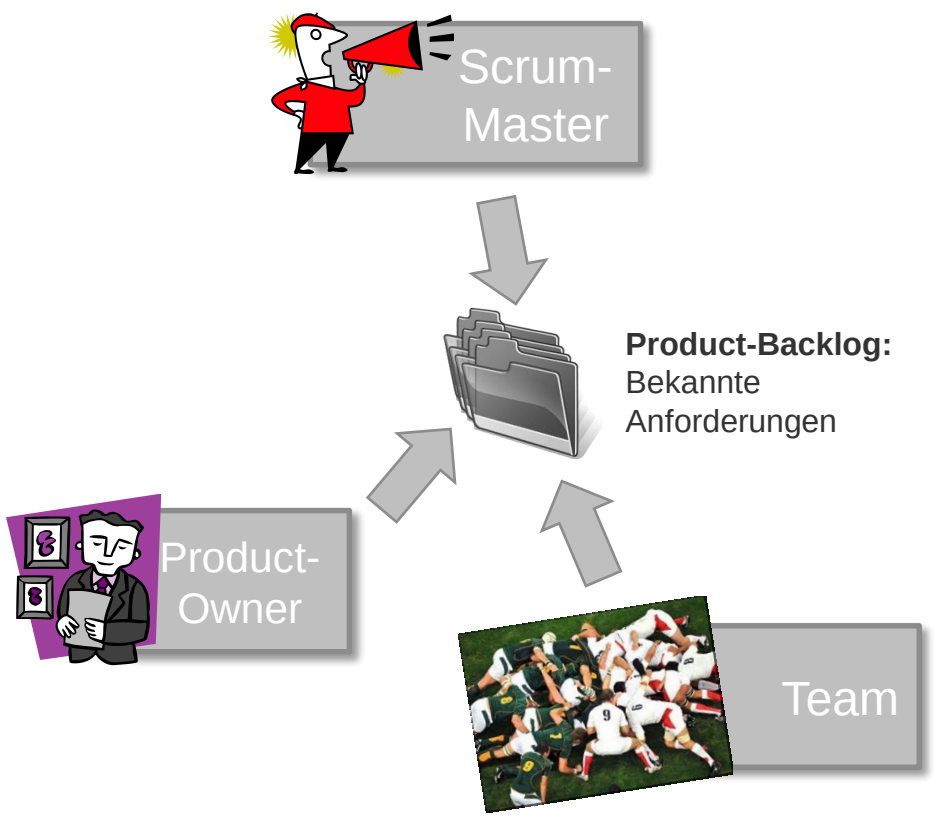
Betreuer
Definiert gemeinsam mit Team
die zu erledigenden Aufgaben im Sprint!

Studierende
Setzen die Aufgaben / Stories um.

KOMPONENTEN

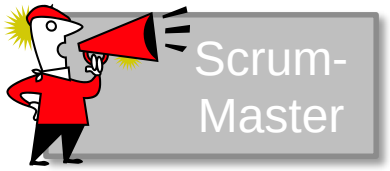
Projektkomponente: Product Backlog

- Enthält Stories
- Stories können jederzeit von allen Projektbeteiligten erstellt werden
- Sie werden im Product Backlog gesammelt
  - Geschätzte und ungeschätzte Stories
- Product Owner entscheidet wie er mit Product Backlog umgeht

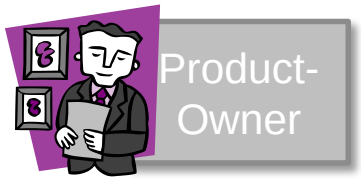


Projektkomponente: Sprint Backlog

- Enthält abzuarbeitende Stories für nächsten Zyklus
- Entwickler-Team ist an der Entscheidung beteiligt (Menge)
- Ausgewählte Stories werden im Sprint-Backlog abgelegt

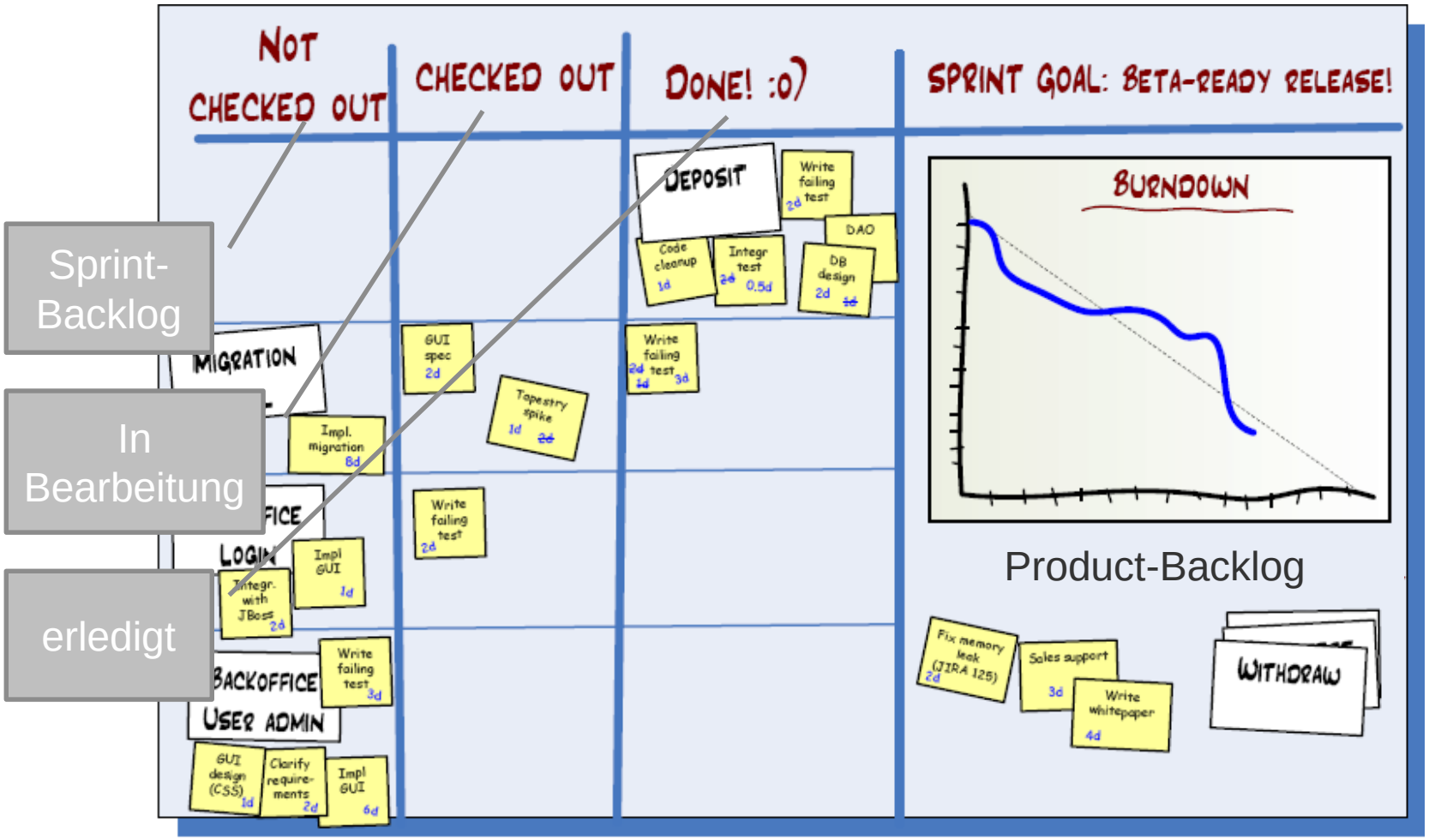


Sprint-Backlog:
Stories
für nächsten Zyklus



Product-Backlog:
Bekannte
Stories

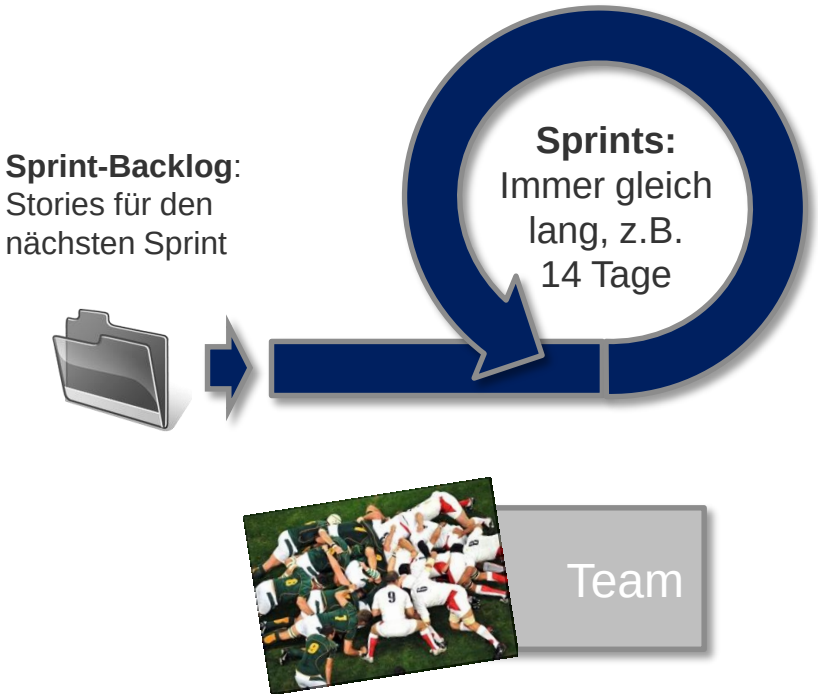
Taskboard



MEETINGS UND ZYKLEN

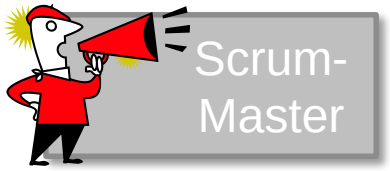
Zyklus: Sprint

- Feste Länge
- Team arbeitet Sprint-Backlog ab
- Ohne Einflüsse von außen
- Es werden keine Stories durch Product Owner oder Scrum Master ergänzt
- Rückfragen sind erlaubt
- Team arbeitet eigenverantwortlich
  - Keine Leitung durch Scrum-Master oder Product Owner



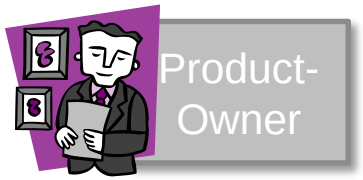
Meeting: Sprint-Planung (Beginn eines Sprints)

- Ziel: Auswahl der Stories für den nächsten Sprint
- Wird von Scrum Master organisiert
- Auswahl wird von Product Owner getroffen
- Team und Product-Owner treffen einer Vereinbarung über Menge an Stories (Commitment)



Sprint-Backlog: Anforderungen für den nächsten Sprint

Sprint-Planung



Product-Backlog: Bekannte Anforderungen

Meeting: Sprint Review (Ende eines Sprints)

- Wird von Scrum Master organisiert
- Team präsentiert dem Product-Owner die neuen Features in Form von lauffähiger Software (kein PowerPoint etc.)
- Nicht erledigte Stories wandern zurück ins Product-Backlog

Team

Sprints:
Immer gleich lang, z.B. 14 Tage

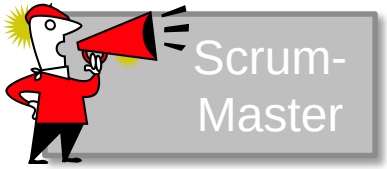
Review

Scrum-Master

Product-Owner

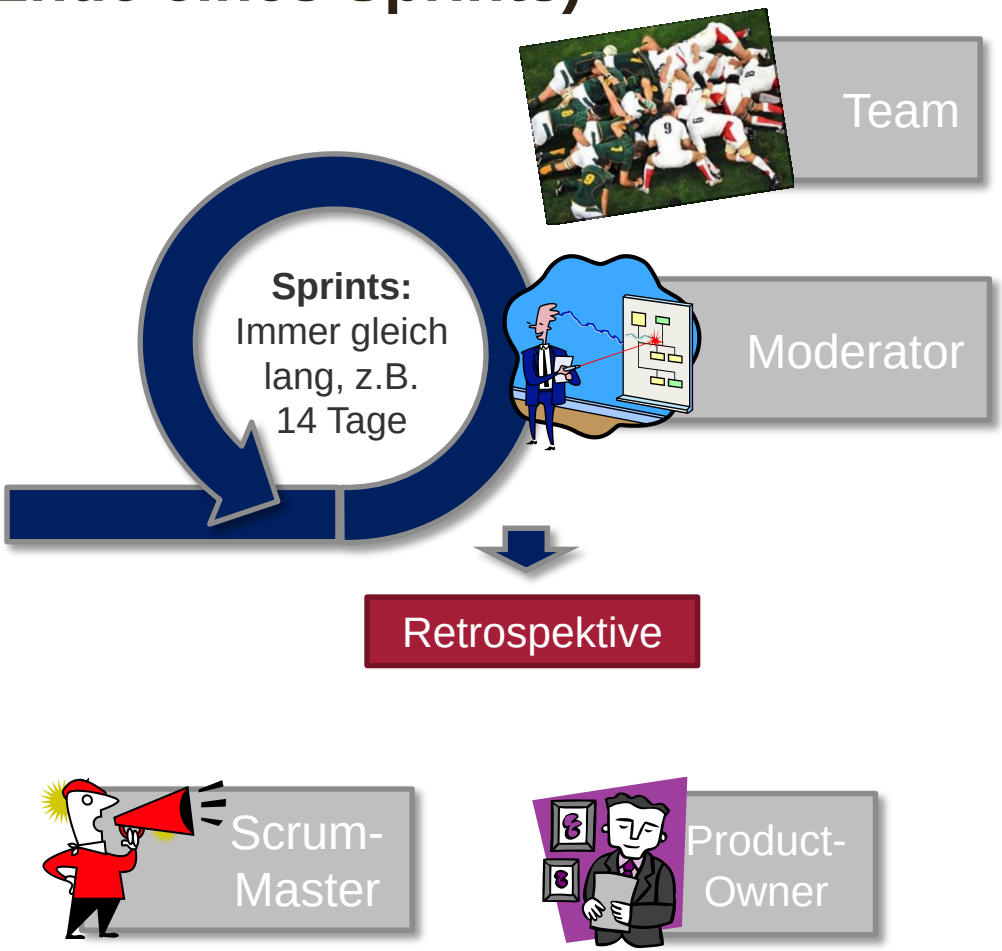
Meeting: Daily Scrum (während Sprint)

- Findet (täglich) zur gleichen Zeit am gleichen Ort statt
- Scrum-Master und Team
- Wird von Scrum Master organisiert
- Dauert max. 15 Minuten
  - Scrum Master achtet auf Einhaltung der Zeit
  - Was wird besprochen?
    - Was habe ich seit dem letzten Daily Scrum getan?
    - Was werde ich bis zum nächsten Daily Scrum tun?
    - Was behindert mich?



Meeting: Retrospektive (Ende eines Sprints)

- Wird von neutraler Person **moderiert** (z.B. Betreuer eines anderen Projekts oder anderer Kollege)
- Wird von Scrum Master organisiert
- Themen
  - Was haben wir gelernt?
  - Was lässt sich verbessern?
- Findet nach jedem Review statt



Meeting: Retrospektive (Ende eines Sprints)

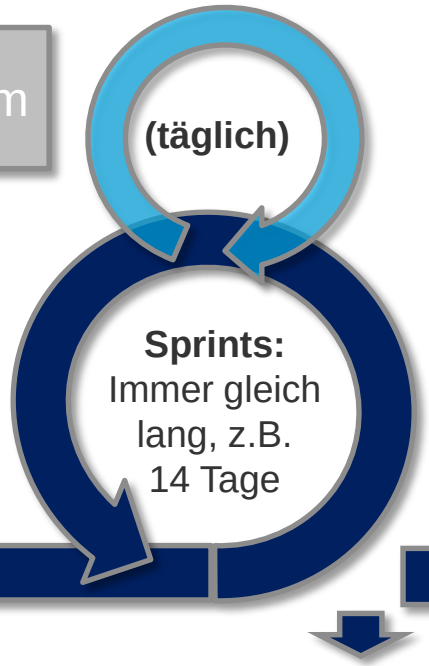
- Thema muss “Verbesserung des Prozesses” sein (nicht nur Technik)
- Rückkopplung ist der zentrale Wert für kontinuierliche Verbesserung
  - Treffen aller Projektbeteiligten zum Zweck des Lernens in regelmäßigen Abständen
  - Reflektion des vergangenen Zeitabschnitts
    - Was hat gut funktioniert? Bzw. was hast sich verbessert?
    - Was hat schlecht funktioniert? Warum? Bzw. was hat sich nicht verbessert?
    - Welche Blockaden sind aufgetreten?

- **Muss durch Scrum-Master dokumentiert werden!**

Scrum: Ablauf

Team

Scrum-Master



Daily Scrum:
Daily Scrum dauert 15 min.
Jedes Teammitglied beantwortet kurz:

1. Was habe ich seit dem letzten Daily Scrum getan?
2. Was hat mich dabei behindert?
3. Was werde ich bis zum nächsten Daily Scrum tun?

Sprint Backlog:
Stories für den nächsten Sprint

Sprint Planung

Product-Owner

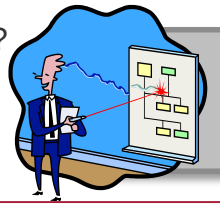
Product Backlog:
Bekannte Stories

Reviews

Sprint Review:
Neue Features werden dem Product-Owner präsentiert

Retrospektive

Sprint Retrospektive:
Was haben wir gelernt?
Was können wir verbessern?



Moderator

Gemeinsamer Besitz des Quelltextes

- Jedes Teammitglied darf in allen Teilen des Quelltextes Änderungen vornehmen
- Kein „Eigentümer“
- Konsequenzen
  - Jeder, der Fehler und Missstände im Code Entdeckt, behebt diese
  - Quelltextkonventionen: Anzahl Leerzeichen bei Einrückungen, Klammersetzung, Namensgebung etc.
  - Software muss automatisch geprüft werden

Testprotokolle

- Stories sollen nach RTD-Schema bearbeitet werden
- Dementsprechend sollte pro Story ein kleines Testprotokoll abgeliefert werden nach dem Schema:
  - Welche Funktionalität wurde getestet?
  - Wie wurde getestet?
  - Was war das Ergebnis?
- Beispiel:
  - Funktionalität: Webservice zum Registrieren eines Benutzers
  - Test: Testrequests (Req.0, Req.1,…)
  - Ergebnis: alle erfolgreich

Danke.

Sebastian Burg

Eberhard Karls Universität Tübingen
Lehrstuhl für Eingebettete Systeme

E-Mail: sebastian.burg@uni-tuebingen.de