



# Scrum

HAW Hamburg / Fachbereich Informatik

Tim Lücke

([Tim.Lueecke@haw-hamburg.de](mailto:Tim.Lueecke@haw-hamburg.de))





# Ballpoint Game

## Playbook

- 2 min Einführung
- 2 min Regeln
- 5 Iterationen:
  - 2 min Vorbereitung  
→ Schätzung
  - 2 min Durchführung
- Debrief



## Regeln

- Sie sind ein großes Team
- Jeder muss den Ball einmal haben
- Ball muss "Air-Time" haben
- Kein Ball zum direkten Nachbarn
- Startpunkt = Endpunkt
- Bälle auf dem Boden sind aus dem Spiel



# Agenda

## ■ **Hintergrund**

- Scrum in a nutshell
- Projektsteuerung in Scrum



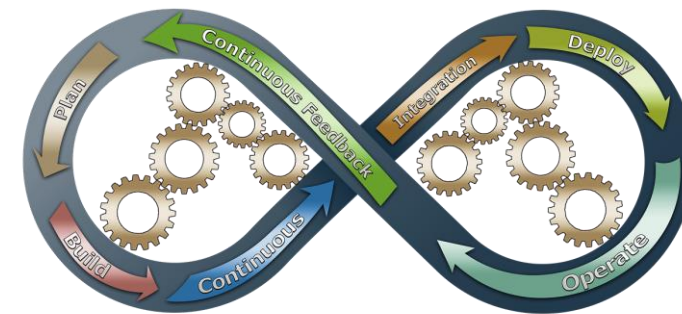
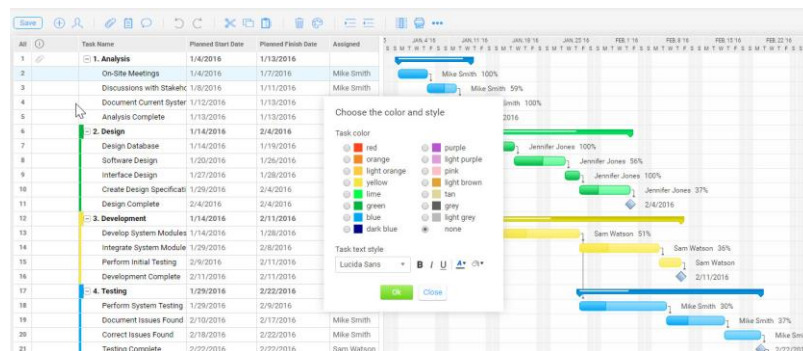
# Man unterscheidet im allgemeinen zwischen zwei unterschiedlichen Methodiken der Projektführung

## Klassische / Traditionelle Ansätze

- Wasserfallmodell
- Iteratives Wasserfallmodell
- V-Modell
- Rational Unified Process

## Agile Methoden

- Extreme Programming
- Kanban
- SCRUM
- ...





# Der Agile Ansatz

- Geisteshaltung durch agiles Manifest populär geworden
- Umgesetzt durch diverse "Agile Methoden"
- Ein wesentlicher Bestandteil: konstantes, schnelles Feedback!

**Agile is not  
a thing you buy.**

**Agile is  
a thing you are.**

## **Agile Manifest**

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Quelle: <http://agilemanifesto.org/>



# Agiles Manifest – Prinzipien 1/2

Our highest priority is to **satisfy the customer** through **early** and continuous delivery of valuable software.

**Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers **must work together daily** throughout the project.

Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of **conveying information** to and within a development team is **face-to-face conversation**.



## Agiles Manifest – Prinzipien 2/2

**Working software** is the primary **measure of progress**.

Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to **technical excellence and good design** enhances agility.

**Simplicity** - the art of maximizing the amount of work not done--is essential.

The best **architectures**, requirements, and designs **emerge** from self-organizing teams.

At regular intervals, **the team reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.



# Agenda

- Hintergrund

- **Scrum in a nutshell**

- Projektsteuerung in Scrum



# Scrum - Kernidee

- Scrum = „Gedränge“ (aus dem Rugby)
- **Annahme:** Entwicklungsprozesse sind so komplex, dass sie sich kaum im Voraus planen lassen
- **Idee:**
  - Scrum gibt einen groben Rahmen vor
  - darin organisiert sich das Team weitestgehend selbst; dies soll die Produktivität steigern
  - das Team übernimmt gemeinsam die Verantwortung für die Fertigstellung von Arbeitspaketen
  - Kontrolle und Regulation „von oben“ wird in Scrum abgelehnt
  - Scrum gibt keine konkreten Artefakte und Entwicklungsmethodiken vor (z.B. Use Cases, Komponentenorientierter Entwurf); dies ist projektindividuell zu gestalten



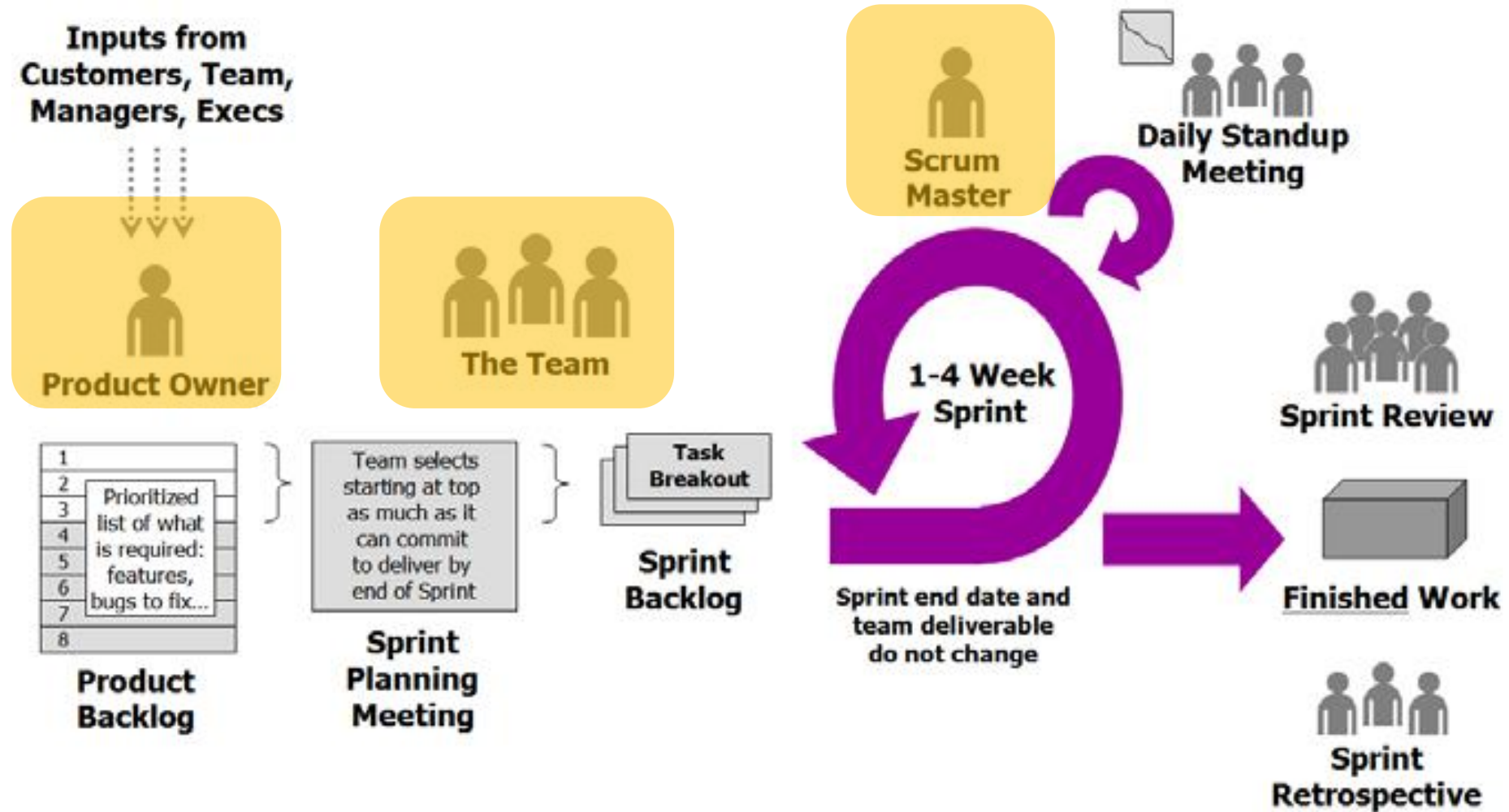


# Scrum im Überblick

- **Scrum** ist ein agiles Managementframework zur Entwicklung von Software mit wenigen klar definierten Regeln
- Anwendung von drei **Rollen**:
  - Product Owner
  - Team
  - Scrum Master
- Das **Product Backlog** enthält alle priorisierten Anforderungen
- Erstellung von Produktinkrementen innerhalb kurzer Arbeitszyklen, genannt **Sprint**
  - Jedes Produktinkrement ist getestet und dokumentiert.
  - D.h. jeder Sprint beinhaltet u.U. Analyse, Design, Implementierung, Test, Bugfixing und Dokumentation!
  - Jeder Sprint führt zu einer neuen Version die ausgerollt werden kann
- Als agiles Framework verkörpert Scrum die Werte des agilen Manifests
- Scrum fördert die enge Zusammenarbeit der Beteiligten
- Scrum ist kein Wundermittel!
- Das erfolgreiche Anwenden ist ein Lernprozess, der Zeit, Geduld und Disziplin benötigt; oft sind die ersten Sprints (Iterationen) schwierig
- Jedes Team/Projekt interpretiert Scrum leicht anders



# Scrum in einem Bild zusammengefasst





# Rollen in Scrum – Product Owner

- Er **repräsentiert** die Endkundenbedürfnisse (traditionell die Rolle des Produktmanagers)
- Er ist für die **Erreichung der wirtschaftlichen Ziele** verantwortlich und steuert dieses durch das priorisierte Product Backlog (und den Releaseplan)
- Er erstellt das Produktkonzept und das **Product Backlog**
  - dies wird kontinuierlich bearbeitet
  - neue Anforderungen kommen hinzu
  - Anforderungen werden (evtl. neu) priorisiert
  - hochpriorisierte Anforderungen werden verfeinert in neue Anforderungen.
  - Er kommuniziert eng mit dem Kunden und bestimmt dessen Bedürfnisse (z.B. durch Kundenworkshops)
- Er erstellt und aktualisiert den **Releaseplan**, er führt somit wichtige Projektmanagementaufgaben selbst durch



# Rollen in Scrum – Team

- Das Team führt **alle Arbeiten** aus, die zur Umsetzung der Anforderungen in auslieferbare Produktinkremente notwendig sind
- Das gesamte Team ist **für die Qualität des Produkts verantwortlich** inkl. der nicht-funktionalen Anforderungen
- Das Team **entscheidet allein**, wie viele Anforderungen es innerhalb des nächsten Sprint in ein Produktinkrement umwandeln kann
  - es entscheidet über die Arbeitsschritte und die Organisation
  - dies ist entgegen des traditionellen Managements, bei dem Verantwortung und Ausführung getrennt werden
  - Scrum Teams sind klein: 5-9 Mitarbeiter sind eine sinnvolle Größe



# Rollen in Scrum – Scrum Master

- Agiert als **Coach** und hilft somit dem Team, Scrum richtig einzusetzen
- **Überwacht** die korrekte Einhaltung der Scrum-Regeln (sorgt für Disziplin)
- **Moderiert** Meetings
- Sollte bemüht sein, dass sich das **Team** möglichst schnell **selbst organisiert**

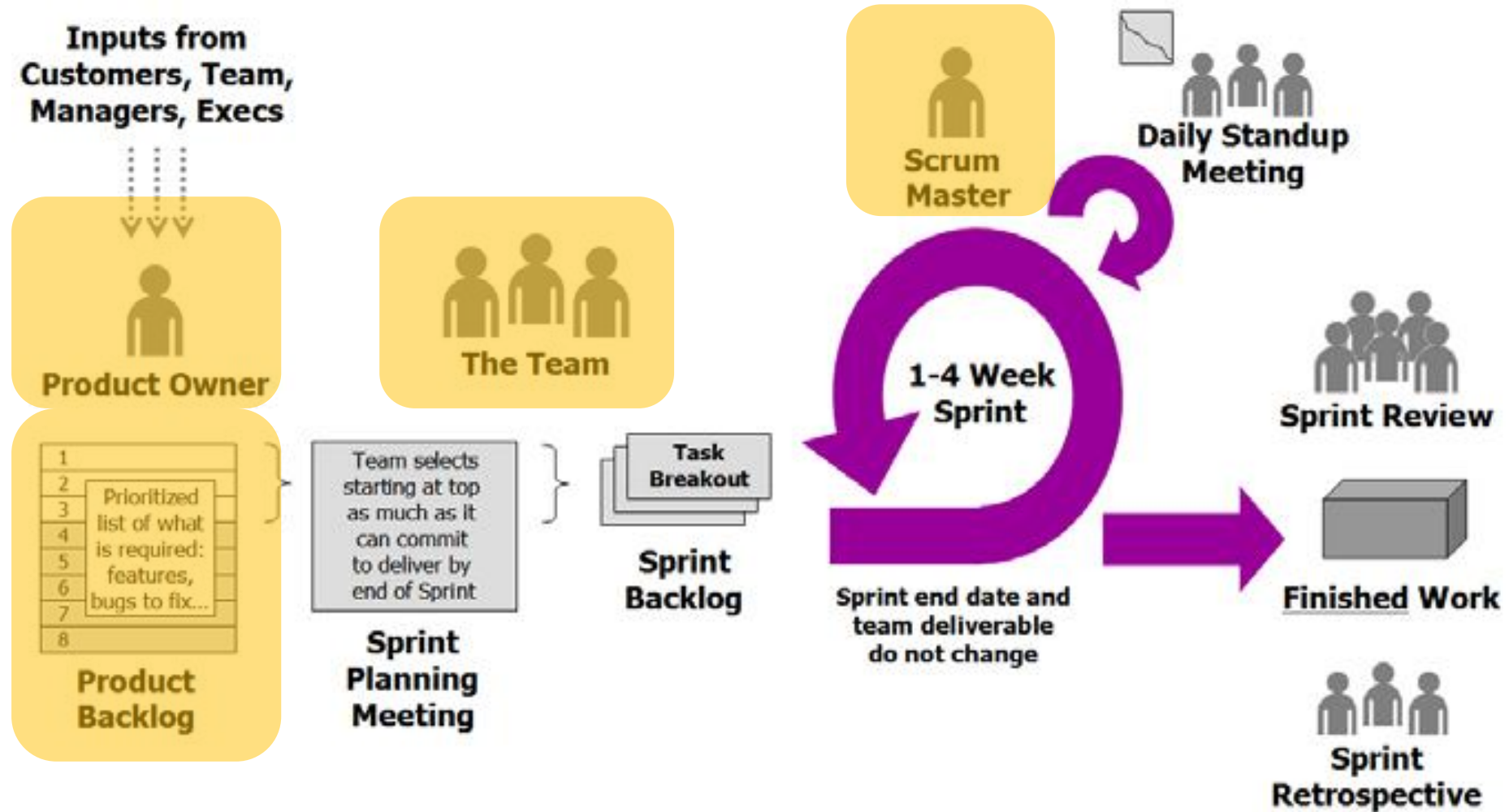


<https://www.youtube.com/watch?v=eNe0UEsBaIA>



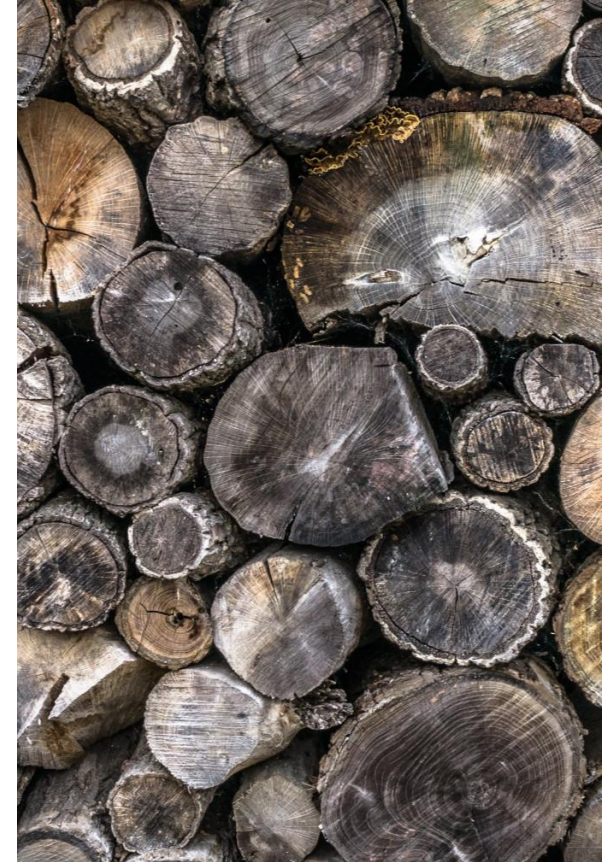


# Scrum in einem Bild zusammengefasst



# Artefakt – Product Backlog (1/3)

- Enthält die **Features/Anforderungen** des zu entwickelnden Produkts
  - Grundlage für die erste Fassung des Product Backlog ist das Lastenheft/die User Stories bzw. Interviews, Workshops, etc.
  - umfasst alle Funktionen die der Kunde wünscht
  - vor jedem Sprint neu bewertet/priorisiert (Backlog Refinement)
  - Lebt, d.h. wird ständig erweitert und gepflegt (Grooming)
  - Enthält z.B. auch: Bugs, Technische Zwischenaufgaben, ...
- **Hoch priorisierte Features** für den nächsten Sprint
  - stehen am Anfang
  - werden von den Entwicklern im Aufwand geschätzt
  - sind ausreichend „klein“ (vergl. gesplittete User Stories)
  - in das Sprint Backlog übernommen und in Unteraufgaben (Tasks) zerlegt
- **Niedrig priorisierte Features**
  - nicht detailliert beschrieben, somit wird nur Zeit für die wesentlichen Elemente verwendet
  - Sollten trotzdem kontinuierlich beobachtet und evtl. geschätzt werden







## Artefakt – Product Backlog (2/3)

Welche Faktoren entscheiden über die **Priorität** und damit die **Reihenfolge**?

- Wieviel **Business Value** steckt hinter einem Feature?
- Gibt es **Abhängigkeiten** zwischen den Items?
- Wieviel **Risiko** steckt in einer Story, wenn Sie umgesetzt wird oder nicht?  
(kann die Reihenfolge in beide Richtungen beeinflussen)
- Wieviel **Aufwand** wird benötigt für die Umsetzung (Story Points)?
- Wie groß sind die **Kosten**, wenn die Story nicht umgesetzt wird?  
(gerade für technische Verbesserungen sehr relevant)
- **Externe Einflüsse** wie das Geschäftsumfeld



## Artefakt – Product Backlog (3/3)

Wann muss mehr Zeit in das Refinement investiert werden?

- Wie **vertraut** sind die Beteiligten (Product Owner, Fachabteilung, Anforderungsanalytiker, Entwicklungsteam etc.) mit der fachlichen Domäne?
- Wie **homogen** werden Geschäftsprozesse und Anwendungsfälle von verschiedenen Benutzern durchgeführt beziehungsweise bearbeitet?
- Wie viele **fachliche Varianten und Ausnahmen** sind zu berücksichtigen beziehungsweise werden erwartet?
- Wie **neu oder verändert** ist der Ablauf?
- Wie viele **verschiedene Personen** sind als Anforderungsgeber oder -beeinflusser zu berücksichtigen?
- Wie hoch ist das Risiko, durch zu wenig Requirements-Engineering wichtige Abhängigkeiten oder Details zu vergessen, die später Mehrkosten oder Verzögerungen verursachen?
- Wie gravierend sind die Folgen, wenn wichtige fachliche Varianten oder Details vergessen werden?

[Quelle: <http://www.heise.de/developer/artikel/Gedanken-ueber-agiles-Requirements-Engineering-948348.html>]



# Artefakt – Product Backlog mit GitLab: Issue Board

The screenshot displays the GitLab interface for the 'GitLab Community Edition' project. The top navigation bar includes links to Projects, Groups, Activity, Milestones, and Snippets. The main header shows the project name and a search bar. The left sidebar contains navigation icons for Home, Issues, Merge Requests, Wiki, and Settings.

The 'Issues' section is active, showing a list of issues. The top of the issues list includes filters for 'Open' (9,395), 'Closed' (19,748), and 'All' (29,143). There are buttons for 'Edit issues' and 'New issue'. A search bar with the placeholder 'Search or filter results...' and a 'Most popular' dropdown are also present.

The issues listed are:

- Relations between Issues** (#4058) · opened a year ago by Job van der Voort · Discussion · Product work · UX · feature proposal · Issues · 266 likes · 1 comment · 127 replies · updated about 19 hours ago
- Add group wiki page support** (#4037) · opened a year ago by Ken Phillis Jr · Next 3-6 months · Accepting Merge Requests · Community Contribution · Platform · UX · customer · feature proposal · shortlist · wiki · 1 like · 215 likes · 67 replies · updated a week ago
- Merge train/Release train/Merge when master succeeds: run build on merged code before merging** (#4176) · opened a year ago by Frederik Zahle · Backlog · CI/CD · EE Premium · Product work · ci-build · customer · direction · feature proposal · frequently duplicated · 208 likes · 1 comment · 79 replies · updated 3 days ago
- Custom Roles** (#12736) · opened a year ago by Rolando Torres · Backlog · Platform · SP2 · customer · direction · feature proposal · frequently duplicated · permissions · user management · 169 likes · 1 comment · 90 replies · updated 4 days ago
- Please bring squash option when merging MRs to CE** (#34591) · opened 3 months ago by Jonas Kello · Discussion · feature proposal · merge requests · stewardship · 166 likes · 2 comments · 24 replies · updated 5 days ago
- Customize branch name when using create branch in an issue** (#21143) · opened a year ago by Adi Gerber · 10.1 · Accepting Merge Requests · Community Contribution · Discussion · In review · UX ready · feature proposal · frequently duplicated · Issues · repository · 1 like · 140 likes · 86 replies · updated about 9 hours ago
- Move Fast-Forward Merge and Semi-Linear Merge to CE** (#20076) · opened a year ago by Markus KARG · 10.1 · Deliverable · Discussion · In review · UX ready · backend · feature proposal · frontend · merge requests · 137 likes · 35 replies · updated a day ago
- Provide an option/toggle in settings so that private repo commits show up on public user profile graph** (#14078) · opened a year ago by Cameron Banga · Next 3-6 months · Platform · feature proposal · frontend · graphs · settings · 133 likes · 1 comment · 108 replies · updated a week ago
- Let's encrypt support for GitLab Pages** (#28996) · opened a year ago by Job van der Voort · Backlog · CI/CD · feature proposal · pages · 117 likes · 47 replies · updated 2 months ago
- Shared CI runners for groups** (#10244) · opened a year ago by Ivan Kertov · Next 3-6 months · CI/CD · CD2 · customer · feature proposal · frequently duplicated · 103 likes · 45 replies · updated about 16 hours ago



# Scrum – Schätzworkshops

- Durchgeführt typischerweise in **Backlog Refinements**
- als **Expertenschätzung** mit Team, Product Owner, Scrum Master
  - Product Owner stellt Anforderungen vor
  - Scrum Master moderiert
  - Den Workshop zeitlich begrenzen; wenn Zeit abgelaufen, dann ist Schluss!
  - Sinnvoll: < 2 Stunden
- Vor Beginn des ersten Sprint **möglichst** alle Einträge des Product Backlog abschätzen
  - Da sich das Product Backlog ständig ändert, sind auch Schätzworkshops während der Sprints nötig
- Team diskutiert/stellt Fragen
  - Was soll gemacht werden?, Gibt es Abhängigkeiten?, Komplexität?, Risiken?, usw.
  - Abschätzungen sind **Teamaufwände**, alle Arbeitsschritte berücksichtigen!
    - jeder Sprint beinhaltet u.U. Analyse, Design, Implementierung, Test, Bugfixing, Dokumentation



# Scrum – Aufwandsschätzung Product Backlog

- Scrum gibt keine Schätzgröße vor, denkbar sind Stunden/Tage, oder eine **abstrakte Größe**
- Populäre Skala: Fibonacci Folge (andere denkbar!):

Punktwert	Semantik
0	Kein Aufwand
1	Sehr kleiner Aufwand
2	Kleiner Aufwand: 2 x sehr kleiner Aufwand
3	Mittlerer Aufwand: kleiner + sehr kleiner Aufwand
5	Großer Aufwand: 2+3
8	Sehr großer Aufwand: 3+5
13	Riesiger Aufwand: 5+8

- Falls Schätzung hoch („13“): Hinweis, dass Anforderung noch nicht richtig verstanden oder zu grobgranular!



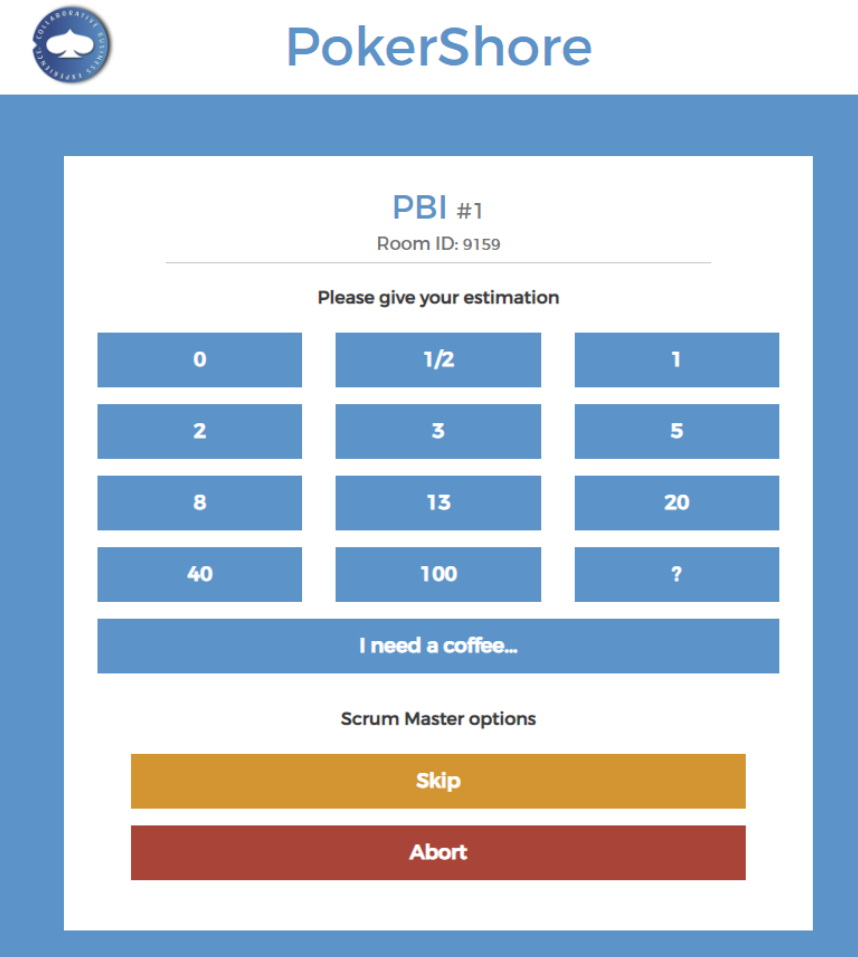
# Scrum – Aufwandsschätzung Product Backlog

- Punkte sind **nur relative** und keine absoluten Schätzgrößen! d.h. Aufwand ist nur in Bezug auf eine andere Anforderung bestimmbar
  - Also: zunächst eine kleine/klare Anforderung wählen, deren Umsetzung vertraut ist
  - nachfolgende Schätzungen im Vergleich zu den vorhergehenden Schätzungen
  - für Schätzungen gleichgroße Anforderungen gruppieren
  - Product Backlog zur Diskussion ausdrucken
  - siehe Regeln für Expertenschätzung
- Es werden **keine perfekten Schätzwerte** erwartet
  - allerdings: es darf auch kein Spekulationsworkshop werden
  - Schätzungen werden in Folgeworkshops verändert
- Schätzung soll sich wie alles andere im Prozess **über die Zeit verbessern**



# Beliebte Variante: Scrum Poker

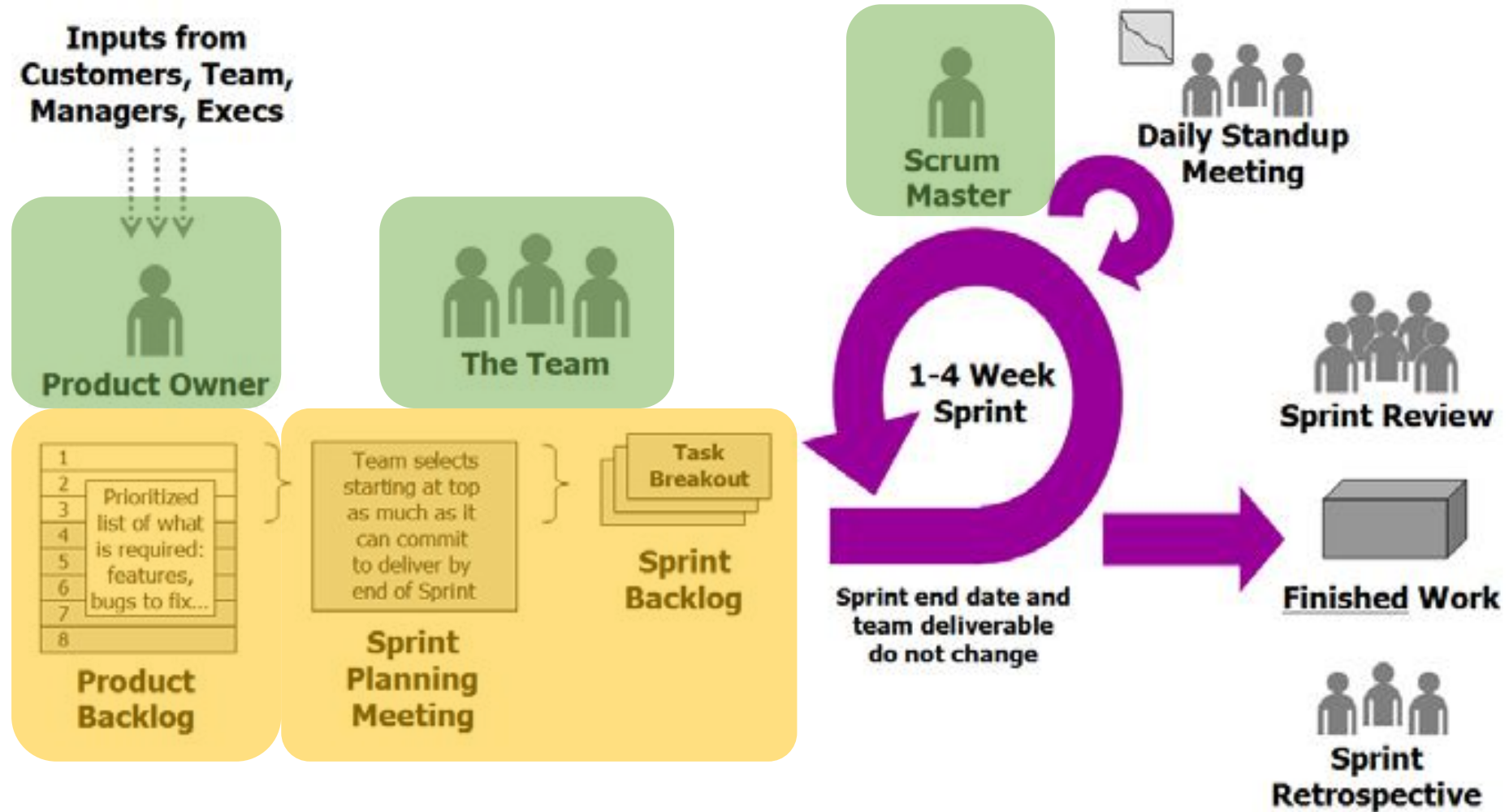
- Jedes Teammitglied ist ein Mitspieler
- Ein Spiel pro Backlog Item (PBI)
- Jeder Mitspieler gibt Schätzung ab
- Scrum Master moderiert und "deckt die Karten auf"
- Anschließend: Diskussion über Unterschiede
- Evtl. Wiederholung der Schätzung
- Kostenlose App: [PokerShore](#)



The image shows a screenshot of the PokerShore app interface. At the top left is a circular logo with a spade symbol. The title "PokerShore" is at the top right. The main screen displays "PBI #1" and "Room ID: 9159". Below this is a section titled "Please give your estimation" with a grid of buttons for values: 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100, and ?. Below the grid is a button labeled "I need a coffee...". At the bottom, under "Scrum Master options", are two buttons: "Skip" (orange) and "Abort" (red).



# Scrum in einem Bild zusammengefasst





# Sprint

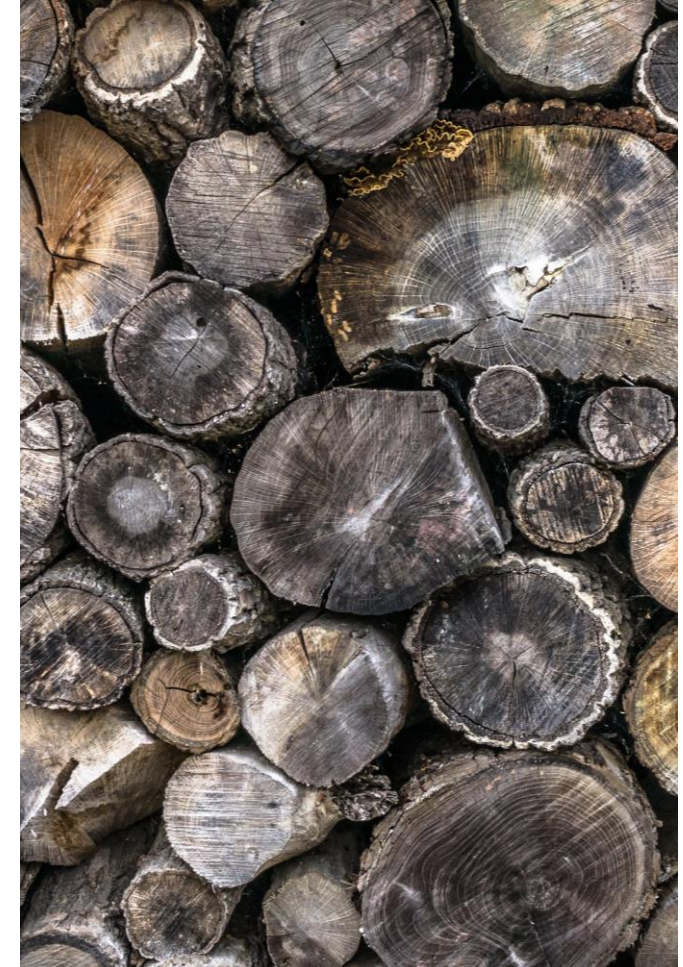
- Ein Sprint ist eine **Iteration** in Scrum
- Sprints haben eine **fixe Länge** (z.B. 2 Wochen, 4 Wochen)
- **Sprintergebnis** sollte eine lauffähige Software sein, die dem Kunden am Sprintende zur Evaluation präsentiert und zur Verfügung gestellt wird
- Während eines Sprints ist **keine Kundeninteraktion** (Product Owner) erlaubt
  - Konsequenz: Features/Anforderungen sind während eines Sprints fixiert
  - Keine Störung von „außen“
  - Wesentliche Aufgabe eines Scrum Master dafür zu sorgen, daß dies eingehalten wird



# Artefakt – Sprint Backlog

Enthält die (meist technischen) Aufgaben („Arbeitspakete“, „Tasks“, „Subtasks“), die notwendig sind, um das **Ziel des Sprint** zu erfüllen

- eine Aufgabe sollte nicht länger als 16 Stunden dauern
- längere Aufgaben in Teilaufgaben zerlegen
- bei der Planung Kapazität/Arbeitsgeschwindigkeit („Velocity“) des Teams berücksichtigen
- Die Aufwände werden durch das Team während eines Sprints aktualisiert
- Jedes Teammitglied kann Pakete ergänzen, ändern, löschen



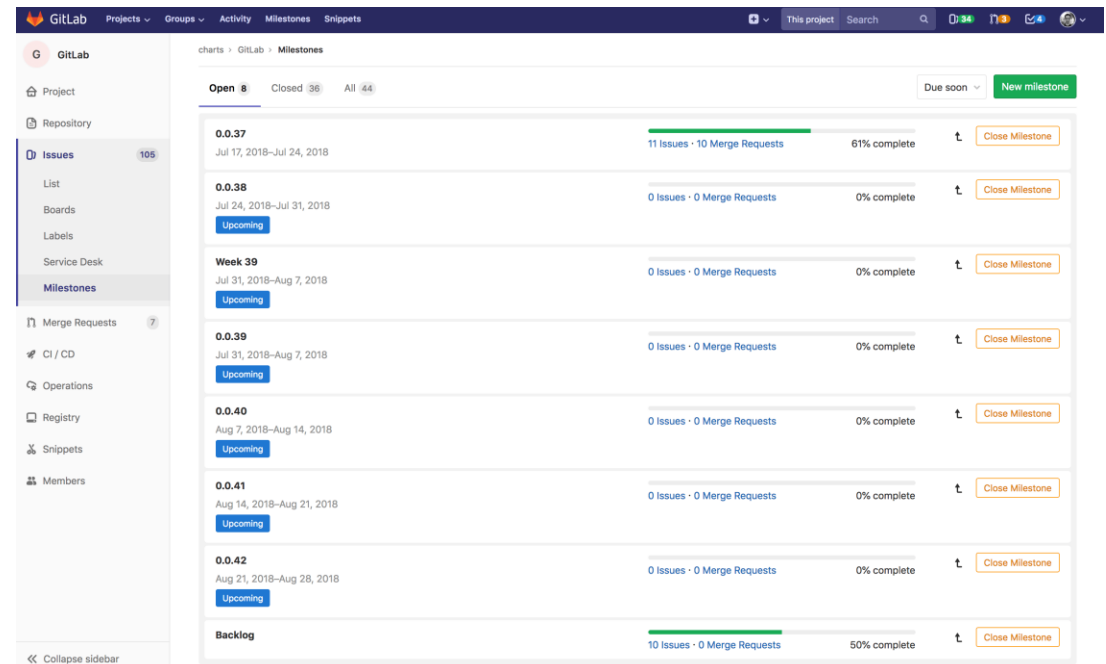


# Scrum – Sprint-Planung

- Sprint Planungstreffen **Teil 1**
  - Product Owner erklärt dem Team die Anforderungen der Backlog-Einträge
  - Einigung über das Sprint-Ziel (welche Anforderungen können wir im Sprint umsetzen?); dies ist die Basis für die Abnahme des Sprints
  - User Stories im Sprint bilden das Sprint Backlog
- Sprint Planungstreffen **Teil 2**
  - eigenverantwortliche Planung durch das Team
  - Zerlegung der Backlog-Einträge in Arbeitspakete (AP), Verteilung der Aufgaben an die Teammitglieder
  - Erstellung des Sprint Backlogs mit Schätzung der AP in Stunden

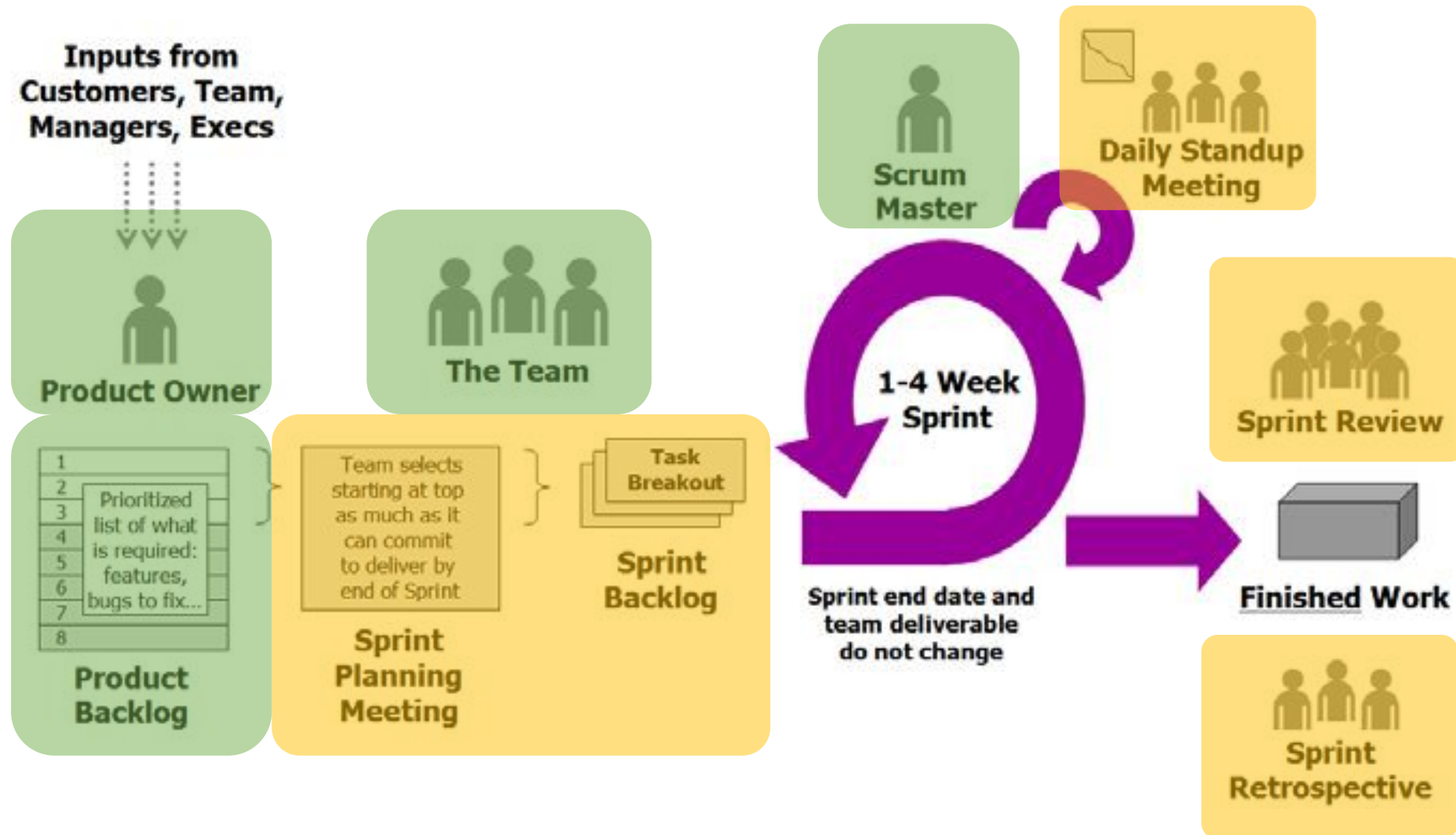
## Sprints mit GitLab

- Sprint = Milestone
- Tasklist für Aufteilung der User Storeis
- Eigenes Board pro Sprint





# Scrum in einem Bild zusammengefasst





# Scrum – Daily Scrum / Statusrunde

- maximal **15-Minuten** Statusmeeting des Teams (im Stehen!)
- dient dem **Informationsaustausch** der Teammitglieder untereinander; alle sollten möglichst alles wissen
- **Jedes Teammitglied** beantwortet die folgenden Fragen kurz und bündig:
  - „Welche Arbeitspakete hast du seit dem letzten Meeting fertiggestellt?“
  - „Welche Arbeitspakete wirst du bis zum nächsten Meeting bearbeiten?“
  - „Gibt es Probleme, die dich bei deinen Aufgaben behindern?“
- **Keine Diskussion** (Scrum Master!)
- **Hindernisse** werden vom Scrum Master in das Impediment Backlog aufgenommen und beseitigt



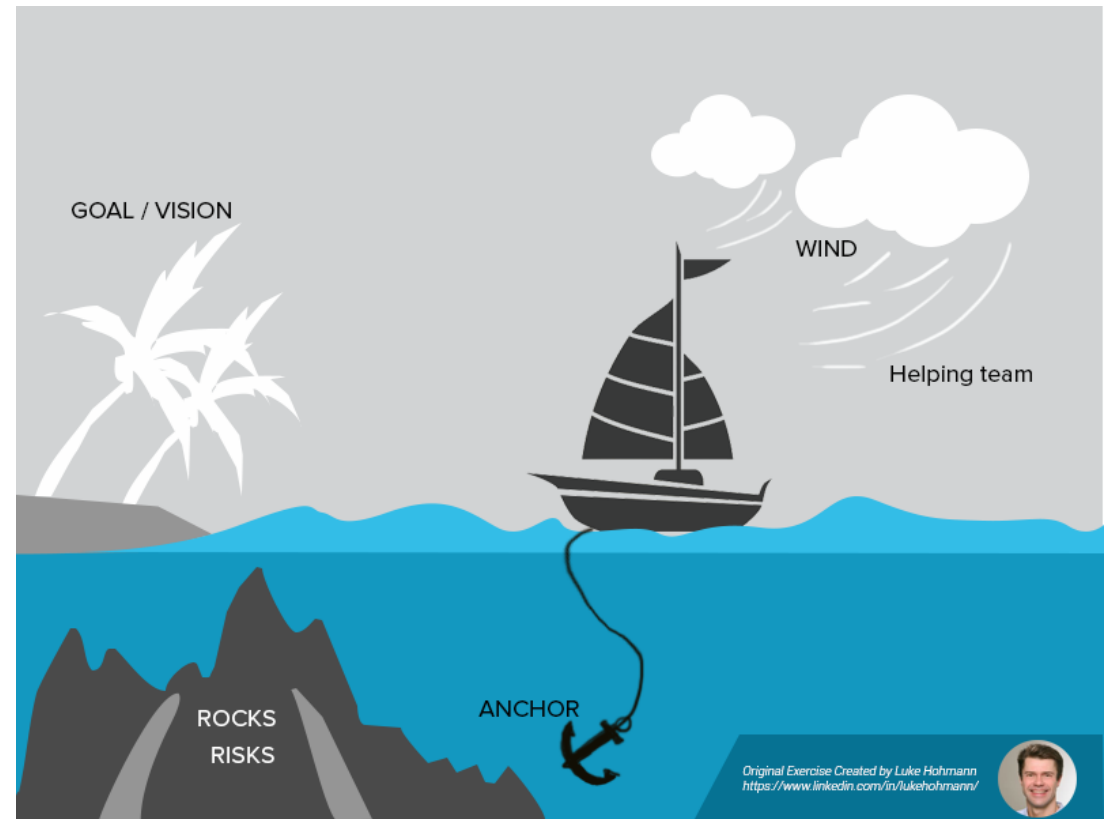
# Scrum – Sprint Review

- das **Sprint Ergebnis** wird durch das Team und dem Kunden einem Review unterzogen
  - Präsentation der neuen Features
  - Präsentation von Bugfixes
  - Präsentation von technischer Infrastruktur
  - Präsentation von technischen Konzepten
  - ...
- **Kunde prüft**, ob dies seinen Anforderungen entspricht; eventuelle Änderungen werden im Product Backlog dokumentiert
- In der Regel wird hierfür eine **neue Version** der Software deployed / zur Verfügung gestellt, um Tests auch im Nachgang zu ermöglichen



# Scrum – Sprint Retrospective

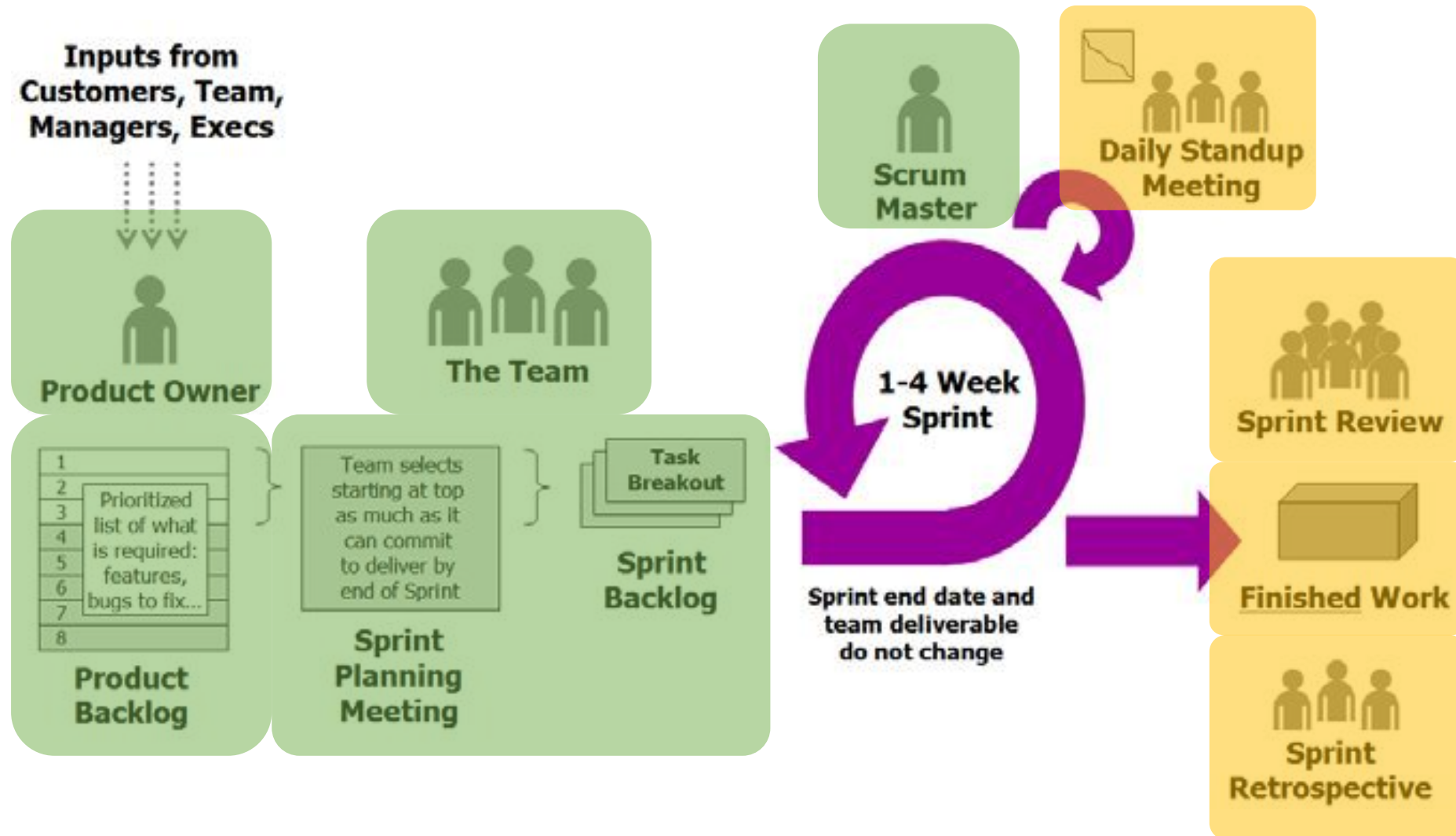
- **Betrachtung** der zurückliegenden Sprint-Phase: „Was war gut“, „Was könnte verbessert werden“
  - eventuelle Anpassung des Impediment Backlogs und/oder des Product Backlogs
- **Wesentliches Element** von Scrum zur Verbesserung der Team Performance
- **Offene und ehrliche** Diskussion
- Einfang des **Stimmungsbarometers**
- **Visualisierung** durch z.B. Sailboat Excercise
- Auch **nach einem Jahr** sollten Verbesserungen gefunden werden!
- Für **Remote** Online Whiteboards nutzen: z.B. <https://www.webwhiteboard.com/>



Quelle: <https://luis-goncalves.com/sailboat-exercise-sailboat-retrospective/>



# Scrum in einem Bild zusammengefasst





# Scrum – Definition of Done

- **Frage:** Wann ist eine Arbeit fertig?
- Für jedes Projekt / Team **unterschiedlich**
- Daher: **Definition of Done**
  - Vertrag für das Team
  - Beinhaltet alle Punkte die gelten sollen:
    - Acceptance Criteria sind erfüllt
    - Alle Test grün
    - Software startet
    - Lasttests durchgeführt
    - ...
  - Pro unterschiedliche Ebene:
    - Story
    - Sprint
    - Release





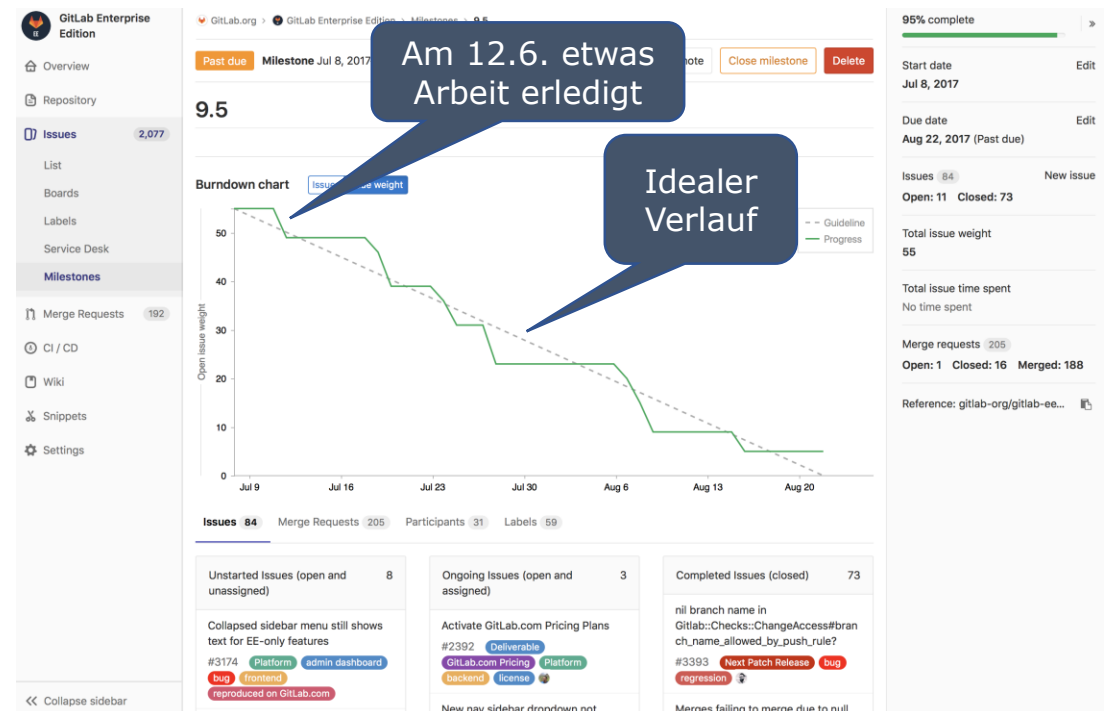
# Agenda

- Hintergrund
- Scrum in a nutshell
- **Projektsteuerung in Scrum**



# Steuerung von Scrum Projekten

- Berichterstattung in Scrum schafft hohes Maß an **Transparenz**
  - Verzögerungen werden ebenso schnell offenkundig wie ein schnellerer Fortschritt
- **Burndown-Bericht**
  - beschreibt den aktuellen Projektfortschritt
  - führt die Summe aller Aufwände im Sprint Backlog auf und zeigt, wie sich diese Aufwände während eines Sprints ändern
  - führt die Summe aller Aufwände im Product Backlog am Ende jedes Sprint auf und zeigt, wie sich Aufwände über Sprint-Grenzen (bis Release- oder Gesamtende) hinweg ändern





# Scrum - Velocity

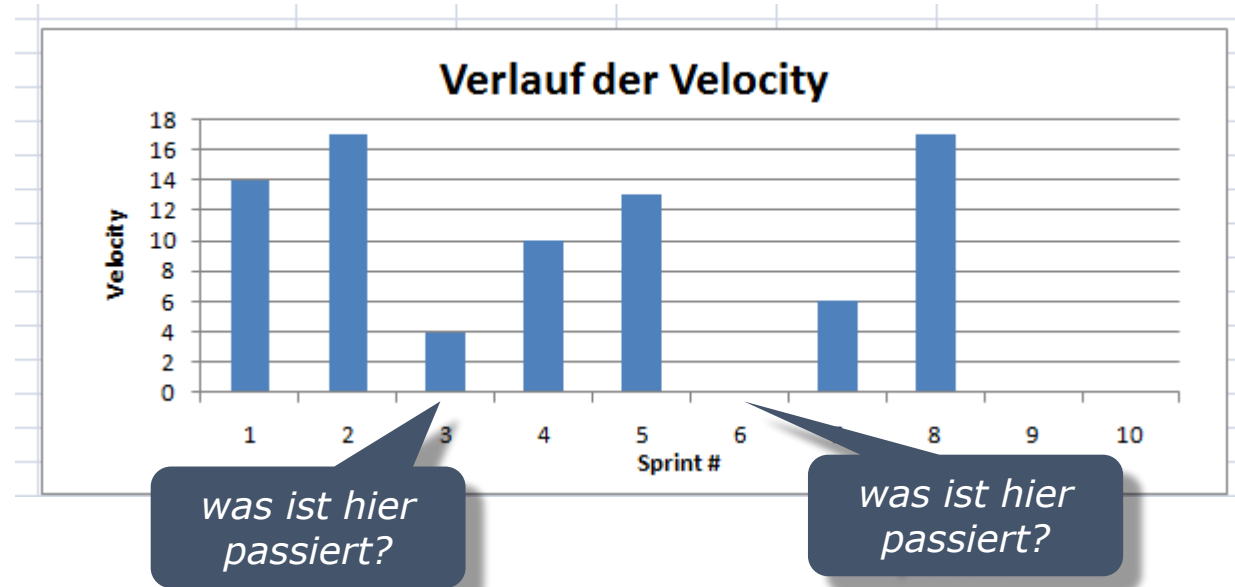
- Für eine bessere Planung benötigt: Einschätzung der **Entwicklungsgeschwindigkeit** (in Scrum: „Velocity“)
- **Velocity** = Summe aller Aufwände der am Sprintende vom Product Owner abgenommenen Arbeitsergebnisse
  - teilweise fertiggestellte oder defekte Ergebnisse werden in Scrum nie abgenommen; „99% fertig“ bedeutet „nicht fertig“!
  - daher werden niemals anteilmäßig Punkte für partielle Ergebnisse vergeben
- Wie können wir die Entwicklungsgeschwindigkeit **bestimmen**?
  - Glaskugel, Erfahrungen aus vorhergehenden ähnlichen Projekten
  - 2-3 Sprints durchführen und den erzielten Mittelwert nehmen

Product Backlog	Punkte geplant	Punkte erzielt
Der Benutzer soll sein Kundenprofil verwalten können.	3	3
Der Benutzer soll Kinokarten reservieren können.	5	0 (nicht abgenommen)
Das System soll nachts eine Statistik über die Kinobesuche des vergangenen Tages erstellen.	3	3
Velocity		6



# Scrum – Velocity

- Beispiel **Verlauf der Velocity** (aktueller Zeitpunkt ist Ende von Sprint 8)



- sollte nach anfänglichem „Holpern“ später ausgeglichen aussehen, sobald sich das Team eingespielt hat!
- **Relative Velocity** berechnen um Schwankungen durch Krankheit/Urlaub, Teamauf- und Abbau auszugleichen



# Fragen oder Anmerkungen?

Diskussion in slack über #scrum



# Literatur

