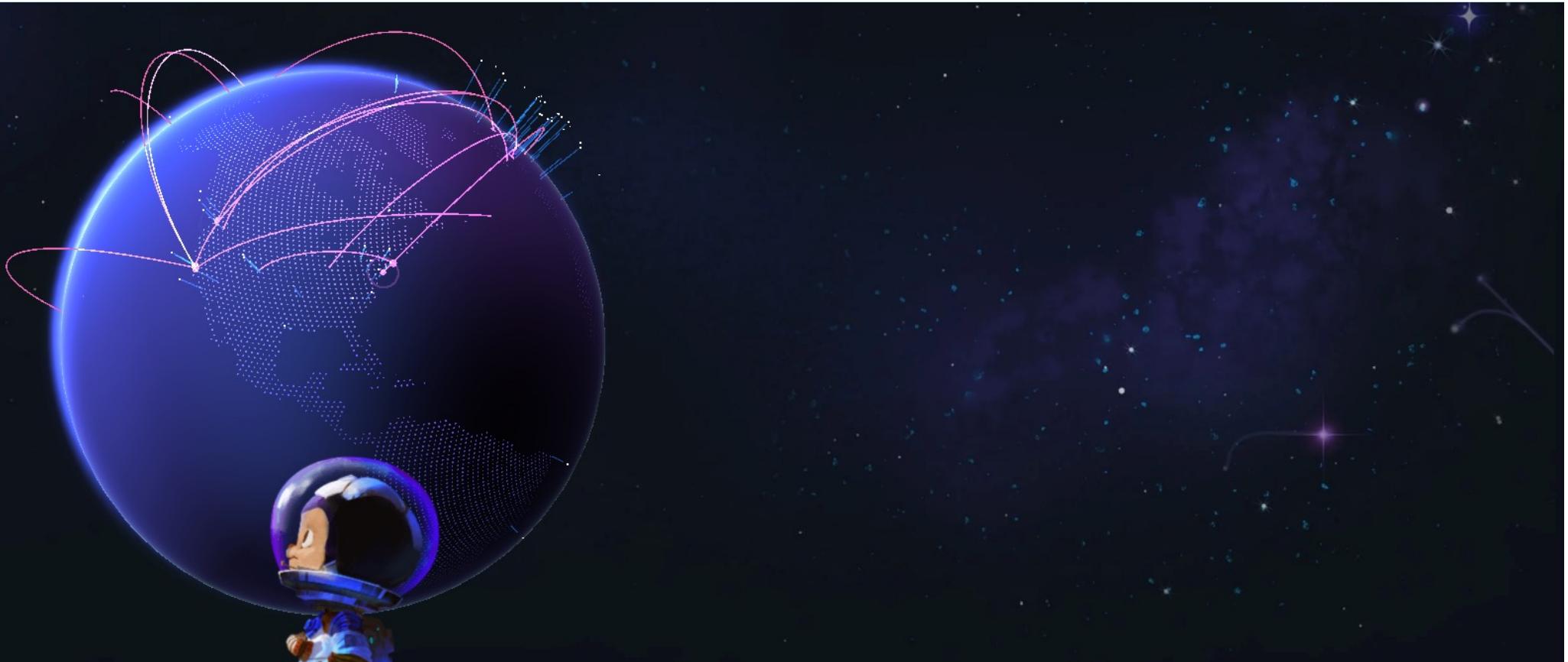


# Version Control Systems (VCS)

## Mit GIT und GITHUB



## **Versionskontrollsystem (Version Control System VCS)**

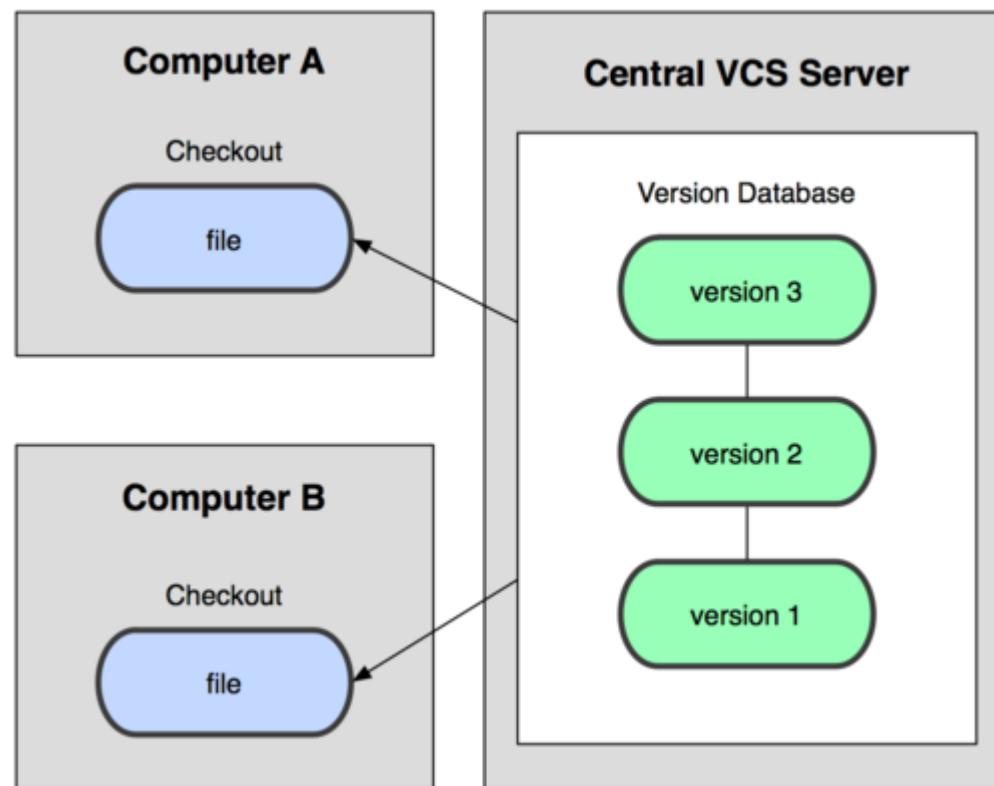
- Ein System zur Verwaltung und Versionierung von Software oder anderer digitaler Information
- Primärzwecke
  - Kollaboration mit digitalen Inhalten
  - Nachvollziehbarkeit von Änderungen und Historie
  - Integrität gewährleisten
  - Widerspruchsfreiheit von konkurrierenden Änderungen an den gleichen Stellen

## GIT

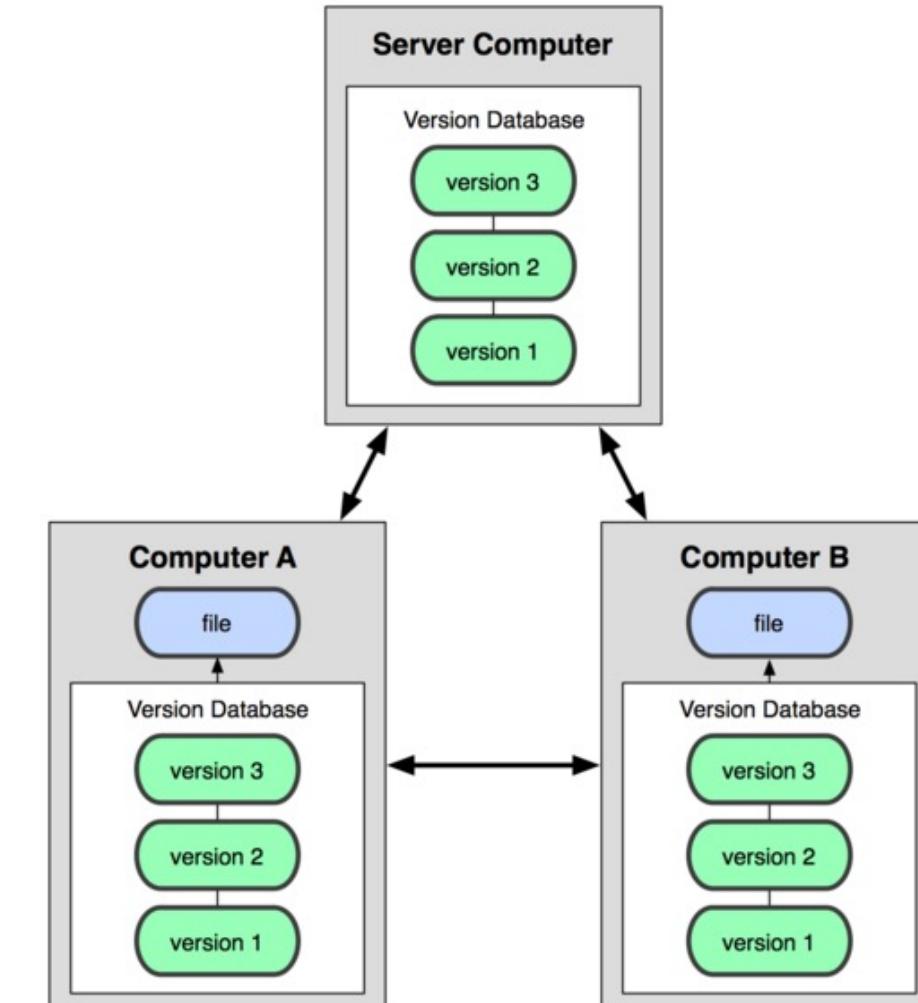
- Grundidee entwickelt von Linux Torvalds in 2005
- Die Linux Foundation suchte nach einer besseren Lösung zur Versionskontrolle für den Linux Kernel
- Primärziele: Schnelligkeit bzw. Performance und überprüfbare Integrität der verwalteten Daten.
- Besonderheit: Weg von einer zentralistischen Versionsverwaltung zu einer dezentralen Verwaltung von Kopien (Vergleichbar mit dem Prinzip der Blockchain)
- Lizenz: GNU-GPLv2-Lizenz
- Kurz: Vorgenommene Änderungen können jederzeit mit allen anderen Projektteilnehmern ausgetauscht und – sofern relevant – in das Repository aufgenommen werden.



## Klassische VCS Systeme vs GIT



Klassisches VCS (SVN)



GIT

## Begriffe: Repository

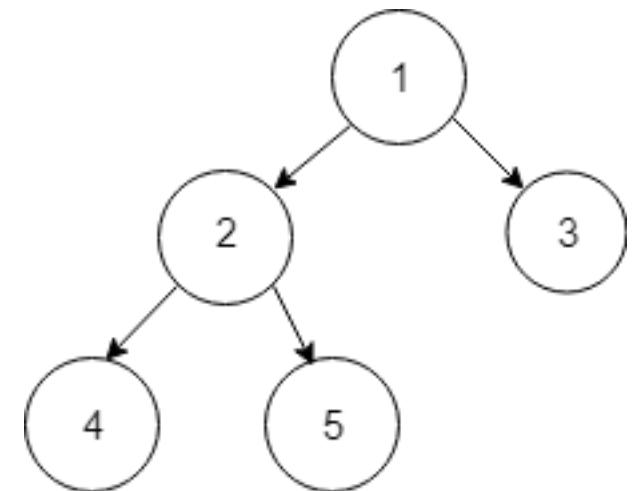
- Eine Ablage von digitalen Daten
- Die Historie der Änderungen an den digitalen Inhalten wird vollständig gespeichert
- Datenbankähnliches System
- Wird entweder an einem zentralen Ort, in der Cloud oder als sog. “Working Copy” auf Entwicklergeräten gefunden
- Kurzbegriff: Repo



Quelle: <http://gitbu.ch/>

## Begriffe: Working Tree

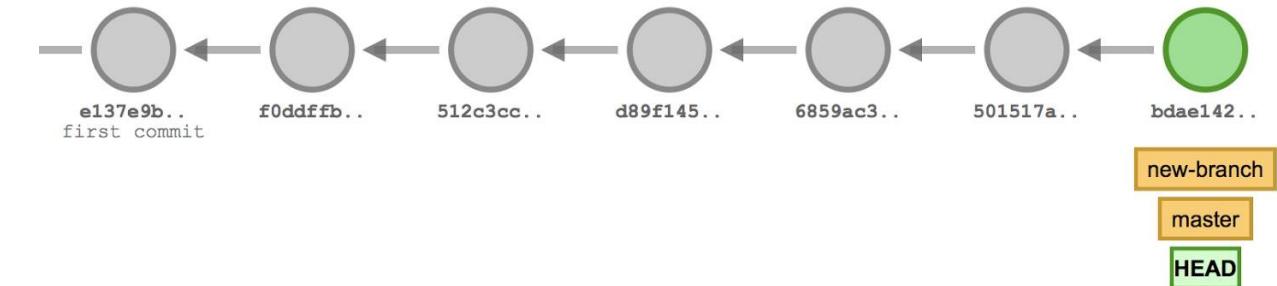
- Das Arbeitsverzeichnis von Git (in anderen Systemen manchmal auch Sandbox oder Checkout genannt).
- Modifikationen am Quellcode werden hier vorgenommen
- Oft auch Working Directory genannt



Quelle: <http://gitbu.ch/>

## Begriffe: Commit

- Veränderungen am Working Tree, also z.B. modifizierte oder neue Dateien, werden im Repository als Commits gespeichert.
- Zustand aller verwalteten Dateien zu einem bestimmten Zeitpunkt.
- Enthält neben den Änderungen auch Metadaten:
  - Timestamp (Datum Uhrzeit)
  - Commit Message (Nachricht mit Informationen zu den Änderungen)
- Bildet immer den Zustand aller versionierten Dateien zum Timestamp “X” ab

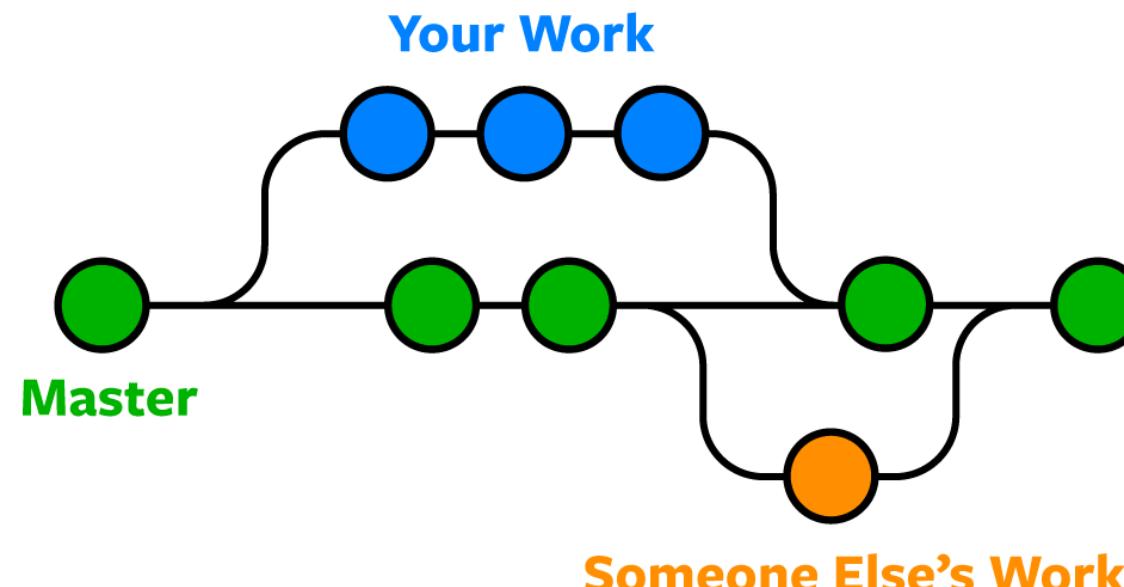


## Begriffe: HEAD

- Eine symbolische Referenz auf den neuesten Commit im aktuellen Branch.
- Von dieser Referenz hängt ab, welche Dateien Sie im Working Tree zur Bearbeitung vorfinden.
- Es handelt sich also um den „Kopf“ bzw. die Spitze eines Entwicklungsstrangs

## Begriffe: Branch

- Eine Abzweigung in der Entwicklung.
- Dient zur Entwicklung neuer Funktionalität oder sogenannten Features
- Ein neuer Zweig muss später dann mittels "Merge" wieder in den Hauptzweig integriert werden



## Begriffe: Clone

- Wenn man ein Repository klonet (clone) dann bekommt man eine lokale Kopie dieses Repositories
- Ein Clone enthält die gesamte Änderungshistorie des Repository
- Wird ein Git-Repository aus dem Internet heruntergeladen, so erzeugen Sie automatisch einen Klon (Clone) dieses Repositories.

Quelle: <http://gitbu.ch/>

## Begriffe: SHA-1

- Der Secure Hash Algorithmus
- Erstellt eine eindeutige, 160 Bit lange Prüfsumme (40 hexadezimale Zeichen)
- Kann für beliebige digitale Informationen eingesetzt werden
- In GIT werden alle Commits in Git nach ihrer SHA-1-Summe benannt (Commit-ID), die aus dem Inhalt und den Metadaten des Commits errechnet wird.
- Es ist sozusagen eine inhaltsabhängige Versionsnummer, z.B.  
`f785b8f9ba1a1f5b707a2c83145301c807a7d661.`

Quelle: <http://gitbu.ch/>

## Begriffe: Tag

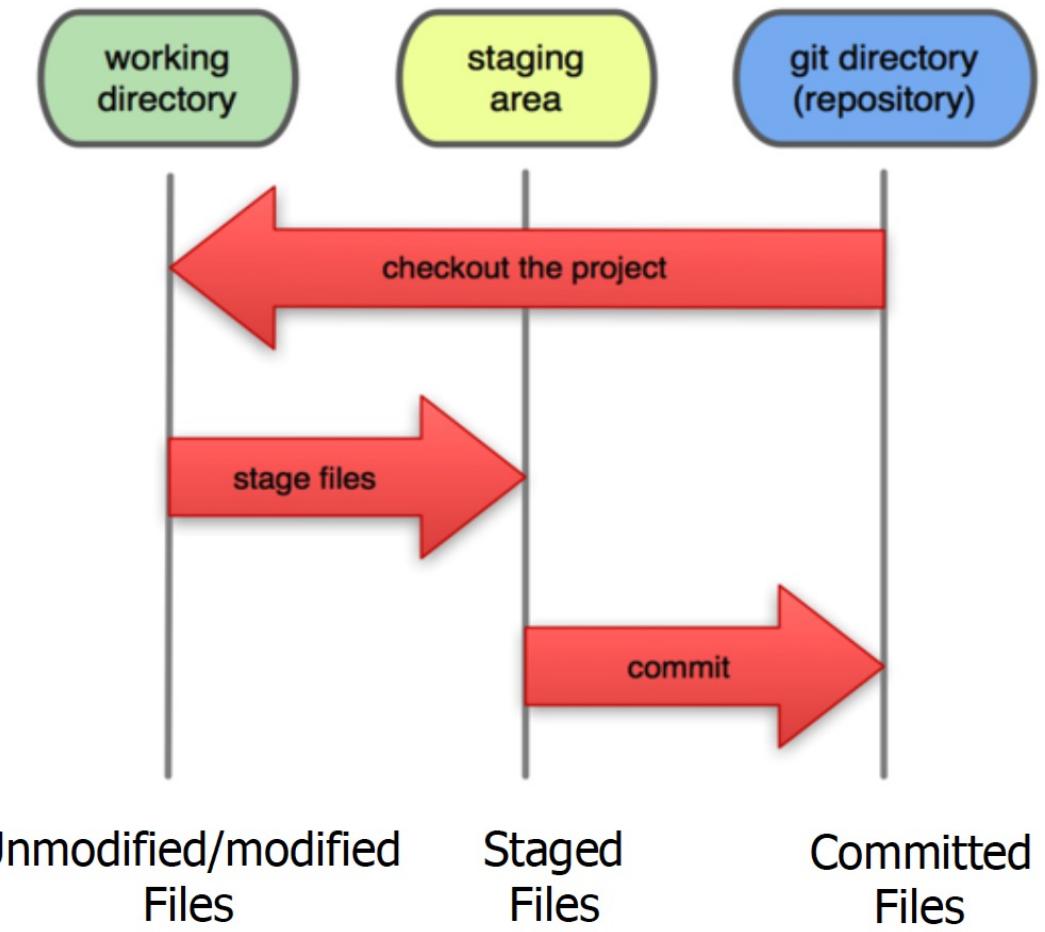
- Tags sind symbolische Namen für schwer zu merkende SHA-1-Summen.
- Wichtige Commits, wie z.B. Releases, können Sie mit Tags kennzeichnen.
- Ein Tag kann einfach nur ein Bezeichner, wie z.B. v1.6.2, sein, oder zusätzlich Metadaten wie Autor, Beschreibung und GPGSignatur enthalten.

Quelle: <http://gitbu.ch/>

## Staging

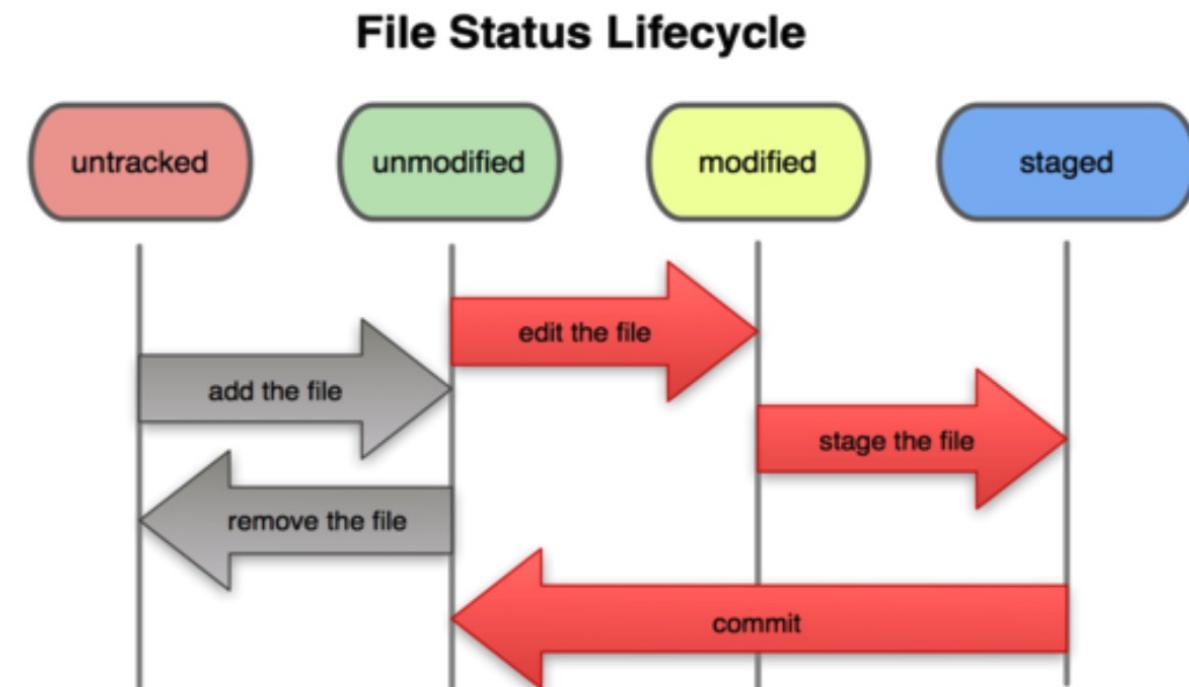
- Git verwendet ein zweistufiges System um lokale Änderungen nachzuverfolgen
- Staging bedeutet, dass Dateien vorbereitet wurden um in den lokalen Working Tree integriert zu werden. Dabei handelt es sich um Snapshots von Änderungen an mehreren Dateien
- Sobald ein “Commit” ausgelöst wird sind die Änderungen amtlich und sind teil der Historie.
- Was während der Staging Phase alles geändert wurde und beim Commit nicht mehr präsent ist wird verworfen. Nur das Delta zum letzten Commit bleibt bestehen

## Local Operations



## Staging

- **Modified:** Lokal geänderte Dateien
- **Staged:** Snapshot von gemachten Änderungen (Nur lokal verfügbar)
- **Commit:** Übertragene definitive Änderungen die teil der Historie geworden sind



## GIT Installieren

### Installation unter Windows

- Git kann für Windows heruntergeladen werden: <https://git-scm.com/>
- Auch nützlich ist GitHub Desktop Client für git. Dieser kann für Windows und Mac hier heruntergeladen werden: <https://desktop.github.com/>

### Installation unter MacOS X

- Für MacOS gibt man im Terminal git –version ein. Falls es noch nicht installiert ist, erscheint eine Abfrage wie man es installieren will.

Die Installation erfolgt mit allen Standard-Einstellungen.

## GIT konfigurieren

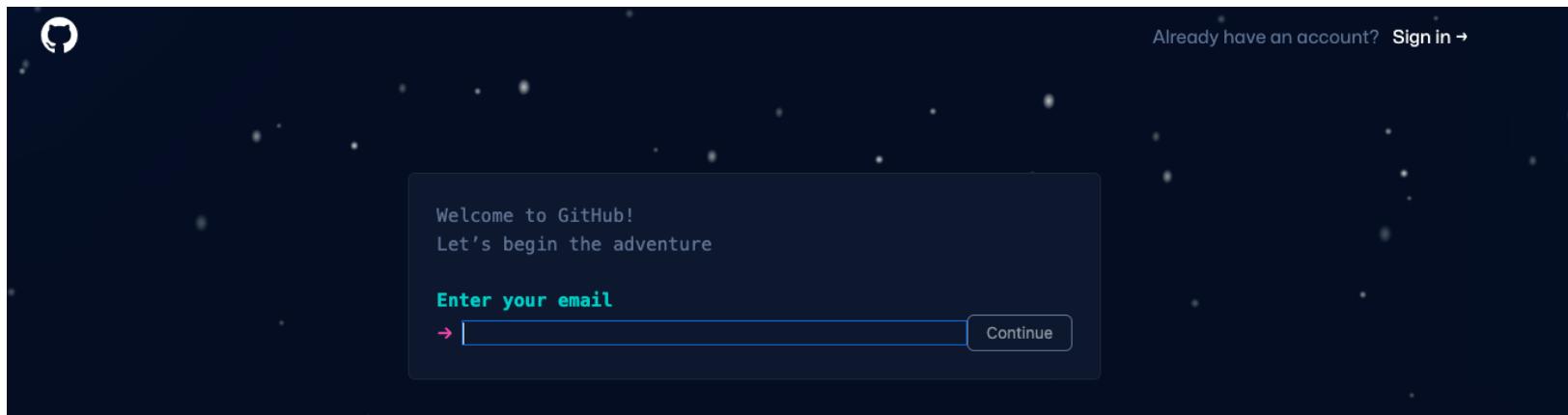
- Nach der Installation müssen wir einmalig eine Konfiguration Durchführen. Dazu öffnen wir die «Git Bash» (oder im Terminal unter MacOS). Es gibt einige Optionen, aber die wichtigsten sind Benutzername und E-Mail.

```
git config --global user.name "Vorname Nachname"
```

```
git config --global user.email name@email.com
```

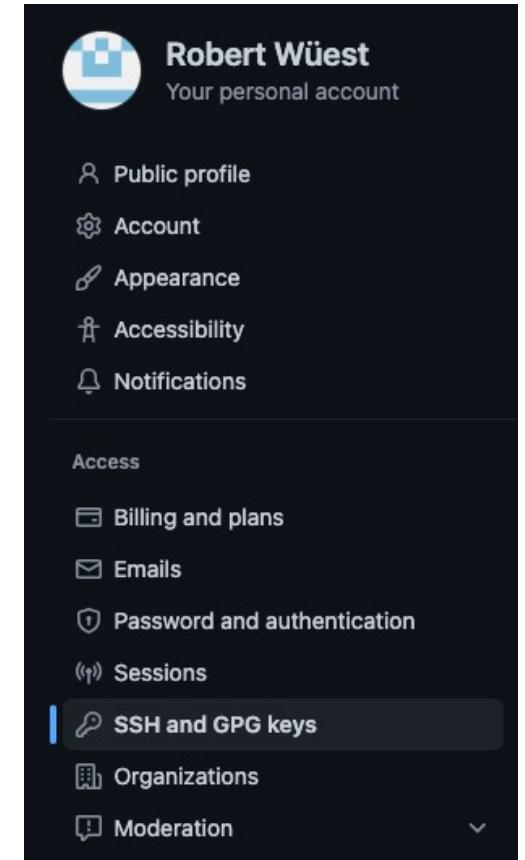
## Github Registrierung

- <https://github.com/signup>



## SSH Key erstellen und auf Github einrichten

- In der Git Bash (oder Konsole) mit ssh-keygen einen neuen Schlüssel erzeugen:  
**ssh-keygen**
- Es wird ein Pfad und Dateiname für den Schlüssel vorgeschlagen, wir belassen das.
- **Eine Passphrase wird bei Versionssverwaltung in der Regel nicht verwendet**, wir wollen mit dem Schlüssel eine sichere Verbindung um genau das Passwort nicht immer einzupinnen müssen.
- Im .ssh Directory befinden sich nun der Private und der öffentliche Schlüssel. Den öffentlichen Schlüssel kopieren wir auf GitHub. Dazu gehen wir auf «Settings» (Klick auf das Profilbild oben rechts). Wir wählen das Menu auf der linken Seite «SSH and GPG keys» und fügen den key mit «New ssh key» hinzu.



## Wichtige Befehle

<b>command</b>	<b>description</b>
<code>git clone <i>url</i> [<i>dir</i>]</code>	copy a Git repository so you can add to it
<code>git add <i>file</i></code>	adds file contents to the staging area
<code>git commit</code>	records a snapshot of the staging area
<code>git status</code>	view the status of your files in the working directory and staging area
<code>git diff</code>	shows diff of what is staged and what is modified but unstaged
<code>git help [<i>command</i>]</code>	get help info about a particular command
<code>git pull</code>	fetch from a remote repo and try to merge into the current branch
<code>git push</code>	push your new branches and data to a remote repository
others: <code>init</code> , <code>reset</code> , <code>branch</code> , <code>checkout</code> , <code>merge</code> , <code>log</code> , <code>tag</code>	

## Befehle: git status

Zu den wichtigsten Git-Grundlagen zählt eine gute Organisation des eigenen Arbeitsverzeichnisses. Über dieses schlagen Sie nicht nur eigene Änderungen und Neuerungen an einem Projekt vor, die anschliessend per Commit („Freischaltung“) übernommen werden, sondern beziehen auch Informationen über die Aktivitäten anderer Nutzer. Die Aktualität Ihrer Arbeitskopie können

Sie über folgende Eingabe überprüfen: `git status`

Bei einem gerade neu angelegten Repository bzw. einer absoluten Übereinstimmung von Hauptverzeichnis und Arbeitskopie erhalten Sie in der Regel die Information, dass es keinerlei neue Anpassungen an dem Projekt gab („No commits yet“). Zudem teilt Git mit, dass Sie aktuell keine eigenen Änderungen für den nächsten Commit geteilt haben („nothing to commit“). Um eine eigene, neue Datei zur Versionsverwaltung hinzuzufügen oder eine überarbeitet.

## Befehle: git add

Um eine eigene, neue Datei zur Versionsverwaltung hinzuzufügen oder eine überarbeitete Datei für den nächsten Commit anzumelden, wenden Sie den Befehl „git add“ auf diese Datei an (diese muss sich hierfür im Arbeitsverzeichnis befinden).

```
git add Test.txt
```

## Befehle: git commit

Sämtliche Änderungen, die Sie (wie im vorherigen Abschnitt beschrieben) für die Versionsverwaltung angemeldet haben, müssen immer per Commit bestätigt werden, damit sie in den HEAD aufgenommen werden. Beim HEAD handelt es sich um eine Art Index, der auf den letzten wirksam gewordenen Commit in Ihrer aktuellen Git-Arbeitsumgebung (auch als „Branch“ bezeichnet) verweist.

Das Kommando für diesen Schritt lautet:

```
git commit -m "Beschreibung der Änderung"
```

Sie erhalten nach der Ausführung von „git commit“ eine zusammenfassende Meldung zu dem Commit: In den voranstehenden eckigen Klammern steht zum einen der Name der Branch (Projektzweig; hier „master“, da unser Arbeitsverzeichnis gleichzeitig auch das Hauptverzeichnis ist), in den die Änderungen übertragen wurden, zum anderen die SHA-1-Prüfsumme des Commits (z.B. „c0fdf55“). Es folgen der frei gewählte Kommentar (hier „Test“) und konkrete Informationen über die vorgenommenen Anpassungen.

## Befehle: git branch / git checkout

Einen neuen Branch zu erstellen, ist nicht schwer: Sie benötigen lediglich die Anweisung „git branch“ und hängen dieser die gewünschte Bezeichnung für den Zweig an. Einen Beispiel-Branch mit dem Namen „test\_branch“ erstellen Sie beispielsweise auf folgende Weise:

```
git branch test_branch
```

Anschliessend können Sie jederzeit mit der Anweisung „git checkout“ in diesen Branch wechseln:

```
git checkout test_branch
```

# Agiles Projektmanagement

Einführung



# Was bedeutet “agile”?

- Iterativer Ansatz
- Unterstützt das Projektmanagement eines Teams
- Verstärkt die Reaktionsfähigkeit über den Verlauf eines Projekts
- Erlaubt Kurskorrekturen um am Ende den Bedürfnissen des Kunden oder Anwenders gerecht zu werden

# Charakteristiken von agilem Projektmanagement

- Schwerpunkte
  - Adaptive Planung
  - Evolutionsbasierte Softwareentwicklung (Iterationen)
  - Early delivery
  - Fortlaufende Verbesserungen
  - Reaktionsfähigkeit auf Unvorhergesehenes

# Das agile Manifest (2001)

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

[agilemanifesto.org](http://agilemanifesto.org)

# Das agile Manifest (2001)

## Kernaussagen

- Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.
- Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
- Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
- Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.
- Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
- Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
- Funktionierende Software ist das wichtigste Fortschrittsmaß.
- Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
- Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
- Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.
- Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
- In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

# Agile Softwareentwicklung

- Iterativer Ansatz zur Softwareentwicklung der konsistent ist mit den agilen Prinzipien
- Fokus auf Flexibilität, Interaktion und Transparenz innerhalb des Teams
- Selbstorganisation
- Interdisziplinarität

# Primärziel

Build what **is** needed, not what **was** planned

# Agiles Projektmanagement in der Organisation

## Conways Law

“If you ask an organisation with four separate teams to build a compiler...  
....you will get a 4-pass compiler!”

Melvin Conway 1968

# Grundidee der iterativen Planung

- Man entscheidet nicht Dinge zu einem Zeitpunkt wo man noch gar nicht das wesentliche Wissen über alle Details hat z.B. zu Projektbeginn
- Es wird das geplant worüber man gerade am meisten weiss um verlässliche Schätzungen zu machen
- Anpassung der Planung sobald neue Erkenntnisse vorliegen
- Zeiträume für die Planung müssen dementsprechend überschaubar gewählt werden

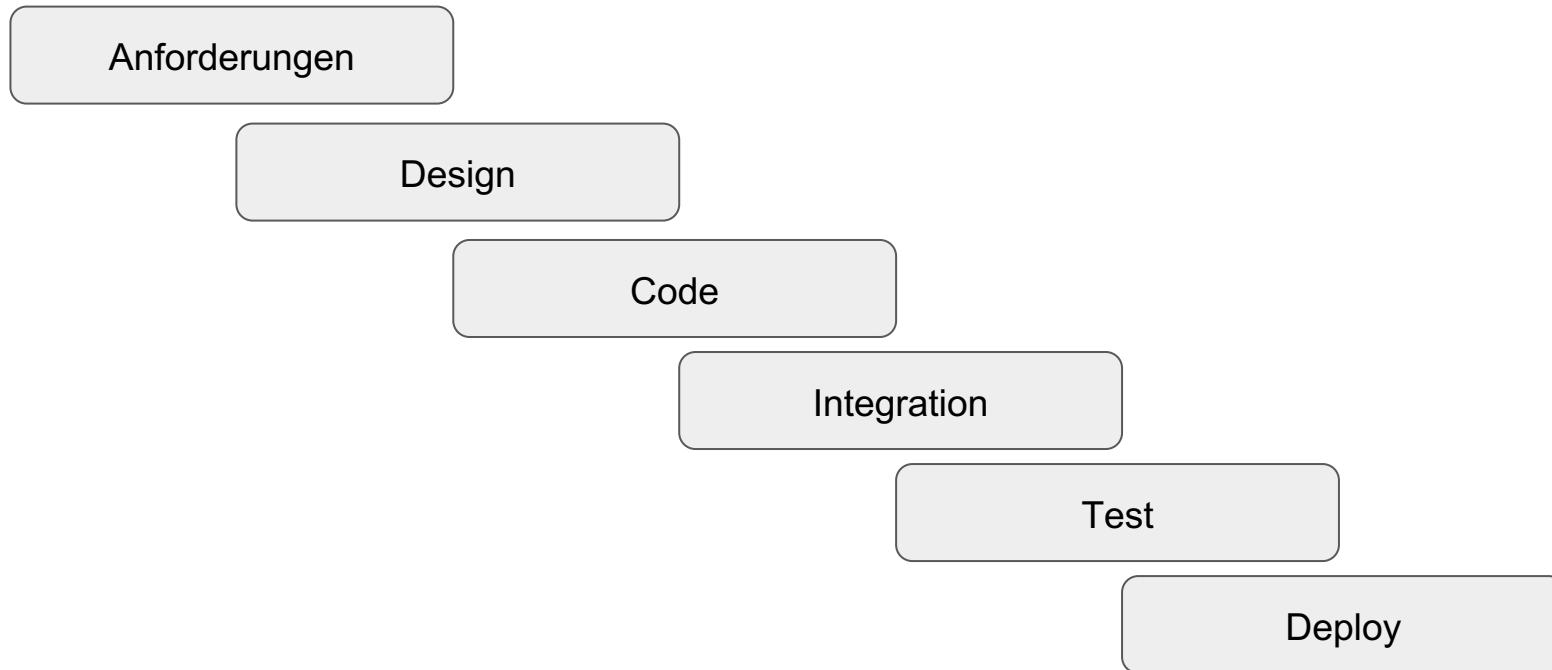
# Gruppenübung

Spaghetti - Challenge



# Methodologien

# Methodologien - Wasserfall

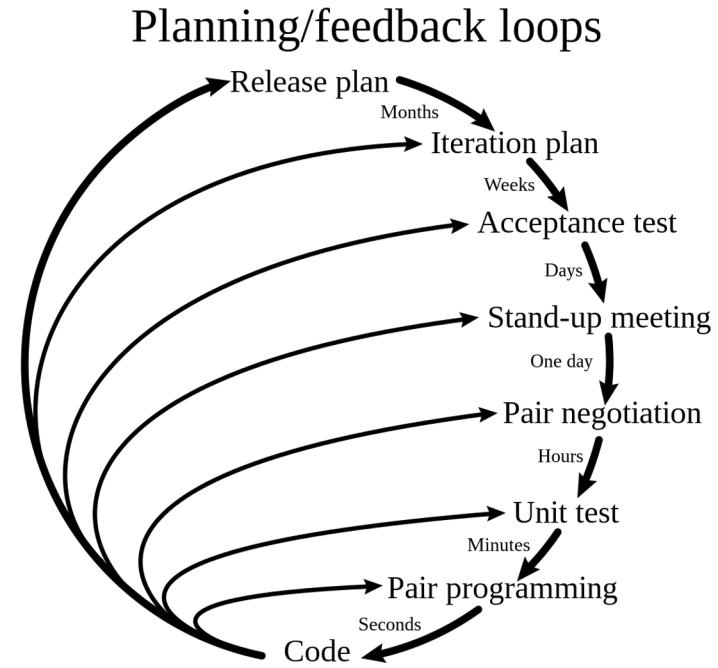


# Methodologien - Wasserfall

- Kein Handhabe für nachträgliche Anpassungen von Anforderungen
- Keine Wahrnehmung über den momentanen Stand und die Qualität der Umsetzung
- Jeder Schritt endet wenn der nächste beginnt
- Fehler die spät im Projekt entdeckt werden sind teuer
- Lange Zeiträume bis ein Projekt in Test oder Produktion gehen kann. Keine Zwischen Reviews
- Verschiedene Teams arbeiten isoliert voneinander (Design, Code oder Test)

# Methodologien - Extreme Programming (XP)

- Iterativer Ansatz
- Kleine Inkremente
- Prinzipien
  - Einfachheit
  - Kommunikation
  - Feedback
  - Respekt
- Ursprung der agilen Methode
- Ursprung: 1996 Kent Beck



# Methodologien - Kanban

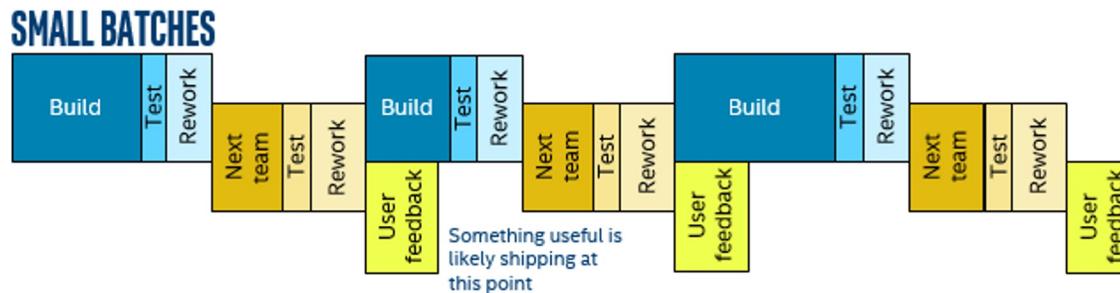
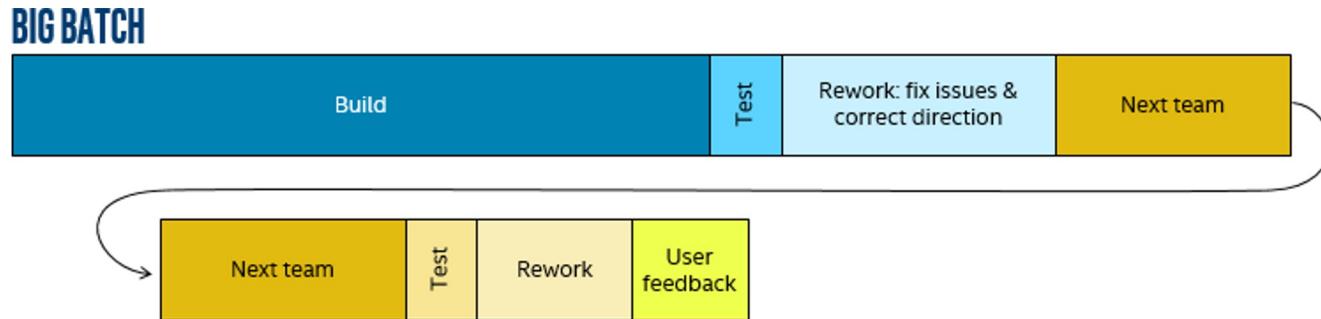
- 1970 aus Japan (aus der Fertigungsindustrie)
- Ursprünglich wurde es verwendet um Materialnachschub in den Produktionsketten zu verwalten mittels Anweisungskarten
- Prinzipien
  - Visualisierung des Workflows
  - Einschränken der momentan bearbeiteten Arbeitspaketen (Work in progress)
  - Kontrollieren und verbessern der Ablaufprozesse
  - Prozessrichtlinien mit expliziten Zieldefinitionen (Definition of done)
  - Fortlaufende Verbesserung (Continuous improvement)
  - Fortlaufende Lieferung (Continuous delivery)

# **Was ist agiles Arbeiten?**

# Agiles Arbeiten

- Arbeit in kleinteiligen Paketen (Small Batches)
- Minimum Viable Product (MVP)
- Verwerfen oder Behalten (Pivot or Persevere)
- Behaviour Driven Development (BDD)
- Test Driven Development (TDD)
- Pair Programming (PP)

# Small Batches



intel.com

# Small Batches

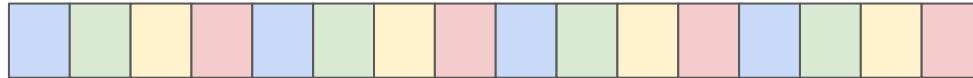
- Fallbeispiel: Versand von Broschüren oder Einladungsdossiers
- Die Arbeit gliedert sich in 4 Arbeitsschritte:
  - Falten, einfüllen, verschliessen, adressieren
- Wenn dies nun ca 50 mal gemacht werden muss und wir pro Arbeitsschritt 6 Sekunden pro Exemplar

Falten	Einfüllen	Verschliessen	Adressieren
--------	-----------	---------------	-------------

- Wie lange dauert es, bis ich das erste Exemplar vollendet habe?
- Welche Probleme könnten hier auftauchen?

# Small Batches

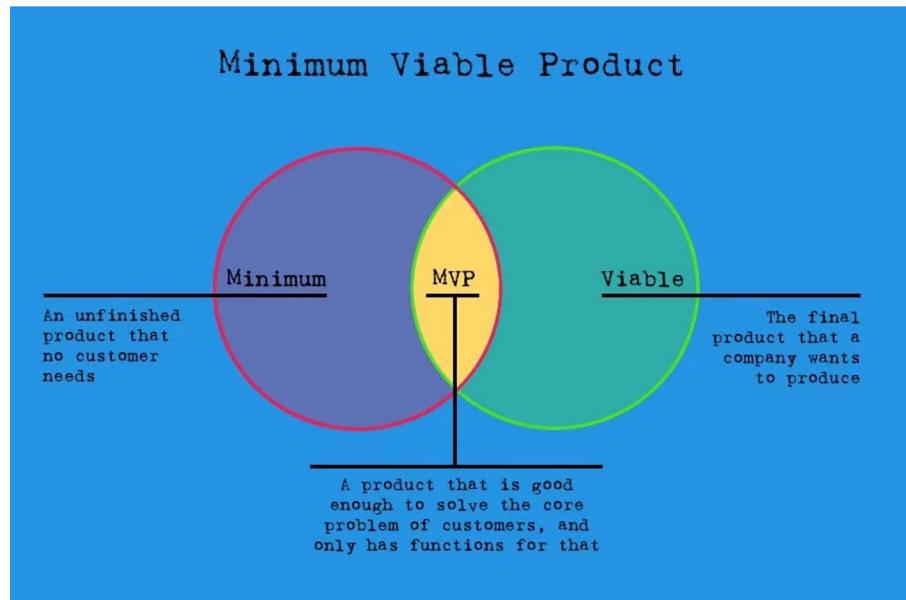
- Lösungsansatz: Verwendung des Single Piece Flow (SPF)



- Wie lange dauert es, bis ich das erste Exemplar vollendet habe?

# Minimum Viable Product

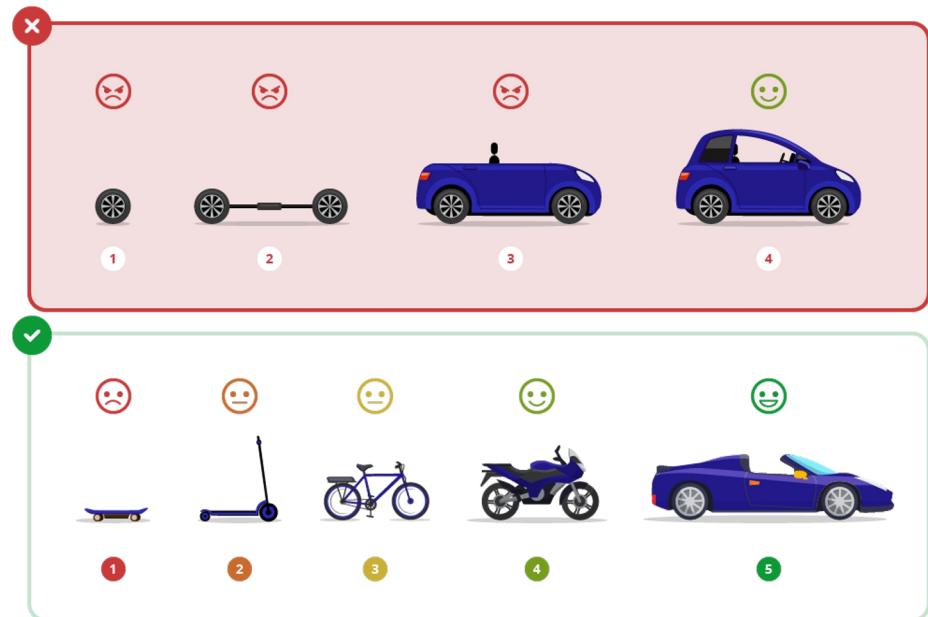
- Erste minimal funktionsfähige Iteration eines Produkts
- Testen einer Markthypothese
- Ausloten des brauchbaren Nutzens der Lösung



# Minimum Viable Product

*“The minimum viable product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort.”*

Eric Ries



<https://mlsdev.com/>

# Behaviour Driven Development

- Grundlage bietet das Verhalten eines Zielsystems von **aussen nach innen betrachtet**
- Definiert einen Test
- Wird meistens gegen ein Benutzerinterface durchgeführt um überprüft werden zu können
- Besteht aus einer simplen aber flexiblen Syntax

# Behaviour Driven Development

## Formulierung in sogenannten Stories (Geschichten)

- Rolle (Role)
- Funktionalität (Functionality)
- Geschäftsergebnis (Business Value)

**Syntax:** Wenn eine Menge von Vorbedingungen existiert und ein bestimmtes Ereignis eintritt, dann soll folgende Beobachtung gemacht werden können.

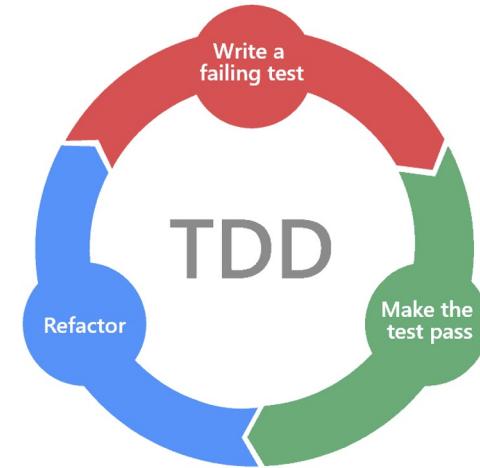
**Beispiel:** Als Benutzer <role> möchte ich eine Login-Eingabe <functionality> präsentiert bekommen, um mich im System anmelden <business value> zu können.

# Test Driven Development (TDD)

- Testet Funktionen eines Systems von **innen nach aussen**
- Es wird zuerst ein Test formuliert und danach die entsprechende Funktionalität definiert
- Man schreibt den Testcode für Programmcode, den man danach bekommen möchte

# Test Driven Development (TDD)

- Workflow für TDD
  - Schreiben des Tests und sehen wie er fehlschlägt (Write a failing test)
    - Beispiel: (testMyHelloWorldFunction() => Expected Result (Text: "Hello World" appears on Screen)
  - Schreiben von genug Programmcode um dafür zu sorgen, dass ein Test erfolgreich zurückkommt (Make the test pass)
  - Verbessern des Programmcodes falls ein Test fehlschlägt (Refactor)



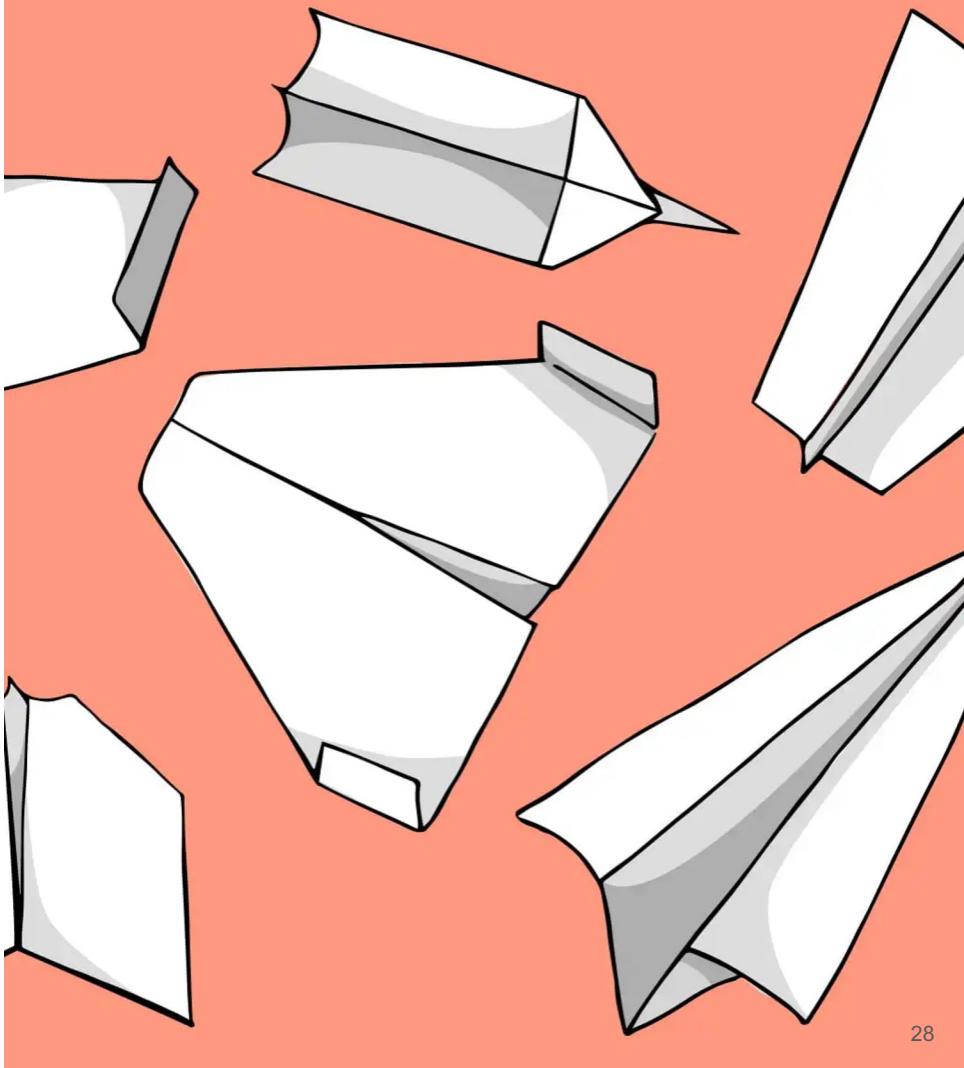
Der grosse Vorteil dieses Ansatzes ist es, dass er automatisiert werden kann.

# Pair Programming (PP)

- Ablauf
  - Zwei Entwickler arbeiten zusammen an einer Arbeitsstation
  - Ein Entwickler schreibt den Code
  - Der andere Entwickler überprüft und gibt Feedback
  - Regelmässiger Wechsel der Rollen (z.B. alle 20 min)
- Nutzen: Steigerung der Code Qualität
- Kritik des Chefs: “Warum soll ich zwei Leute für die Arbeit einer Person bezahlen?”

*Writing code is the easy part <> Debugging and maintaining is the hard part*

# Paper Plane Challenge



# Agiles Projektmanagement

## Die Scrum Methode



# Die Scrum Methode

- Scrum ist eine Management Framework für inkrementelle Produktentwicklung
- Geeignet für kleine selbstorganisierte Teams
- Definiert eine Struktur von Rollen, Meetings, Regeln und Lieferobjekten
- Definiert den Ablauf und die Dauer von Iterationen in sogenannten Sprints

*Leicht zu verstehen - schwer zu meistern*

# Ziele der Scrum Methode

- Höhere Produktivität
- Höhere Qualität des Produkts
- Reduzieren der Time to Market
- Höher Stakeholder Zufriedenheit
- Besseres Mitarbeiterklima

# Sprint

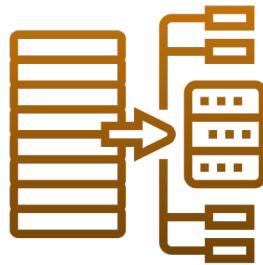
- Stellt eine Iteration in der Entwicklung dar mit verschiedenen Phasen
- Jeder Sprint hat ein klares Ziel
- Durchschnittlich werden Sprints auf ca. 2 Wochen Dauer ausgelegt



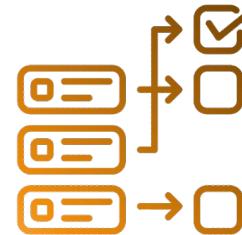
<https://startinfinity.com/>

# Scrum Artefakte

Product backlog



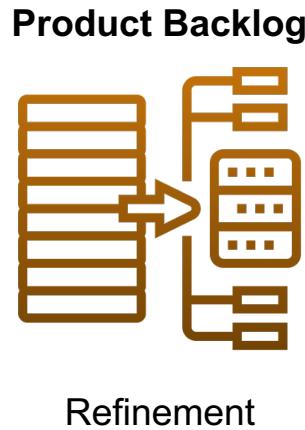
Sprint backlog



Done increment



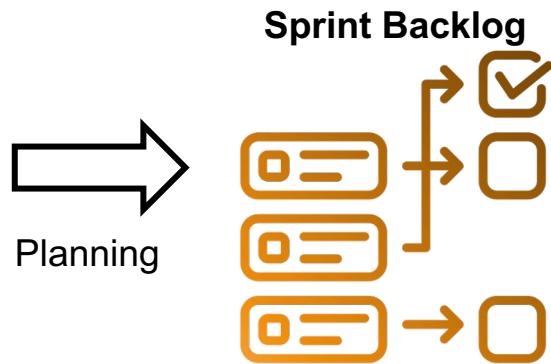
# Schritte in Scrum: Backlog Refinement



## Backlog Refinement

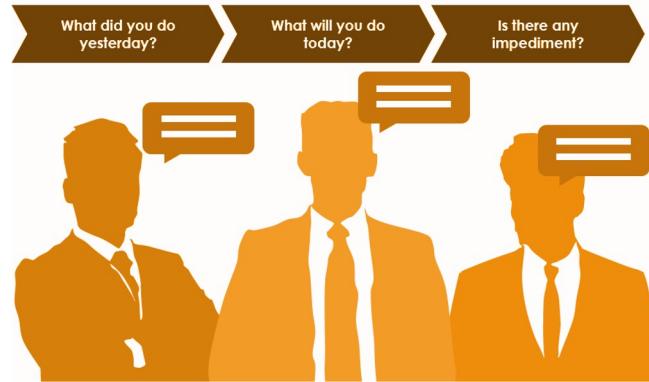
- Enthält alle Stories welche das Produkt umsetzen möchte
- Die Liste ist niemals abschliessend
- Das sog. Refinement ist dazu da diese Stories zu schärfen, zu priorisieren und offene Fragen zu klären
- Stories können hier auch verworfen werden
- Neue Stories können erschaffen werden

# Schritte in Scrum: Planning



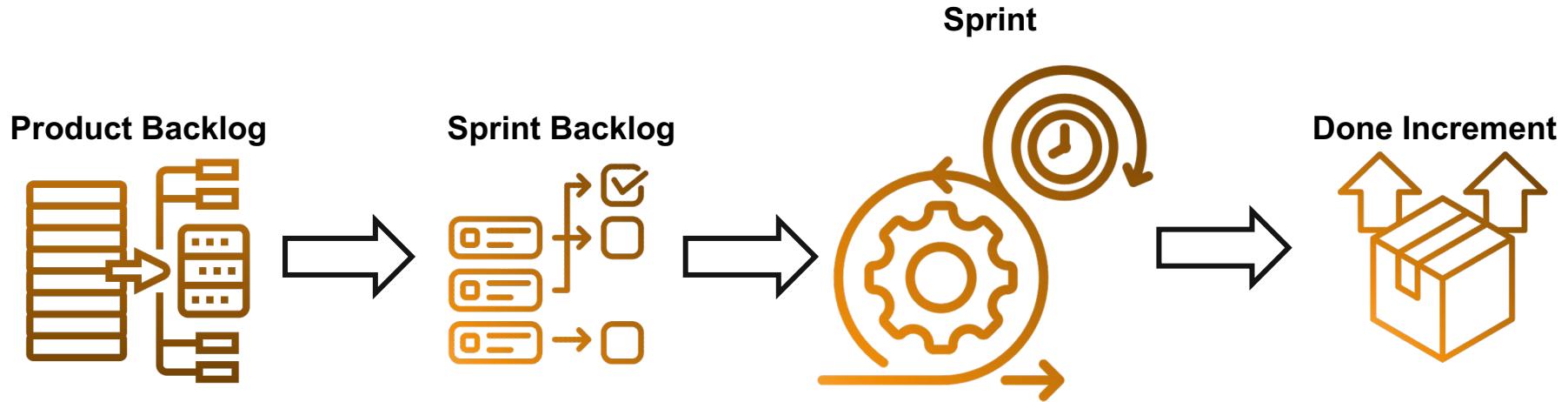
- Ist kleiner als das Product Backlog
- Hier gelangen alle Stories rein, die wir in einem zweiwöchigen Sprint erledigen wollen

# Schritte in Scrum: Daily Standup

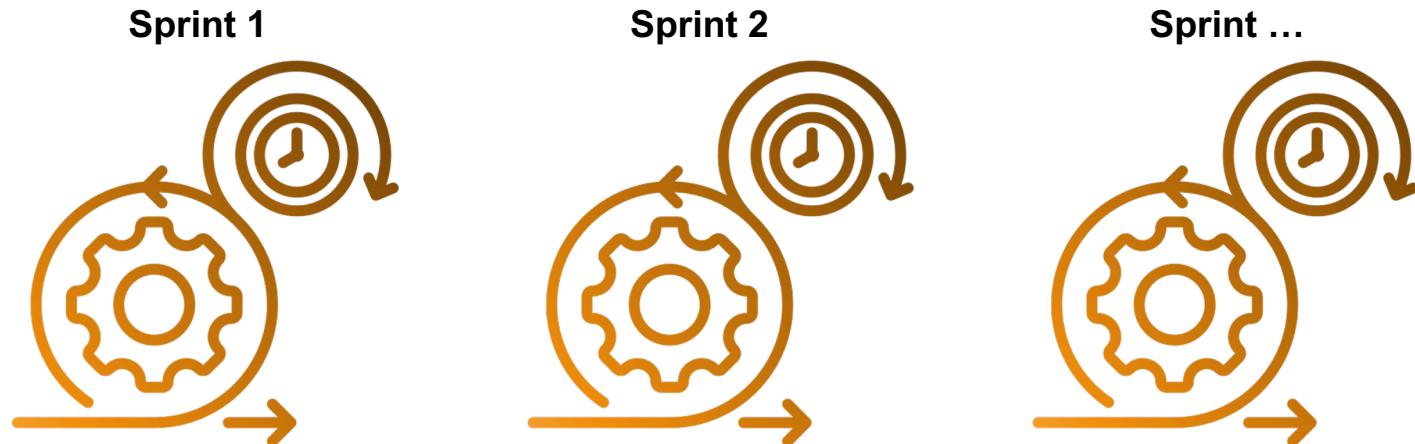


- Kurzer täglicher Austausch mit dem Team nicht länger 10 bis 15 Minuten
- Kernfragen des Daily Standup:
  - Was habe ich kürzlich gemacht?
  - Was habe ich heute vor?
  - Gibt es Hindernisse?

# Scrum Prozess: Ablauf einer Iteration



# Scrum Prozess: Iterationen



# Scrum Rollen

- Scrum Rollen sind ein Schlüsselement von Scrum (Kanban z.B. hat keine Rollen)
- Rollen einzunehmen erfordert Training und Erfahrung
- Wichtige Rollen wie Product Owner oder Scrum Master sollte nie ohne Training übernommen werden.
- Beispiel: Man beordert einen Projektmanager zur Rolle des Product Owners  
→ Das Ergebnis sind völlig unangemessene Verhandlungen über den Projekt-Scope

# Scrum Rollen

- Product owner
- Scrum master
- Scrum team



# Scrum Rollen: Product owner

- Ist kein Projektmanager
- Vertritt die Interessen des Geschäfts und definiert diese
- Formuliert und pflegt die Vision des Endprodukts
- Schlüsselentscheidungen bei Anforderungen
- Priorisierung des Product Backlogs
- Gibt das Ergebnis einer Iteration frei oder weist es zurück
- Entscheidet: Pivot vs. Persevere
- Hat kein direktes Bestimmungsrecht über den Ablauf innerhalb des Teams
- Kann nicht bestimmen wie viel des Product Backlogs in einem Sprint landet



# Scrum Rollen: Scrum Master

- Fördert und pflegt den Scrum Prozess
- Verfügt über viel Erfahrung in Scrum
- Fungiert als Coach für das Team
- Fördert eine selbstorganisierte Kultur innerhalb des Teams
- Vertritt das Team gegenüber Ansprüchen von oben
- Hilft beim Lösen von Hindernissen
- Schaut das Meetingzeiten eingehalten werden
- Hat keine Micromanagment Funktion er ist ein “Enabler” für das Team und in erster Linie dem Team verpflichtet
- Sammelt empirische Daten über den Fortschritt des Projektes und Stand der Arbeit



# Scrum Rollen: Scrum Team

- Interdisziplinär (Entwickler, Tester, Business analyst, Forscher, Fachspezialist)
- Selbstorganisiert und alle Mitglieder sind ebenbürtig
- Weisen sich selbst die Arbeit
- Ideale Grösse: 7 +/- 2 Personen
- Sollen sich regelmässig in denselben Räumen begegnen
- Vollzeit Mitglieder sofern möglich
- Verhandelt das sog. Commitment mit dem Product owner für den kommenden Sprint (Wir schaffen das)
- Entscheiden und tragen die Verantwortung für den Scope (Wieviel wird in diesem Sprint gemacht)



# Scrum Rollen: Product Manager vs Product Owner

- Product Manager
  - Überwacht das Budget und die Zeitplanung
  - Konzentriert sich auch die Einhaltung der Vorgaben der Geschäftsleitung
- Product Owner
  - Pflegt die Vision des Projekts und hält dem Team während dem Sprint den Rücken frei
  - Verhandelt den Scope des Projekts gegenüber der Geschäftsleitung

# Scrum Rollen: Project Manager vs Scrum Master

- Project Manager
  - Weist die Arbeit den Teammitgliedern zu
  - Legt die Zeiträume fest und sorgt für die Einhaltung der festgelegten Deadlines
  - Dokumentiert Risiken die durch Hindernisse entstehen
- Scrum Master
  - Coached das Team um den Fokus zu behalten auf dem momentan laufenden Sprint
  - Räumt Hindernisse aus dem Weg

# Scrum Rollen: Entwickler-Team vs Scrum-Team

- Entwickler-Team
  - Ein Team bestehend aus Softwareentwicklern oder Ingenieuren
- Scrum-Team
  - Ein interdisziplinäres Team wo es Entwickler, Spezialisten und weitere Fachbereiche geben kann

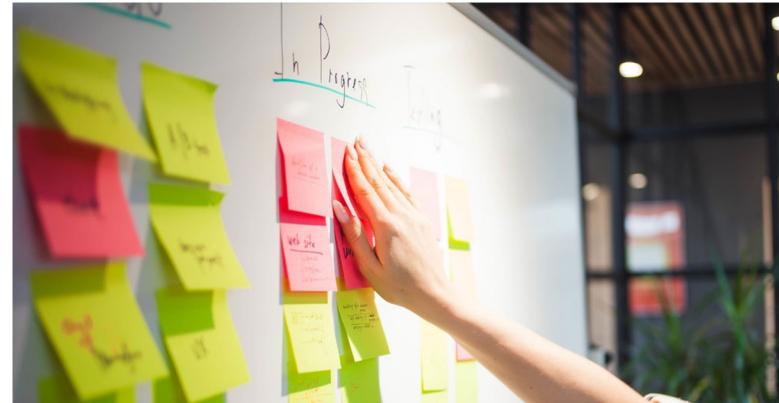
# Scrum Events

- Backlog Refinement
  - Priorisieren, befüllen, klären und verhandeln für alle möglichen bekannten Stories
- Sprint Planning
  - Erstellen des Sprint Backlogs, schätzen der Stories und Vereinbaren des Sprintziels
- Daily Standup
  - Täglicher Austausch über woran man arbeite, was als nächstes kommt und ob es Hindernisse gibt
- Sprint Review
  - Den vergangenen Sprint besprechen
- Sprint Retrospective
  - Besprechen des Ablaufs des vergangenen Sprints auf team-dynamischer Ebene. Was lief gut, was nicht? Gibt es Handlungsbedarf?

Dazu später mehr...

# Agile Planning Tools

# Das Kanban Board



# Software Lösungen



Azure Boards



Projects / Beyond Gravity

Board

Epics

GROUP BY Choices

Column	Items
TO DO	12
IN PROGRESS	4
IN QA	4
DONE	4

TO DO (12 items):

- Implement feedback collector (NUC-205)
- Bump version for new API for billing (NUC-206)
- Add NPS feedback to wallboard (NUC-214)
- Add analytics events to pricing page (NUC-209)
- Resize the images for the upcoming campaign (NUC-210)

IN PROGRESS (4 items):

- Update T&C copy with v1.9 from the writers guild in all products that have cross country compliance (NUC-213)
- Bump feedback icon version (NUC-214)
- Tech spike on new stripe integration with paypal (NUC-215)
- Change phone number field type to 'phone' (NUC-217)

IN QA (4 items):

- Adapt web app no new payments provider (NUC-346)
- Purchasing error - edit fields (NUC-354)
- Multi-dest search UI web (NUC-338)
- Shopping cart purchasing error - quick fix required. (NUC-354)

DONE (4 items):

- Quick booking for accomodations - web (NUC-336)
- Fluid booking on tablets (NUC-343)
- Fluid booking on tablets (NUC-343)
- Shopping cart purchasing error - quick fix required. (NUC-354)

# Praktischer Teil

Trello & Github Setup



# Agiles Projektmanagement

## User Stories



# User Stories

- Eine User Story stellt einen kleinen abgegrenzten Geschäftsnutzen dar
- User Stories sind vielfältiger als ein einfacher Anforderungskatalog
- Kernfragen einer User Story sind:
  - Für wen ist es?
  - Was wird benötigt?
  - Warum wird es benötigt
- Inhalt einer User Story
  - 1) Eine gute Beschreibung (Description) des Geschäftsnutzen
  - 2) Annahmen, Abgrenzungen und Details werden formuliert.
  - 3) Eine klar formulierte Akzeptanzkriterien (“Definition of done”)

# User Stories: Beschreibung

- Syntax gemäss BDD (Behaviour Driven Development)
  - Als ein <role>  
möchte ich <functionality>  
so dass <business value>
  - **Beispiel:** Als Benutzer <role> möchte ich eine Login-Eingabe  
<functionality> präsentiert bekommen, um mich im System anmelden  
<business value> zu können.

# User Stories: Annahmen, Abgrenzungen & Details

## Beispiele

- Welche Dateiformate werden als Grundlage angenommen. CSV, XML, JSON
- Ein Benutzerinterface für die Dateneingabe existiert bereits und erfüllt x-Anforderungen
- Ein Endpunkt für die Konsumation der Daten existiert beim Anbieter X oder ist an folgender Stelle dokumentiert
- In diesem Use Case wird Anforderung x nicht berücksichtigt

# User Stories: Akzeptanzkriterien “Definition of done”

- Definiert ein messbares oder bewertbares Ergebnis
- Kann durch eine Syntax formuliert werden
  - Es sei <Vorbedingung>  
wenn <Ereignis geschieht>  
dann <Erwartetes Ergebnis>
  - **Beispiel**  
Der Benutzer befindet sich im Warenkorb, wenn er den “Zur Kasse”-Button betätigt dann wird der Dialog für die Zahlungsangaben präsentiert

# User Stories: Beispiel

## Beschreibung

Als Marketingmanager brauche ich eine Liste von Kundennamen und E-Mails, so dass ich diese mit Angeboten informieren kann.

## Annahmen, Abgrenzungen & Details

- Wir verwalten Kundendaten mit E-Mails in einer Datenbank
- Benutzer sind für Angebote angemeldet (Newsletter)

## Akzeptanzkriterien

Es seien 100 Kunden in der Datenbank und 90 haben sich für E-Mail Angebote angemeldet dann solle ich eine Liste von 90 Kunden E-Mails zurückbekommen

# User Stories: Basiskriterien (INVEST)

Bill Wake's INVEST: Eine User Story sollte sein:

- **I**ndependent (unabhängig)
- **N**egotiable (verhandelbar)
- **V**aluable (wertvoll)
- **E**stimable (schätzbar)
- **S**mall (klein)
- **T**estable (testbar)

# User Stories: Epics

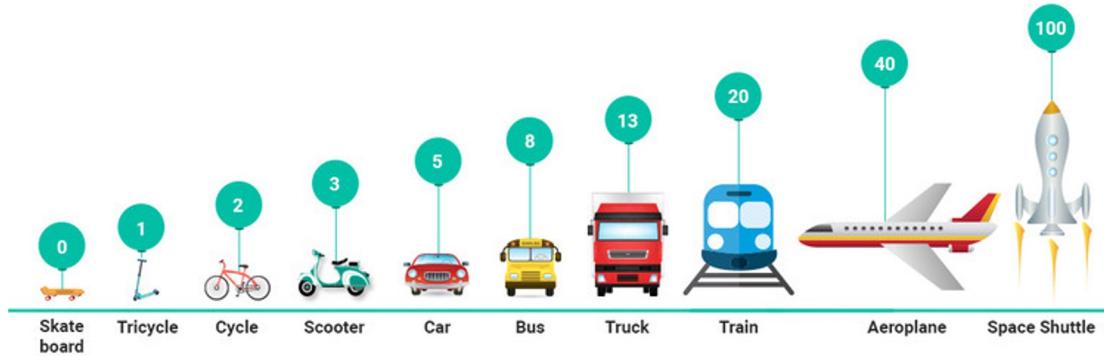
- Was ist ein Epic?
  - Big Ideas
  - Eine User Story die sicherlich mehr als ein regulärer Sprint umfassen wird
  - Eine User Story die als ganzes nicht direkt geschätzt werden kann
- Wann nutzen wir Epics?
  - Wenn wir Anforderungen haben die viele Teilaspekte betreffen und es offene Fragen gibt
  - Epics werden typischerweise im späteren Verlauf in kleiner User Stories heruntergebrochen
  - Ein gesamtes Epic landet nie als Ganzes in einem Sprint

# User Stories: Schätzungen

- Schätzungen sollten wenn möglich **nicht** in absoluten Einheiten wie Stunden oder Minuten betrachtet werden
- Schätzungs-Einheiten sind Verhandlungssache des Teams und nicht ein Kontrollmechanismus für Projektmanager
- Schätzungen mit Story Points
  - Eine Einheit wo jedes Teammitglied eine ungefähre Vorstellung hat wieviel Arbeit es bedeutet
  - Geeignet für eingespielte erfahrene Teams
  - Der Aufwand für einen Story Point muss bereits über längere Zeit über das gleiche Projekt im gleichen Team “kalibriert” werden
- T-Shirt Größen
  - S, M, L, XL
- Fibonacci
  - 1, 2, 3, 5, 8, 13, 21

# User Stories: Story Points

Story Points sind immer relative Größen den vorhergehenden Einschätzungen und sind nie direkt quantifiziert sondern entwickeln sich entlang der Leistungsfähigkeit des Teams



# User Stories: Story Größen

- Eine User Story sollte klein genug sein damit sie im Sprint implementiert und getestet werden kann
- Große Stories müssen in kleinere Stories heruntergebrochen werden
- Manchmal gibt es auch Unteraufgaben zu einer Story
- Anti Patterns:
  - Wanduhr Zeit Schätzungen
  - Gantt Charts

# Agiles Projektmanagement

## Scrum Events

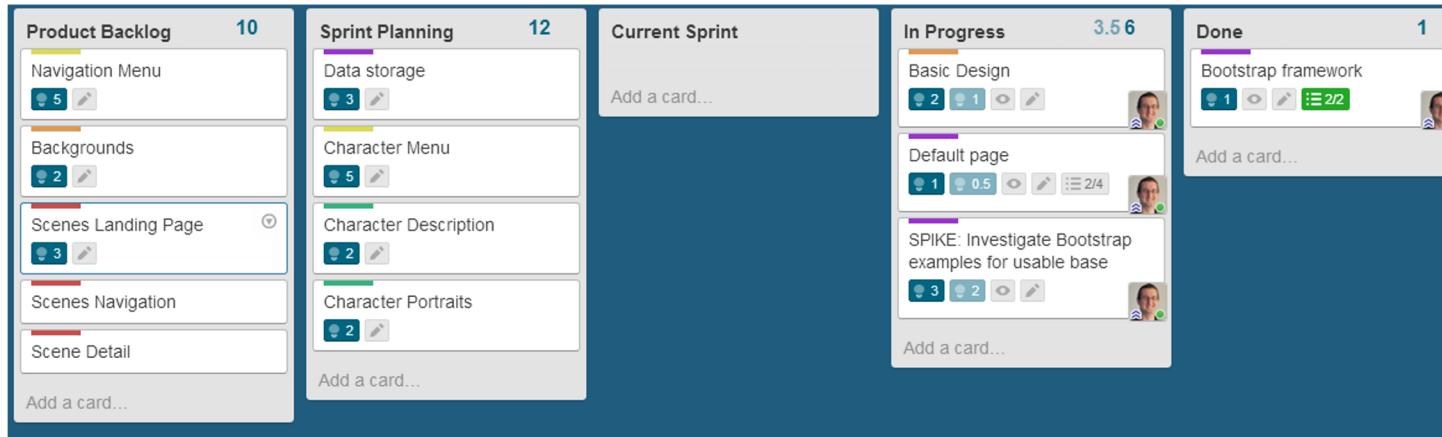


# **Product Backlog & Refinement**

# Product Backlog: Erstellen

- Enthält alle User Stories (auch Issues genannt) die noch nicht implementiert wurden
- Die User Stories werden top-down in eine Rangordnung gebracht je nach Wichtigkeit oder Business Value
- Je detaillierter eine User Story ist so weiter oben “sollte” sie sein

# Product Backlog: Beispiel



# Product Backlog: Erstellen

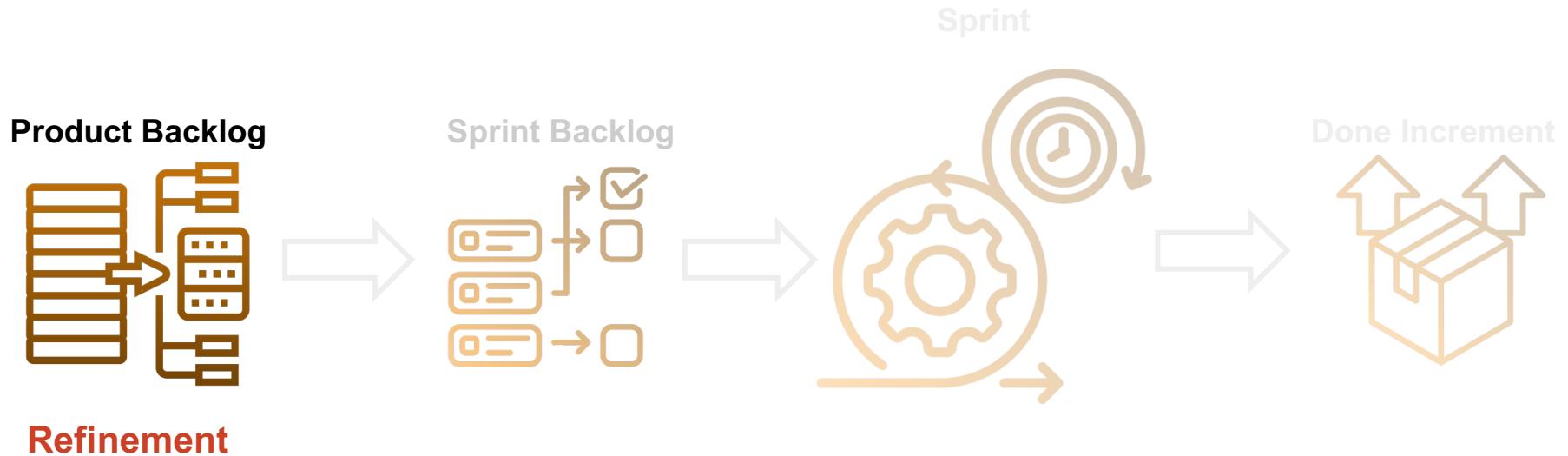
## Neue User Stories aus der Businessanalyse

1. Als Benutzer der Website brauche ich eine Anzeige wieviele Gegenstände in meinem Warenkorb sind
2. Als Benutzer möchte ich mehrere Warenkörbe verwalten können
3. Als Benutzer möchte ich Feedback zu einzelnen Produkten abgeben können
4. Als Benutzer brauche ich einen Login-Button wenn ich ausgeloggt bin
5. Als Administrator muss ich alle registrierten Benutzer einsehen können
6. Als Benutzer muss ich an einem Indikator feststellen können ob ich eingeloggt oder ausgeloggt bin
7. Als Benutzer möchte ich in einem separaten Segment die neuesten Produkte präsentiert bekommen



## Product Backlog

# Product Backlog: Refinement Meeting



# Product Backlog: Refinement Meeting

- Ein zentrales Meeting im agilen Entwicklungsprozess
- Priorisierung der User Stories
- Komplexe User Stories in Kleinere herunterbrechen damit sie leichter priorisiert werden können
- User Stories können bei Unklarheiten geschärft werden
- Kleine User Stories mit hohem Detailgrad landen oben auf in der Rangliste
- Primäres Ziel: Ein priorisiertes (ranked) Product Backlog mit User Stories die bereit für die Umsetzung sind (Ready for Sprint)
- Zeitraum: Sollte mindestens einmal pro Sprint stattfinden

# Product Backlog: Refinement Meeting

## Teilnehmer

- Product Owner
  - Schreibt User Stories gemäss dem geforderten Business Value und gibt Antwort an das Team bei Unklarheiten
- Scrum Master
  - Unterstützt den Product Owner bei der Priorisierung und gewährleistet die Interessen des Teams
- Scrum Team (Optional)
  - Teile des Teams (Architekten oder Lead Developers) geben Antwort auf Komplexität der User Stories und helfen dem Product Owner das Product Backlog zu schärfen und eine erste Schätzung abzugeben

# Product Backlog: Triage

- Wenn neue User Stories erstellt werden sollten diese nicht unbedingt gleich im Product Backlog landen (Evtl. sind die Anforderungen aus dem Business unklar)
- Diese Triage ist ebenfalls Teil der Aufgabe des Product Owners
- User Stories sollten niemals in der Triage hängen bleiben. Entweder sie werden priorisiert im Product Backlog oder sie werden zurückgewiesen (Rejected)

# Product Backlog: User Story komplettieren

<p><b>Beschreibung:</b> Als Benutzer der Website brauche ich eine Anzeige wieviele Gegenstände in meinem Warenkorb sind</p> <p><b>Annahmen &amp; Details:</b> Eine Warenkorbverwaltung existiert und wir können die Anzahl darin abfragen. Die Anzeige wird soll in Form einer klar ersichtlichen Zahl auf einem Warenkorbsymbol erfolgen</p> <p><b>Akzeptanzkriterien:</b> Wenn ich ein neues Produkt in den Warenkorb legen mittels dem Button “Dem Warenkorb hinzufügen” dann erhöht sich die Zahl der Anzeige um die Anzahl der hinzugefügten Gegenstände</p>	<p><b>Assignee:</b></p> <p><b>Labels:</b></p> <p><b>Milestones:</b></p> <p><b>Estimate:</b></p>
---	---

# Labels für User Stories

- Labels helfen die Arbeit und den Fortschritt zu visualisieren
- Ist oft farbenfroh um Dringlichkeit zu vermitteln
- Helfen dabei zu sehen wo Klärungs- oder Handlungsbedarf besteht
- Sind eine sehr individuelle Sache und nicht als allgemeingültige Menge zu sehen
- User Stories können mehrere Labels bekommen

Beginner

Bigger project

Bug

Critical

Docs

Enhancement

Intermediate

Question

Refactoring

Tests

# Labels für User Stories

Beispiel User Stories: Welche Labels würden hier wo passen?

1. Als Benutzer der Website brauche ich eine Anzeige wieviele Gegenstände in meinem Warenkorb sind
2. Als Benutzer möchte ich mehrere Warenkörbe verwalten können
3. Als Benutzer möchte ich Feedback zu einzelnen Produkten abgeben können
4. Als Benutzer brauche ich einen Login-Button wenn ich ausgeloggt bin
5. Als Administrator muss ich alle registrierten Benutzer einsehen können
6. Als Benutzer muss ich an einem Indikator feststellen können ob ich eingeloggt oder ausgeloggt bin
7. Als Benutzer möchte ich in einem separaten Segment die neuesten Produkte präsentiert bekommen

Beginner

Bigger project

Bug

Critical

Docs

Enhancement

Intermediate

Question

Refactoring

Tests

# Labels für User Stories

**Beschreibung:** Als Benutzer möchte ich Feedback zu einzelnen Produkten abgeben können

**Annahmen & Details:** ...

**Assignee:**

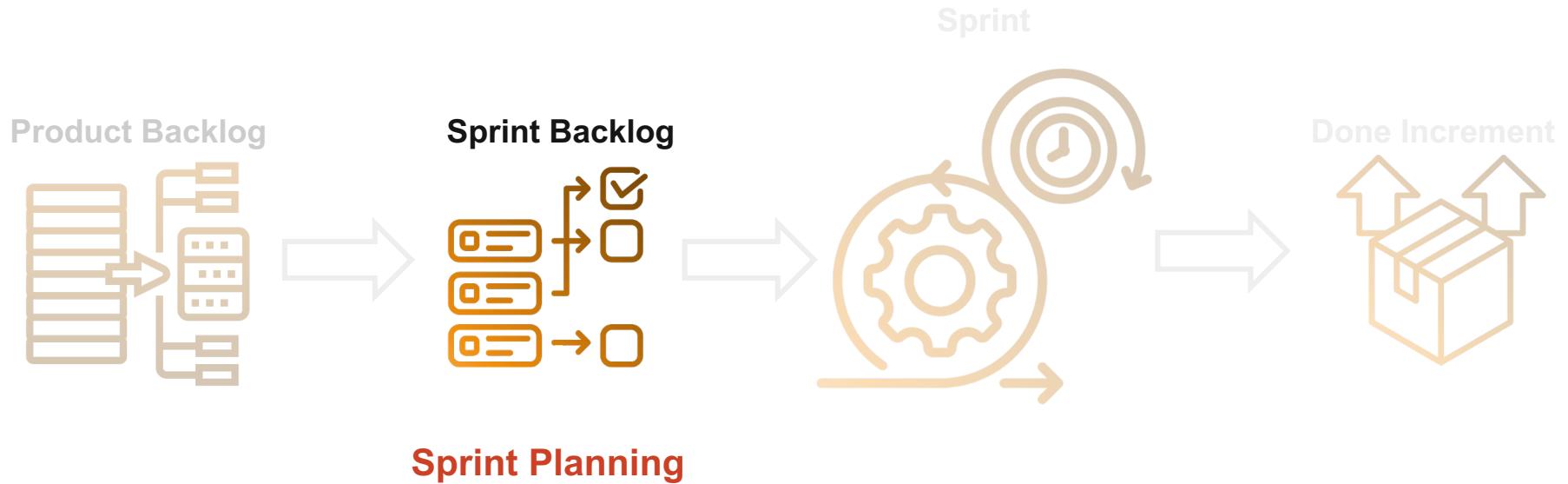
**Labels:**  Bigger project

**Milestone:**

**Estimate:**

## Technical Debt

# Sprint Planning



# Sprint Planning Meeting

- Generiert das Sprint Backlog für den kommenden Sprint
- Hier wird definiert was in der nächsten Iteration (typischerweise 2 Wochen) erreicht werden soll
- Je detaillierter das Refinement Meeting war umso schneller geht das Planning Meeting
- Teilnehmer:
  - Product Owner
  - Scrum Master
  - Scrum Team

# Sprint Planning Meeting

- Sprint Ziel definieren: Der Product Owner erklärt das Ziel welches diesen Sprint erreicht werden soll und welche User Stories aus dem Backlog aus seiner Sicht zielführend wären.
  - Manchmal erstellt der Product Owner auch einen Milestone der dieses Ziel beschreibt und terminiert auf des Endes kommenden Sprints
- Sprint Backlog: Die Mitglieder des Scrum Teams nehmen User Stories oben vom Backlog und bewegen sie in das Sprint Backlog
  - Sie fügen Labels hinzu wo sinnvoll
  - Sie geben die belastbaren Schätzungen ab. z.B. in Story Points oder T-Shirt Größen etc.
  - Dieser Vorgang ist Verhandlungssache des Teams
- Wenn die Team velocity (Geschwindigkeit) erreicht wurde werden keine neuen Stories hinzugefügt
- Das Team gibt am Ende des Plannings ein sog. Commitment (Bekenntni) ab
  - Dies soll gewährleisten, dass alle involvierten Personen mit dem Stand des Sprint Backlogs einverstanden sind

# Sprint Planning: Team velocity

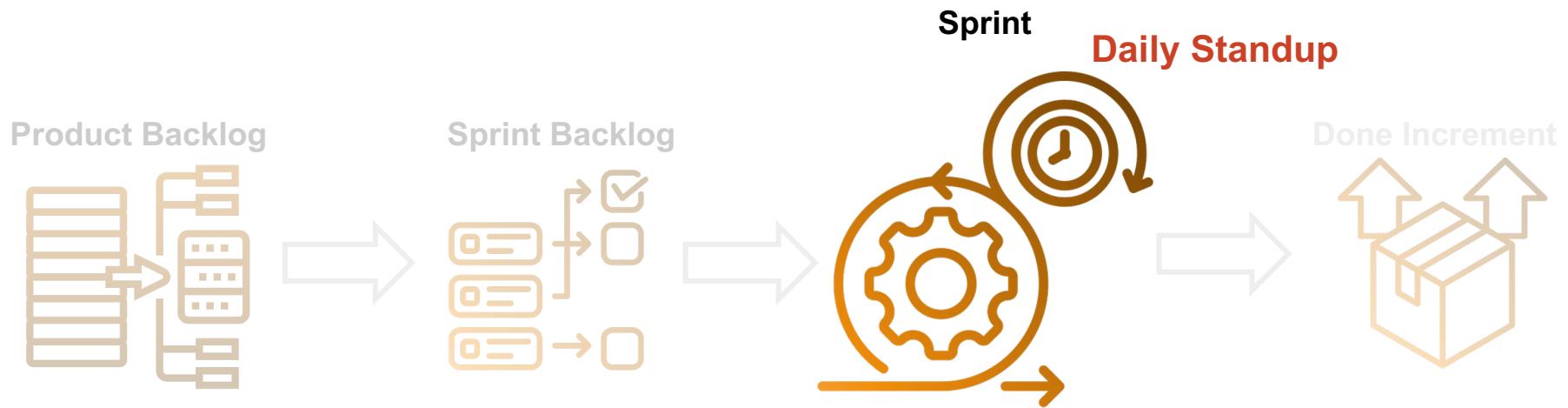
- Alle Story Points aller User Stories im Sprint Backlog zusammengenommen ergeben den Gesamtaufwand für den Sprint
- Dies muss am Anfang vom Team über mehrere Sprints ausgelotet werden
- Kann sich über den Zeitraum des Projekts verändern
- Teams können nicht direkt verglichen werden, da Story Points keine direkt messbare Grösse in Zeit darstellen dürfen.
- Medium Size ist niemals die gleiche Grösse über mehrere Teams

# Sprint Backlog: User Story

<p><b>Beschreibung:</b> Als Benutzer der Website brauche ich eine Anzeige wieviele Gegenstände in meinem Warenkorb sind</p> <p><b>Annahmen &amp; Details:</b> Eine Warenkorbverwaltung existiert und wir können die Anzahl darin abfragen. Die Anzeige wird soll in Form einer klar ersichtlichen Zahl auf einem Warenkorbsymbol erfolgen</p> <p><b>Akzeptanzkriterien:</b> Wenn ich ein neues Produkt in den Warenkorb legen mittels dem Button “Dem Warenkorb hinzufügen” dann erhöht sich die Zahl der Anzeige um die Anzahl der hinzugefügten Gegenstände</p>	<p><b>Assignee:</b></p> <p><b>Labels:</b>  Intermediate  Enhancement</p> <p><b>Milestones:</b> Sprint 1</p> <p><b>Estimate:</b> 4 - small</p>
---	---

# Daily Standup

# Daily Standup

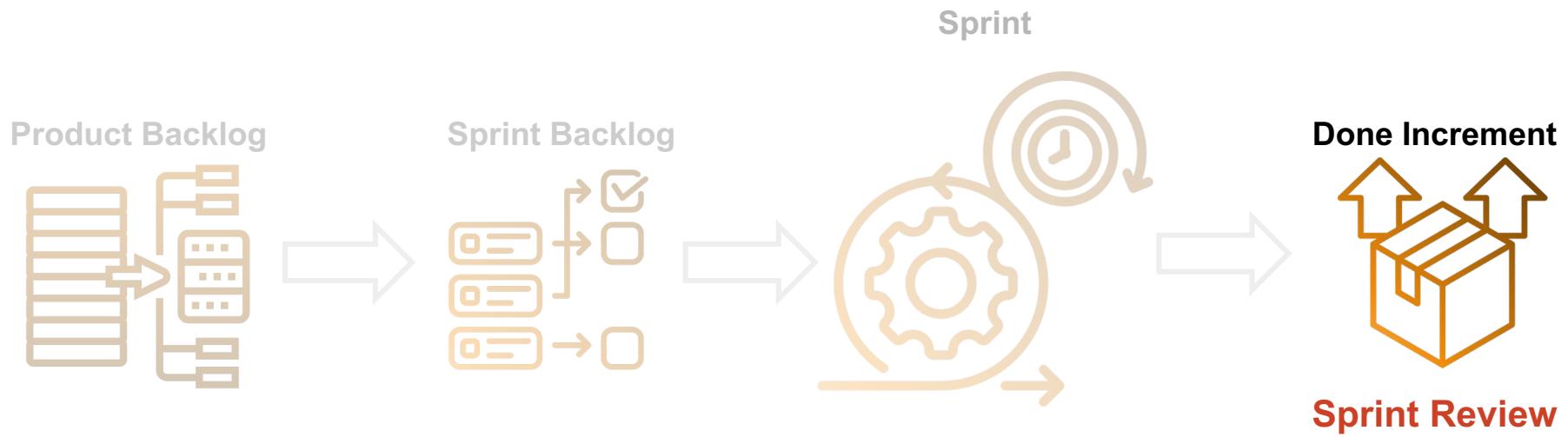


# Daily Standup Meeting

- Niemals länger als 15 Minuten
- Findet täglich statt
- Jedes Teammitglied erklärt sich kurz anhand der folgenden drei Fragen:
  - Woran habe ich gestern gearbeitet?
  - Woran möchte ich heute arbeiten?
  - Gibt es Hindernisse?
- Wer ist dabei?
  - Scrum Master, Scrum Team, Product Owner (Optional)
- Es ist kein Status-Meeting, d.h. der Product Owner kann nur zur Klärung von Fragen sich bemerkbar machen
- Der Scrum Master moderiert und hilft bei Hindernissen
- Detailfragen werden nicht im Daily Standup geklärt (15 Minute Timebox)

# Sprint Review

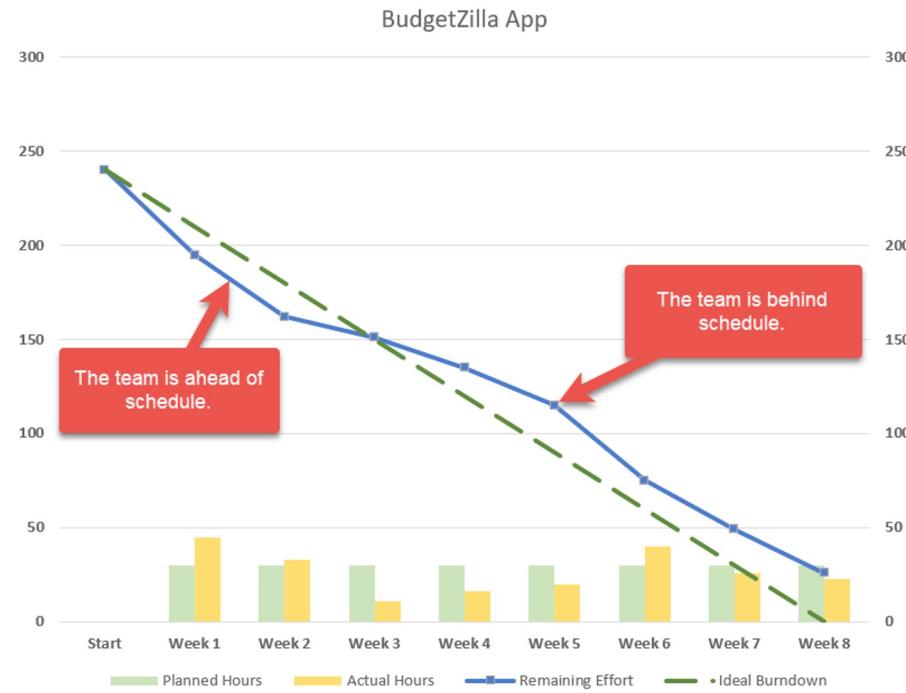
# Sprint Review



# Sprint Review

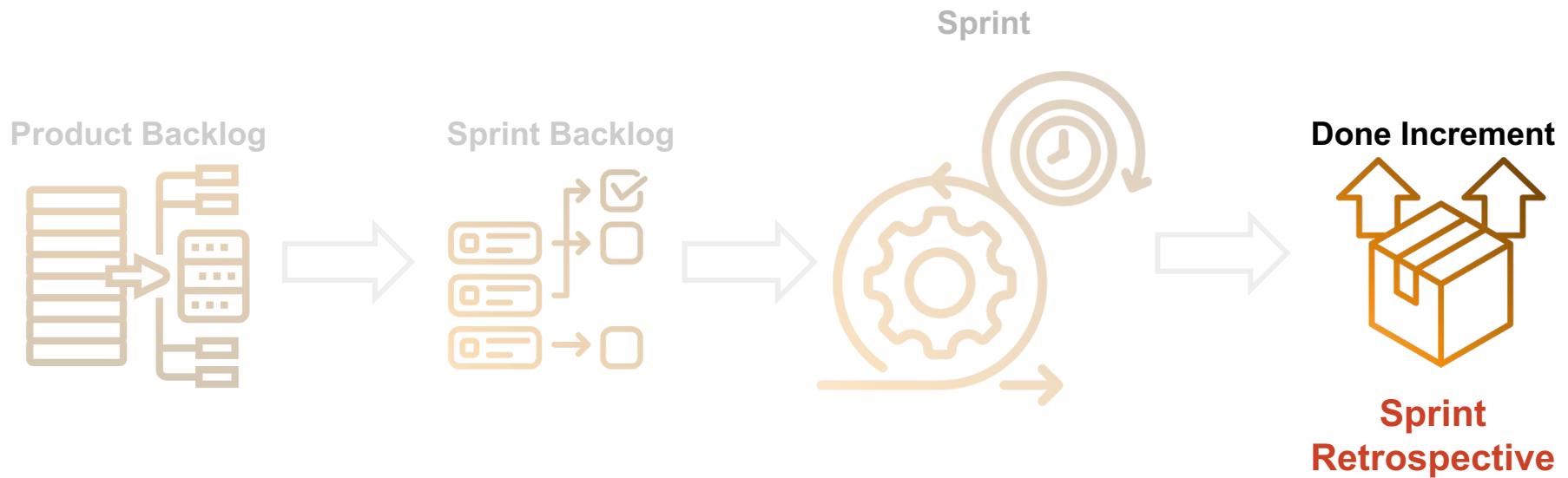
- Findet nach dem Abschluss eines Sprints statt und ist eine Live Demonstration
- Product Owner nimmt die User Stories die bearbeitet wurden aufgrund der Akzeptanzkriterien ab
- User Stories die auf “Done” sind und akzeptiert wurden landen auf “Closed”
- Teilnehmer: Alle! (Kunden oder sonstige Stakeholder inklusive wenn erwünscht)
- Es wird Feedback gesammelt und evtl. schon neue User Stories daraus abgeleitet
- Nicht akzeptierte User Stories werden ebenfalls geschlossen aber es muss für jede dieser User Stories eine Neue erstellt werden welche die angepassten Akzeptanzkriterien enthält und eine Label dass die User Story markiert
- Die Sprint Velocity soll durch zurückgewiesene User Stories nicht beeinträchtigt werden

# Sprint Review: Burndown Chart



# Sprint Retrospective

# Sprint Retrospective



# Sprint Retrospective

- Reflexion des Teams über den vergangenen Sprint “Health Check”
- Muss in einer Atmosphäre stattfinden, die es dem Team erlaubt offen zu sprechen
- Teilnehmer: Scrum Master, Scrum Team
- Kernfragen:
  - Was lief gut? (Went well)
  - Was lief schlecht? (Went not well)
  - Was können wir anders machen? (Learning)
  - Optional: “Blumenstrauss”
- Es werden Massnahmen abgeleitet “Action Points” um beim nächsten Sprint die Problempunkte zu adressieren

# Sprint Retrospective