Pull Request:
Ein Pull Request (auch als Merge Request bezeichnet, abhängig von der verwendeten Versionskontrollplattform) ist ein Vorgang, bei dem ein Entwickler Änderungen, die er in einem seiner Branches vorgenommen hat, in den Haupt- oder Zielbranch des Projekts einbringen möchte. Dieser Vorgang findet in verteilten Versionskontrollsystemen wie Git statt. Ein Pull Request wird normalerweise verwendet, um anderen Entwicklern oder dem Projektleiter die Möglichkeit zu geben, die vorgeschlagenen Änderungen zu überprüfen, Feedback zu geben, Diskussionen zu führen und gegebenenfalls Änderungen anzufordern. Der Entwickler, der den Pull Request erstellt hat, kann die vorgeschlagenen Änderungen weiterhin aktualisieren, während das Feedback eingeht. Wenn alle Beteiligten mit den Änderungen zufrieden sind, kann der Pull Request gemerged werden.

Merge:
Der Merge ist der Vorgang, bei dem die Änderungen aus einem Branch in einen anderen Branch übernommen werden. In Bezug auf einen Pull Request bedeutet dies, dass die vorgeschlagenen Änderungen aus dem Quellbranch in den Zielbranch zusammengeführt werden. Nachdem der Pull Request überprüft wurde und Änderungen vorgenommen wurden, um das Feedback zu berücksichtigen, kann der Merge-Prozess gestartet werden. Dabei werden die Änderungen aus dem Quellbranch in den Zielbranch integriert. Dies kann automatisch erfolgen, wenn die Versionskontrollplattform über entsprechende Funktionen verfügt, oder manuell, indem der Entwickler den Merge-Vorgang ausführt. Sobald der Merge abgeschlossen ist, sind die Änderungen offiziell im Zielbranch enthalten und für alle Projektmitglieder sichtbar.

Azure DevOps bietet verschiedene Tools und Funktionen, die ein Scrum-Team bei den Reviews und Retrospektiven unterstützen können. Hier sind einige wichtige Tools, die dabei helfen können:

1. Azure Boards: Azure Boards ist ein agiles Projektmanagement-Tool, das die Planung, Verfolgung und Diskussion von Arbeitselementen ermöglicht. Es bietet Funktionen wie das Erstellen von Arbeitsaufgaben, das Zuweisen von Aufgaben an Teammitglieder, das Hinzufügen von Kommentaren und Diskussionen sowie das Verfolgen des Fortschritts der Arbeitselemente. In den Reviews können Teammitglieder den aktuellen Stand der Arbeit überprüfen und Feedback geben.
2. Azure Repos: Azure Repos ist eine Versionskontrollplattform, die Funktionen wie Git-Repositories und Team Foundation Version Control (TFVC) bietet. Durch die Integration mit Azure Boards können Teammitglieder Commits, Pull-Anfragen und Code-Reviews durchführen. Diese Funktionen können während der Reviews genutzt werden, um den Code zu überprüfen und Feedback zu geben.
3. Azure Pipelines: Azure Pipelines ist ein Continuous Integration/Continuous Deployment (CI/CD)-Tool, das die Automatisierung von Build-, Test- und Bereitstellungsprozessen ermöglicht. Es unterstützt verschiedene Plattformen und Programmiersprachen. Während der Review- und Retrospektivsitzungen können Teammitglieder die Ergebnisse der automatisierten Tests überprüfen und Feedback geben.
4. Azure Test Plans: Azure Test Plans ist ein Tool, das das Testmanagement unterstützt. Es ermöglicht das Erstellen, Ausführen und Verfolgen von Tests sowie das Erfassen von Testergebnissen. Teammitglieder können während der Reviews Feedback zu den Testergebnissen geben und Testfälle aktualisieren.
5. Azure Artifacts: Azure Artifacts ist ein Tool für das Paketmanagement und ermöglicht das Speichern und Verwalten von Paketen, Bibliotheken und Artefakten. Während der Reviews können Teammitglieder auf die benötigten Artefakte zugreifen und Feedback geben.

Sprint Review

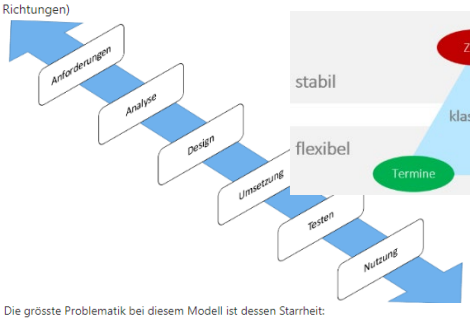
Die Ergebnisse des fertigen Sprints werden im **Sprint Review** dem Kunden (inkl. Benutzern) vorgestellt. Teilnehmer sind also alle Teammitglieder und der Kunde, die Leitung liegt beim Product Owner. Es werden alle PBI's (User Stories) nacheinander abgehandelt. Wichtig ist es, zuerst die Akzeptanzkriterien in Erinnerung zu rufen, damit keine Missverständnisse entstehen. Es werden nur *fertiggestellte und funktionierende* PBI's gezeigt! Der Kunde kann durchaus interaktiv involviert werden, um die Akzeptanz möglichst hoch zu halten.
Falls beim Sprint Review Differenzen zu Tage treten welche Anpassungen erfordern, werden diese in den Product Backlog aufgenommen und entsprechend der Dringlichkeit priorisiert. Oft kommen neue Ideen ins Spiel, dies sollte nicht als Kritik verstanden werden! Bei dieser Gelegenheit sollte auch überprüft werden, ob das Projekt zeitlich und kostenmässig im Rahmen geblieben ist.
Am Ende des Meetings wird gemeinsam besprochen was als nächstes zu tun ist, daraus ergibt sich ein aktueller, verbindlicher **Product Backlog**.
Zu diesem Meeting gibt es keine Übung in diesem Moment. Im praktischen Teil wird dieses dann geprobt.

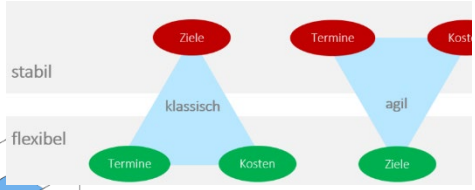
Sprint Retrospective

Einen "internen Abschluss" findet der Sprint in der sogenannten **Sprint Retrospective**, eine Besprechung, in der nur das Scrum Team zusammensitzt. Ziel ist eine offene Kommunikation der Erfahrungen aus dem Sprint. Es wird die Möglichkeit geboten Dinge zu nennen, die beim nächsten Sprint verbesserungswürdig erscheinen. Der Scrum Master nimmt die vom Team als gutgeheissenen Punkte in seine Hindernisliste (Impediment Backlog) auf und kümmert sich um Lösungen. Alle Teammitglieder sollten Einsicht in den Fortschritt dieser Verbesserungen haben, da sonst die Gefahr bestehtdass die Sprint Retrospective nicht mehr ernst genommen wird. Dabei ist sie die Möglichkeit zur Verbesserung der Umstände, Arbeitsqualität, Produktequalität usw.

Als Vergleich: Die Lineare Methode

Obwohl heute nur noch wenig eingesetzt, muss das Wasserfallmodell als schlechtes Beispiel einer Projektmethode herhalten. Beim Wasserfallmodell gibt es einen Durchlauf, welcher die verschiedenen Phasen vom Entwurf bis zur Nutzung einer Software abbildet. Ursprünglich gab es nur einen Weg „nach vorne". Später wurde das Modell angepasst, so dass man auch wieder in die vorherige Phase zurück kann bei Misserfolg. (Daher Pfeile in beiden Richtungen)

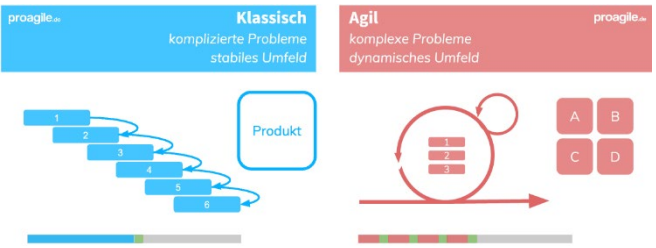




Die grösste Problematik bei diesem Modell ist dessen Starrheit:

- Es gibt nur eine Testphase innerhalb der gesamten Projektlaufzeit!
- Anpassungen oder Kundenwünsche während des Projekts bedeuten „gehe zurück zum Anfang"
- Man kann das Resultat erst am Schluss nutzen

Daraus ergeben sich erhöhte Risiken, welche für die Informatikwelt bis heute einen Imageschaden bedeuten. (zu spät, zu teuer...)



Vergleich aus Kundensicht

Bei klassischen Projektmethoden sind die Ziele starr, es muss alles erreicht werden. Der zeitliche und finanzielle Horizont ist variabel, wodurch Projekte verspätet und/oder verteuert werden können.

Bei agilen Projektmethoden werden Preis- und Zeithorizont fixiert, dafür die Ziele variabel gehalten. Es kann also sein, dass der Kunde mehr oder weniger für sein Geld erhält.

Nehmen Sie nochmals die Projektbeschreibung Ihres Teams zu Hand und kontrollieren, ob Ihre Geschäftsziele ersichtlich sind darin! Wenn nicht, passen Sie die Inhalte entsprechend an.

Aus Kostensicht, sollte durch die agile Vorgehensweise ein Vorteil gegenüber dem Wasserfallmodel entstehen. Änderungen mitten in der Umsetzungsphase fallen weniger ins Gewicht, weil diese mittels der Sprints einfacher ins laufende Projekt eingeführt werden können. (Prinzip der Agilität)

Komplex ist eine Arbeit immer dann, wenn z.B. viele unbekannte Aufgaben mit sich ständig verändernden Parametern zu lösen sind. Die Entwicklung einer neuen Software ist beispielsweise sehr komplex, da sich Anforderungen im Projektverlauf häufig verändern können. Ein linearer Projektplan wäre in einem komplexen Projektumfeld schnell wieder veraltet.

Komplizierte Aufgaben können mit Wissen gelöst werden. Auch unbekannte Variablen, die sich berechnen lassen, gehören zu den komplizierten Anteilen einer Arbeit. Soll ein Softwareentwickler z.B. eine neue Programmiersprache lernen, handelt es sich um eine komplizierte Aufgabe, die er mit Zeit und ausreichend Übung lösen kann. Diese Aufgabe ist plan- und steuerbar.

Wir erschließen bessere Wege, Software zu entwickeln,
indem wir es selbst tun und anderen dabei helfen.
Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

Individuen und Interaktionen mehr als Prozesse und Werkzeuge
Funktionierende Software mehr als umfassende Dokumentation
Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
Reagieren auf Veränderung mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden,
schätzen wir die Werte auf der linken Seite höher ein.

Wir folgen diesen Prinzipien:

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.

Ein Scrum Projekt beginnt mit den Anforderungen an die auszuliefernde Software. Diese Anforderungen werden als "**User Stories**" festgehalten. Bei der Erarbeitung der Software werden iterativ sogenannte **Sprints** durchgeführt. Ein Sprint hat eine vordefinierte Länge von ca. 2-8 Wochen. **Nach jedem Sprint ist die Software praktisch lauffähig!** Nach mehreren Sprints wird eine Major Version erstellt.

Hauptversionsnummer
(englisch: *major release*) indiziert meist äußerst signifikante Änderung am Programm – zum Beispiel, wenn das Programm komplett neu geschrieben wurde (zum Beispiel GIMP 2.x nach der Version 1.x) oder sich bei Bibliotheken keine Schnittstellenkompatibilität aufrechterhalten lässt.

Nebenversionsnummer
(englisch *minor release*) bezeichnet meistens die funktionale Erweiterung des Programms.

Revisionsnummer
(englisch *patch level* oder *micro release*) enthält meist Fehlerbehebungen.

Buildnummer
(englisch *build number*) kennzeichnet in der Regel den Fortschritt der Entwicklungsarbeit in Einzelschritten, wird also zum Beispiel bei 0001 beginnend mit jedem Kompilieren des Codes um eins erhöht. Version 5.0.0-3242 stünde also für das 3242. Kompilationsprodukt einer Software. Verwendet man Versionskontrollsysteme, so wird an Stelle der Build-Nummer gerne eine Nummer verwendet, die die Quellen zum Kompilat innerhalb des Versionskontrollsystems eindeutig identifiziert. Das erleichtert im Fehlerfall, die zugehörigen Quellen zu finden.[1]

```
2.3.5-0041
      |  |  |
      |  |  Buildnummer
      |  Revisionsnummer
      Nebenversionsnummer
      Hauptversionsnummer
```

Alpha
während der Entwicklung der Software, sehr frühes Stadium

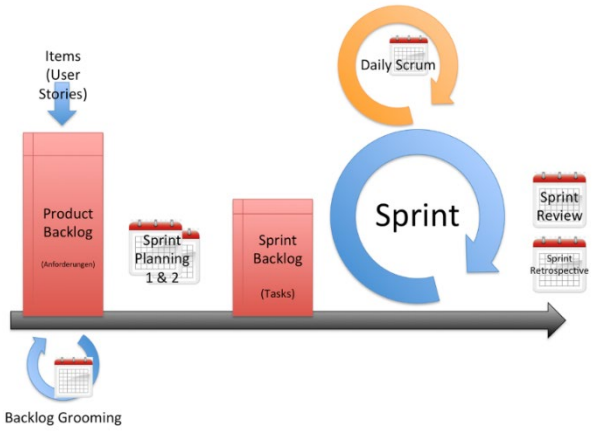
Beta
zum Testen vorgesehen, begrenzter Anwenderkreis

RC
Veröffentlichungskandidat (release candidate, *rc*), abschließende Testversion

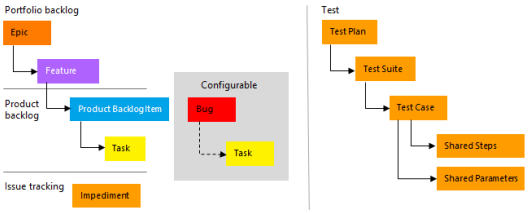
Release (final)
endgültige Version (englisch *Release* für Veröffentlichung)

Patch
auch patchlevel, *pl*, deutsch Korrekturlevel

Rollen (Roles)	Es gibt drei innere und drei äussere Rollen, generell sind alle Personen im Projekt an eine Rolle gebunden und erhalten dementsprechend ihren eigenen Aufgabenbereich.
Artefakte (Artifacts)	Darunter fallen alle Dokumente und Inhalte der Planung, Grafiken und Auflistungen des Projekts (administrativer Output) sowie das programmierte Produkt selbst!
Besprechungen (Ceremonies, Events)	Jede Aktivität hat vordefinierte Besprechungen, damit wird die Durchgängigkeit des Projekts gewährleistet, ohne dieses aber zu förmlich werden zu lassen. (Ich habe absichtlich den Begriff "Sitzung" nicht gewählt!)



Epics - oberstes und grösstes Element
Features
User Stories (Product Backlog Items = PBI) - Diese sind PFLICHT!
Tasks - kleinste Arbeitseinheit - Diese sind PFLICHT!



Priorisierung der Backlog Items

Alle PBI's werden nach deren Erstellung in eine eindeutige Reihenfolge gebracht. Wie wird diese festgelegt?

Wirtschaftlicher Nutzen
PBI's mit hohem wirtschaftlichen Nutzen für den Kunden werden höher priorisiert. Dadurch erhält der Kunde schneller eine Software, welche er für sein Kernbedürfnis einsetzen kann. Das Vertrauen wird bei Projektbeginn erhöht, da der Kunde für ihn wichtige Fortschritte sieht. (Ihn interessieren technische Details sicher weniger...)

Technisches Risiko
PBI's mit hohem technischen Risiko für die Entwicklung werden ebenfalls höher priorisiert. Damit wird verhindert, dass komplexe oder unlösbare Aufgaben zu spät im Projektverlauf entdeckt werden. Der Druck auf die Entwicklung ist also zu Beginn am höchsten, sollte dann abnehmen.

Abhängigkeiten zwischen PBI's
Manchmal ist es erst möglich ein PBI zu erstellen, wenn bereits ein anderes besteht. Aber Achtung! Viele Abhängigkeiten sind auflösbar. Dazu Beispiele:

- Erstellen Sie keinesfalls zuerst Loginprozesse und Benutzerverwaltungen, nur weil diese im Programmablauf für den Anwender "zuerst" kommen. Ein MVP (Minimal Viable Product) benötigt kein Login.
- Um einen Bestellprozess zu implementieren benötigen Sie keine funktionierende Kundenverwaltung mit GUI usw. Es reicht wahrscheinlich fürs Erste, eine Kundentabelle zu erstellen, ein paar Daten manuell einzufügen und mit diesen zu arbeiten. Die ganze Verwaltung (CRUD) der Kunden kann nachträglich erstellt werden. Sie hat weder einen hohen wirtschaftlichen Nutzen, noch ein grosses technisches Risiko im Vergleich zum Bestellprozess.

```
Bullet lists nested within
numbered list:

  1. fruits
     * apple
     * banana
  2. vegetables
     - carrot
     - broccoli
```

Text using Markdown syntax

```
Heading
=======

Sub-heading
-----------

# Alternative heading

## Alternative sub-heading

Paragraphs are separated
by a blank line.

Two spaces at the end of a
line
produce a line break.
```