Prozessmodelle

Der Software- (Engineering-) Prozess ist eine Aneinanderreihung von Aktivitäten und damit verbundenen Ergebnissen, die ein Softwareprodukt erzeugen.

Anforderungsspezifikation

- Die Software-Spezifikation definiert die zu produzierenden Software und deren Einschränkungen im Betrieb.
- Software-Entwicklung beinhaltet das Software-Design (Bauplan) und die Implementierung der eigentlichen Software.
- Softwarevalidierung stellt sichern, dass die Software der Anforderungen des Kunden erfüllt und korrekt funktioniert.
- Software-Weiterentwicklung oder Anpassung ist der Prozess der Anpassung der Software an die sich ändernde Umgebung, z.B. an neue Marktanforderungen des Kunden.

Software- (Engineering-) Prozessmodelle sind vereinfachte und abstrakte Beschreibungen eines Software-Prozesses, der einen Abschnitt vom Ganzen darstellt.

Prozessmodelle

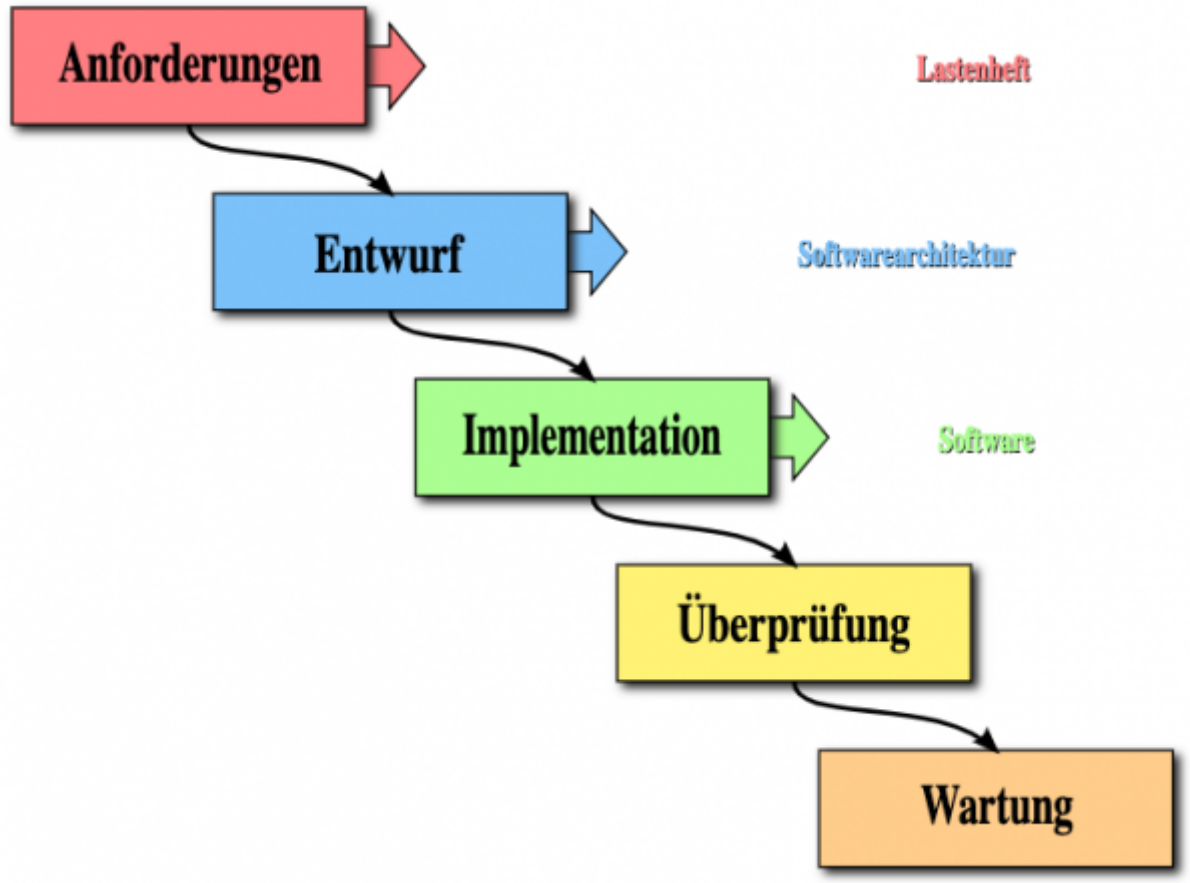
Prozessmodelle können Aktivitäten enthalten, die Teil der Software sind Softwareprodukte wie beispielsweise Architekturbeschreibungen, den ganzen Quellcode, Code-Fragmente oder Benutzerdokumentation so wie die Rollen der beteiligten Personen bei der Softwareentwicklung. Die gängigsten Prozessmodelle sind hierbei:

- Das Wasserfallmodell
- Scrum
- “V-Modell (XT)”
- eXtreme Programming

Dabei können große Projekte auch aus unterschiedlichen, so wie mehreren Modellen in verschiedenen Abschnitten des Entwicklungsprozesses bestehen.

Das Wasserfallmodell

Das Wasserfallmodell eines in der Vergangenheit am meisten Benutzte Modell kann als ein generisches Prozessmodell betrachtet werden.



Quelle: Wikipedia

1. Anforderungsanalyse und Definition: Die Anforderungen werden in Absprache mit dem vom Benutzer verwendeten System festgelegt. Danach werden diese detailliert definiert und dienen als Systemspezifikation für die Software.
2. System- und Software-Entwurf: Nachdem die Systemarchitektur definiert ist, beginnt die grundlegend Abstraktion und Identifikation der Software.
3. Implementierung und Unit-Tests: Das Software-Design wird hierbei als ein Abschnitt, der Programmierung dargestellt, eben so das Testen / Überprüft, ob die Teilabschnitte der Software ihren Spezifikationen entsprechen.
4. Integration und Systemtests: Die Software wird als Gesamtsystem integriert und getestet.
5. Betrieb und Instandhaltung: Das installierte System wird nun an den Kunden / Benutzer ausgeliefert. Dieser Abschnitt beinhaltet die Wartung so wie die Korrektur von Fehlern die erst bei der Nutzung entdeckt wurden. Ebenso wie die Weiterentwicklung der Software bei neuen Anforderungen.

Wichtiges Merkmal ist dabei, dass man bei jedem Prozessabschnitte wieder eine „Stufe“ oder mehrere „Stufen“ zurück gehen kann um dann diese Teilabschnitte oder diesen Teilabschnitt erneut durchlaufen kann. Eine Überspringen einer „Stufe“ oder mehrer „Stufen“ ist möglich.

Zusammenfassung

Das Ergebnis jeder Phase ist eine Reihe von genehmigten Artefakten. • Die folgende Phase beginnt, nachdem die vorherige Phase abgeschlossen ist. (In der Praxis kann es aber auch zu Überschneidungen kommen.) • Im Fehlerfall müssen vorherige Prozessschritte wiederholt werden. •

Passt zu anderen (Hardware-) Engineering-Prozessmodellen. (Auch die Hardwareentwicklung bewegen sich seit neustem in die Richtung agiler Methoden!)

Agile Entwicklung

Agile Softwareentwicklung – Prinzipien, Muster und Praktiken bauen auf iterativen Ansätzen auf.

Ziel ist es, Software schneller zu entwickeln, um schneller auf neue Anforderungen reagieren zu können. Ursprünglich ist die Agile Entwicklung nur für kleine und mittlere Teams konzipiert worden, um Agilität zu erreichen. Hierfür werden Praktiken angewendet, welche die notwendige Disziplin bieten und schnell Feedback geben. Es wurde ein Designprinzip entwickelt, dass die Software flexibel und wartbar macht. Dazu muss man die Designmuster genau kennen, um spezifische Probleme auszugleichen zu können.

Die Verwendung einer agilen Methode bedeutet nicht, dass die Stakeholder immer das bekommen, was sie wollen. Es bedeutet vielmehr, dass die Stakeholder das Team leichter steuern können, um den höchsten geschäftlichen Nutzen bei geringsten Kosten zu erzielen.

Der Stakeholder

Als Stakeholder wird der Project Manage oder eine von Ihm ernannte Person bezeichnet, die ein berechtigtes Interesse am Verlauf des Prozesses oder Projektes hat.

Das Manifest der agilen Softwareentwicklung

- Einzelpersonen und Interaktionen vor einzelnen Prozessen und Werkzeugen. Denn die besten Tools helfen nicht, wenn das Team nicht zusammenarbeitet. Beginnen Sie klein und wachsen Sie nach Bedarf.
- Arbeitssoftware und umfassende Dokumentation. Die Struktur des Systems und die Gründe für das Design sollten immer dokumentiert werden.
- Arbeiten sich immer eng mit dem Kunden zusammen. Dies ist sehr nützlich bei Vertragsverhandlungen. Darüberhinaus sollte der Vertrag angeben, wie die Zusammenarbeit zwischen Entwicklungsteam und Kunden aussieht. Informieren Sie Ihren Kunden frühzeitig über Veränderungen.
- Auf Veränderungen reagieren und den Plan darauf anpassen. Nicht strickt nach Plan untergehen. Nutzen Sie Veränderungen und machen diese zu Ihrem Vorteil.

Prinzipien

Unsere oberste Priorität ist es, den Kunden frühzeitig und zufrieden zu stellen. Durch kontinuierliche Software-Lieferungen unterstützen Sie diesen wertvollen Prozess. Liefern Sie dabei regelmäßig funktionsfähige Software. Je nach Projekt und Kunde sollte dies Wochenweise bis Monatsweise erfolgen. Kürz gehaltene Abstände zeigen dabei auch kleinere Details. Lange Abstände zeigen oft keinerlei Veränderung der Software.

Funktionierende Software ist das wichtigste Maß für den Fortschritt. Denn wenn 30% der

Funktionalität implementiert sind, sind auch 30% des Projekts abgeschlossen. Nichtfunktionale Software oder teilfunktionale Software ist nicht repräsentierbar und somit nicht fertig.

Ständige Aufmerksamkeit für technische Exzellenz und gutes Design erhöht die Beweglichkeit Ihres Projektes. Sie können schneller auf den Kunden reagieren. Und sind besser Vorbereitet, wenn der Kunde Änderungen in Erwägung zieht.

Einfachheit – Es ist keine Kunst, die Menge der nicht geleisteten Arbeit zu maximieren. Kleine Abschnitte, dafür kontinuierlich voranzuschreiten ist besser als schwierige Probleme in großen Team zu lösen.

Seien Sie offen für Änderungen und neue Projekt-Anforderungen, auch wenn Ihr Projekt in der Entwicklung weit voran geschritten ist. Der Agile Prozesse erlaubt es Veränderungen für Ihre Wettbewerbsfähigkeit gegenüber dem Kunden als Vorteil zu nutzen.

Ziehen Sie Ihr Team in regelmässig Abständen zu Verbesserungsvorschlägen heran, Ihr Team ist näher am Projekt wie Sie. Ihr Team weiss wie es mehr erreichen und wie es effektiver arbeiten kann, stimmen Sie dann Ihr Team entsprechend ab.

Die besten Architekturen, Anforderungen und Designs entstehen in selbstorganisierten Teams, wo jeder seiner Fähigkeit genau da einbringen kann, wo diese gebraucht werden. Falsch organisierte Teams, bei denen die persönlichen Präferenzen ausser Acht gelassen werden bremsen Ihr Projekt.

Unternehmer/Geschäftsleute und Entwickler während des gesamten Projekts zusammenarbeiten. Nicht jedem sind alle Aspekte der Software plausibel.

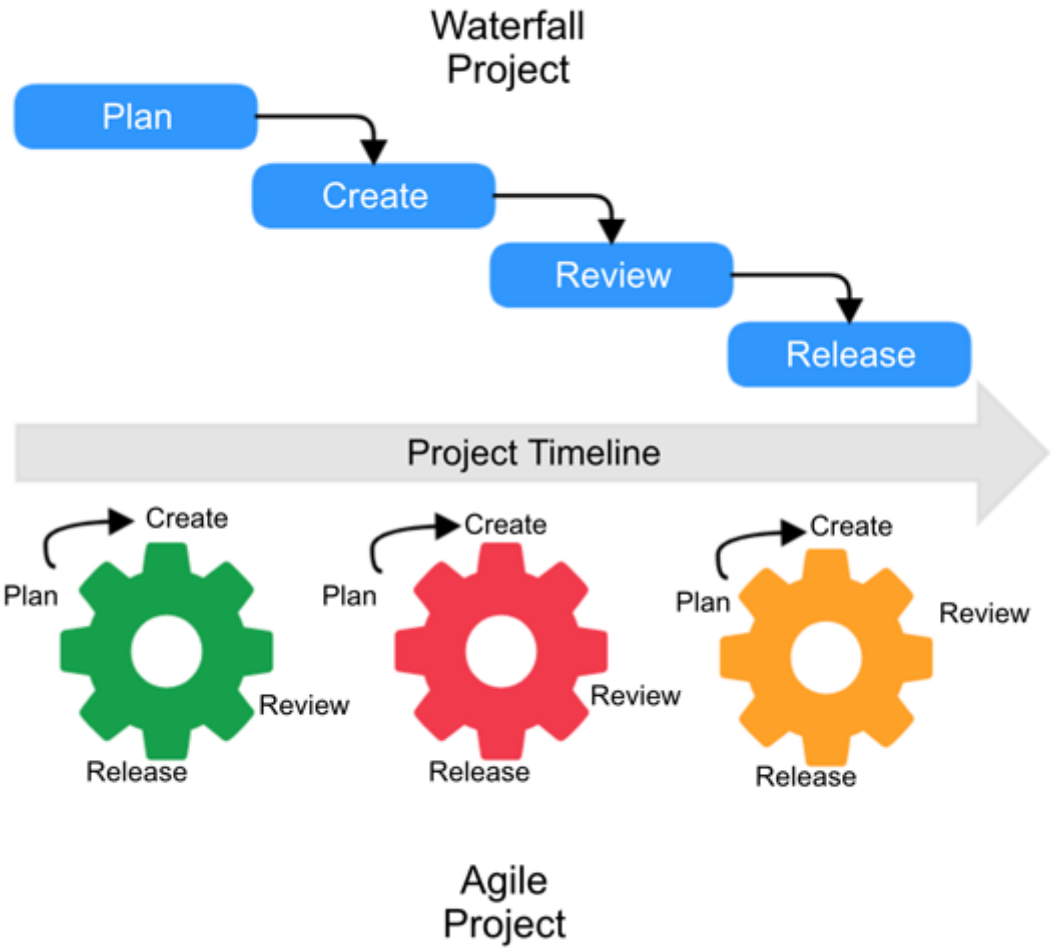
Projekte um motivierte Einzelpersonen herum aufbauen. Geben Sie Ihrem Team die Umgebung und Unterstützung, die sie brauchen. Dadurch bauen Sie Vertrauen auch Sicherheit auf. In einem Guten Arbeitsklima werden Jobs leichter und somit schneller erledigt.

Agile Prozesse fördern eine nachhaltige Entwicklung. Versuchen Sie bei der Entwicklung ein ein konstantes Tempo aufrecht zuhalten, dadurch entsteht weniger Stress und der Unterschied zwischen weniger stressigen zu stressigen Phasen ist nicht so Markant.

SCRUM

Einheitlicher Prozess – Phasen (im Projektmanagement)

1. **Inception** (~ dt. Konzeption): Machbarkeitsphase, in der gerade genug Nachforschungen angestellt werden, um mit dem Projekt fortzufahren oder es zu stoppen. Bekennen Sie sich zu einer Seite und unterstützen eine Entscheidung.
2. **Ausarbeitung** oder **Entwurf**: Die Kernarchitektur wird iterativ implementiert. hohe Risiken sind abzuschwächen um das Projekt fortzusetzen.
3. **Konstruktion**: Fortführung der iterativen Implementierung unter geringerem Risiko und Implementierung einfacher Elemente so wie Vorbereitung für die Bereitstellung für den Kunden.
4. **Transition** (~ dt. Übergabe): Betatests und Bereitstellung für den Kunden.



Quelle: redbooth.com

Allgemeine Durchführung

- Bewältigen Sie Probleme mit hohem Risiko und hoher Wertigkeit in frühen Iterationen
- Fordern Sie den Kunden kontinuierlich zur Bewertung, für Feedback und zu Anforderungen
- Erstellen Sie in frühen Iterationen eine zusammenhängende Kernarchitektur
- überprüfen Sie die Qualität kontinuierlich, testen Sie früh, oft und realistisch
- Wenden Sie typische Anwendungsfälle an
- Führen Sie einige visuelle Modellierungen durch
- verwalten Sie Anforderungen sorgfältig
- Üben Sie das Änderungsaufträge und Konfigurationsänderungen

Extreme Programming

Extreme Programming besteht aus einer Reihe von einfachen, voneinander abhängigen Verfahrensregeln.

User Stories

Anforderungen werden mit dem Kunden besprochen und in wenigen Worten niedergeschrieben. (Eine Karteikarte für eine Anforderung.

Kurze Zyklen

Liefern Sie alle zwei Wochen oder nach jeder Iteration eine funktionale Software an den Kunden. Die gelieferte Software kann, muss aber nicht direkt in Produktion gehen. Iterationen sind zeitgesteuert – Verspätungen sind illegal, wenn Sie nicht liefern können, schließen Sie alle anderen, für diese Iteration geplanten Aufgaben ab.

Iterationsplan

Bei jeder Wiederholung sind die User Stories und deren Prioritäten festgelegt. Der Kunde wählt die User Storys aus, die er umgesetzt haben möchte. Die Anzahl der User Stories ist durch das wurde von den Entwicklern festgelegte Budget begrenzt.

Release Plan

Eine Karten besteht aus ca. sechs Iterationen. Jede Karte kann jederzeit geändert werden.

The Planning Game

Aufgabenteilung zwischen Kunden und Entwickler. Der Kunde entscheidet, wie wichtig eine Funktion ist und die Entwickler entscheiden, wie viel diese Funktion zu implementieren kostet.

Initial Exploration (Projektstart)

- Entwickler und Kunde fassen nur die wichtigsten User Stories zusammen
- Es werden nicht alle User Stories aufgeschrieben
- Die Entwickler schätzen den relativen Aufwand jeder User Stories in Story-Punkte ab. Eine User Story die voraussichtlich doppelt so lange dauern wird hat demnach auch doppelt so viele Punkte.
- Um die exakte Dauer zu kennen, benötigen wir die Velocity vom Team, (Velocity = benötigte Zeit pro Story-Punkt).
- Die Velocity wird mit Verlauf des Projekts immer genauer. Anfangs ist es nur eine grobe Abschätzung, auf der Grundlage von „Erfahrungen“.

Oft wird deswegen auch ein Prototyp im Vorfeld entwickelt, um die Velocity besser abschätzen zu können. Dies wird dann als „Spike“ bezeichnet.

Iterationsplanung oder Iteration Planning

- der Kunde wählt eine User Story für eine Iteration aus
- die Bearbeitungsreihenfolge der User Stories innerhalb einer jeden Iteration ist eine technische Entscheidung des Projekt-Managers
- eine jede Iteration endet an einem festgelegten Datum (timeboxed oder Stichtag), auch wenn nicht alle User Stories fertig wurden.
- die Schätzungen aller fertigen User Stories werden dann summiert und die Velocity für die benötigten Iteration berechnet
- die geplante Velocity für jede Iteration ist die gemessene Velocity der vorherigen Iterationen

Aufgabenplanung oder Task Planning

- zu Beginn jeder Iteration setzen sich Entwickler und Kunde zum Planen zusammen
- die User Stories werden in kleinere Aufgaben unterteilt, welche dann innerhalb 4 bis 16 Stunden implementiert werden sollen
- jeder Entwickler übernimmt dafür eine beliebige Aufgabe – auch wenn er kein Experte darin ist

The Planning Game oder “Planning Poker”

- der Produktmanager gibt einen kurzen Überblick über das Projekt.
- dem Team wird dann die Gelegenheit gegeben, Fragen zu stellen und kurz über das Projekt zu diskutieren, um Annahmen und Risiken zu klären.
- jede Person erstellt verdeckt eine Karte mit seiner persönlichen Abschätzung für den Arbeitsaufwand. Die verwendete Einheiten können variieren, egal ob Tage, Stunden oder Story-Punkte.
- dann deckt jeder gleichzeitig seine Karten auf, und nennt seine Abschätzung.
- die Personen mit hohen und niedrigen Schätzungen müssen ihre Schätzung begründen.

Weitere Informationen zu diesem Thema finden Sie auf Wikipedia.

Schätzen Sie den Aufwand für die Implementierung anhand der Funktionalität ab. Erstellen Sie dazu eine Annotations-Datei in der jedes Objekt eine kommentierte Zeile bekommt. In dieser wird das Objekt genau spezifiziert. Dabei kann die Zeile entweder leer sein oder mit einem „#“ beginnen. Wenn es sich um einen Kommentar handelt, muss das zu verwendende Muster wie folgt aussehen:

'['<TYPE>']'<KEY>'='<VALUE>

Wenn beim Analysieren einer Zeile ein Fehler auftritt, wird die Zeile ignoriert und die Analyse fährt mit der nächsten Zeile fort. Nach dem Analysieren der vollständigen Datei wird eine Karte mit den validierten Rückgabewerten und Eigenschaften erstellt. Alle Zeilen, die nicht analysiert werden konnten oder die Validierung fehlschlug, werden ebenfalls zurückgegeben.

Extreme Programming – Umsetzung

Akzeptanztests

Details der User Storys werden in Form von Akzeptanztests erfasst. Diese Akzeptanztests sind üblicherweise Black-Box-Tests und werden vor oder gleichzeitig mit der Implementierung einer jeden User Story geschrieben. Wenn ein Akzeptanztests bestanden ist, wird dieser dem „Set of Passing“

hinzugefügt. Bestandene Akzeptanztests dürfen nie wieder fehlschlagen.

Pair Programming

Der Code wird von einem „Programmierer-Paar“ geschrieben, dabei implementiert einer von beiden einen Code-Abschnitt und das andere Mitglied überprüft dann den implementierten Code-Abschnitt. Bestenfalls ändern sich die Paarungen nach einem oder einem halben Tag, um sicherzustellen, dass Wissen verbreitet wird und dass es zu keiner „Betriebsblindheit“ oder Gewohnheit kommt.

Eine Erweiterung dieser Art ist “Mob-Programmierung„

Umgestaltung oder Refactoring

Führen Sie häufig Refactorings beim Hinzufügen von neuen Funktionen durch, um dadurch „rots“ im Code zu vermeiden.

„Rots“ bedeutet, dass unter ständiger Aktualisierung, Erweiterung und Änderungen Code „unschön“ wird. Da der vorhandene Code nicht 100% passt, wird dieser im Laufe der Zeit immer wieder abgeändert und dadurch kommen mehr Fehler in den Code. Im Schlimmsten Fall kann es auch passieren, dass der alte Code nicht mehr „korrekt“, oder zu sehr hässlichem „hacking“ Code wird. Dies reduziert die Integrität des Codes und „verrottet“ diesen, bis er schließlich unbrauchbar wird.

Umgestaltung oder Refactoring bedeutet Verbesserung der (Code-) Struktur ohne das eigentliche Verhalten zu ändern.

Test-Driven Development

Hiebei stehen die Testes (Unit-Tests) im Vordergrund. Diese werden zuerst von den Entwicklern implementiert und dienen dann als Grundstruktur des Projektes. Dabei wird der gesamte Code um die (Unit-)Tests zu bestehen geschrieben. Wurden zu einem Abschnitt alle Testfälle vollständig bestanden, wird das Refactoring durchgeführt dies führt dazu, dass es zu weniger gekoppelten Code kommt.

Refactoring implizit weniger gekoppelten Code.

Continuous Integration

Programmierer überprüfen ihren Code und integrieren diesen mehrmals am Tag. Dazu wird „non-blocking Source Control“ oder „Nicht blockierende Quellcodeverwaltung“ verwendet. Das bedeutet nach dem „check-in“, „Einchecken“ oder „Push to Master“ durchläuft der Code mehrere Test, besteht der Codeabschnitt alle diese Tests wird er in das System gebaut oder ausgeliefert.

Einfaches Design

Halten Sie das Design so einfach und ausdrucksstark wie nur möglich. Konzentrieren Sie sich auf die aktuellen User Stories und machen Sie sich dabei keine Sorgen über anstehende User Stories und deren Kompatibilität. Z.B. Fügen Sie die Änderung nur dann durch, wenn eine User Story sie dazu zwingt und es keine andere Möglichkeit gibt. Jede User Story wird als ein einzelner Block betrachtet.

Suchen Sie nach der einfachsten Möglichkeit die Funktion umzusetzen! Dazu suchen Sie die einfachste Entwurfsoption für die aktuelle User Story.

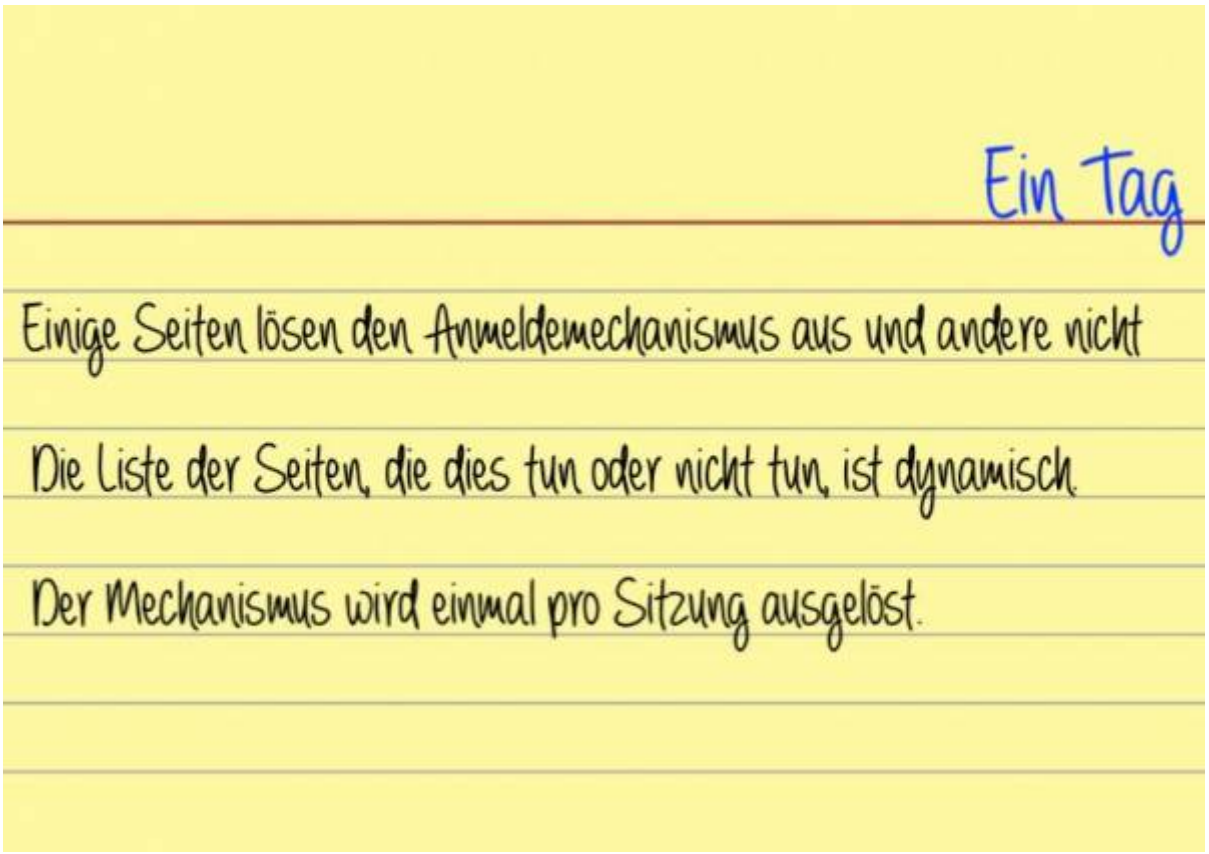
You aren’t going to need it! Rechnen Sie immer mit der Möglichkeit, dass Sie mit Voranschreiten des Projektes die aktuelle User Story nicht mehr benötigen werden. Verlieren Sie darum nicht unnötige Zeit bei der Umsetzung. Fügen Sie eine Code-Erweiterung nur hinzu, wenn ein Sicherheit oder mindestens ein überzeugender Beweise für die Notwendigkeit vorliegt.

Einmal ist ausreichend! Dulden Sie keine Code-Duplizierung. Beseitigen Sie redundanten Code durch Abstraktionen. Entfernen Sie Muster vor der Entfernung des redundanten Codes.

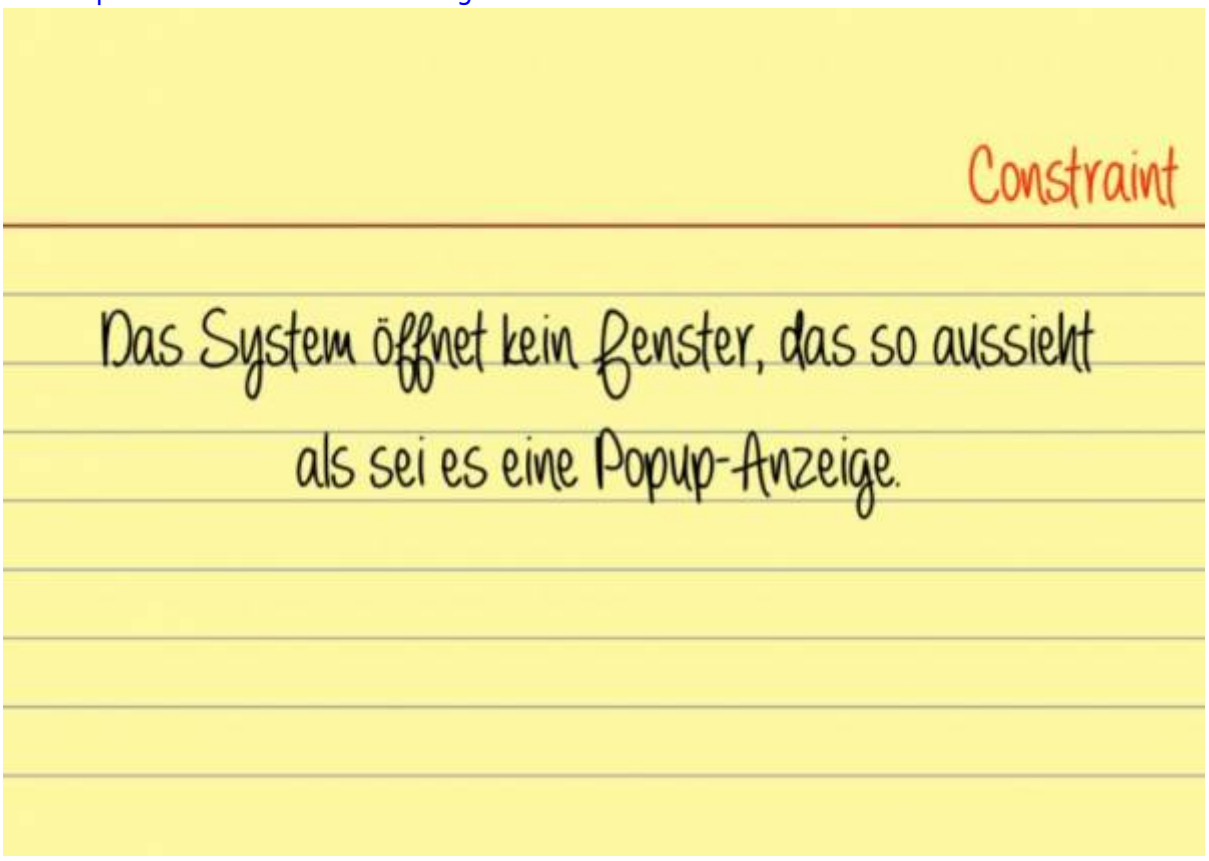
User Stories

Beispiel

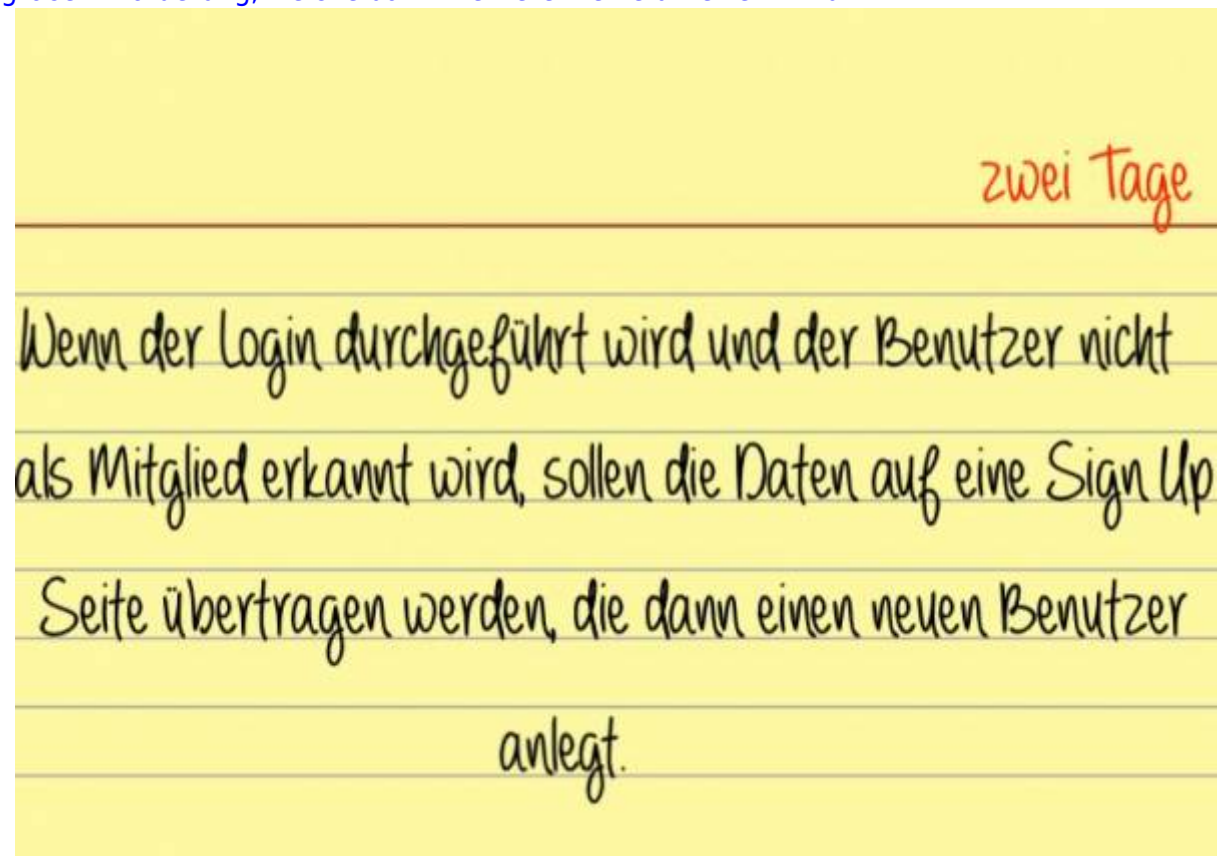
User Stories für eine Web Application



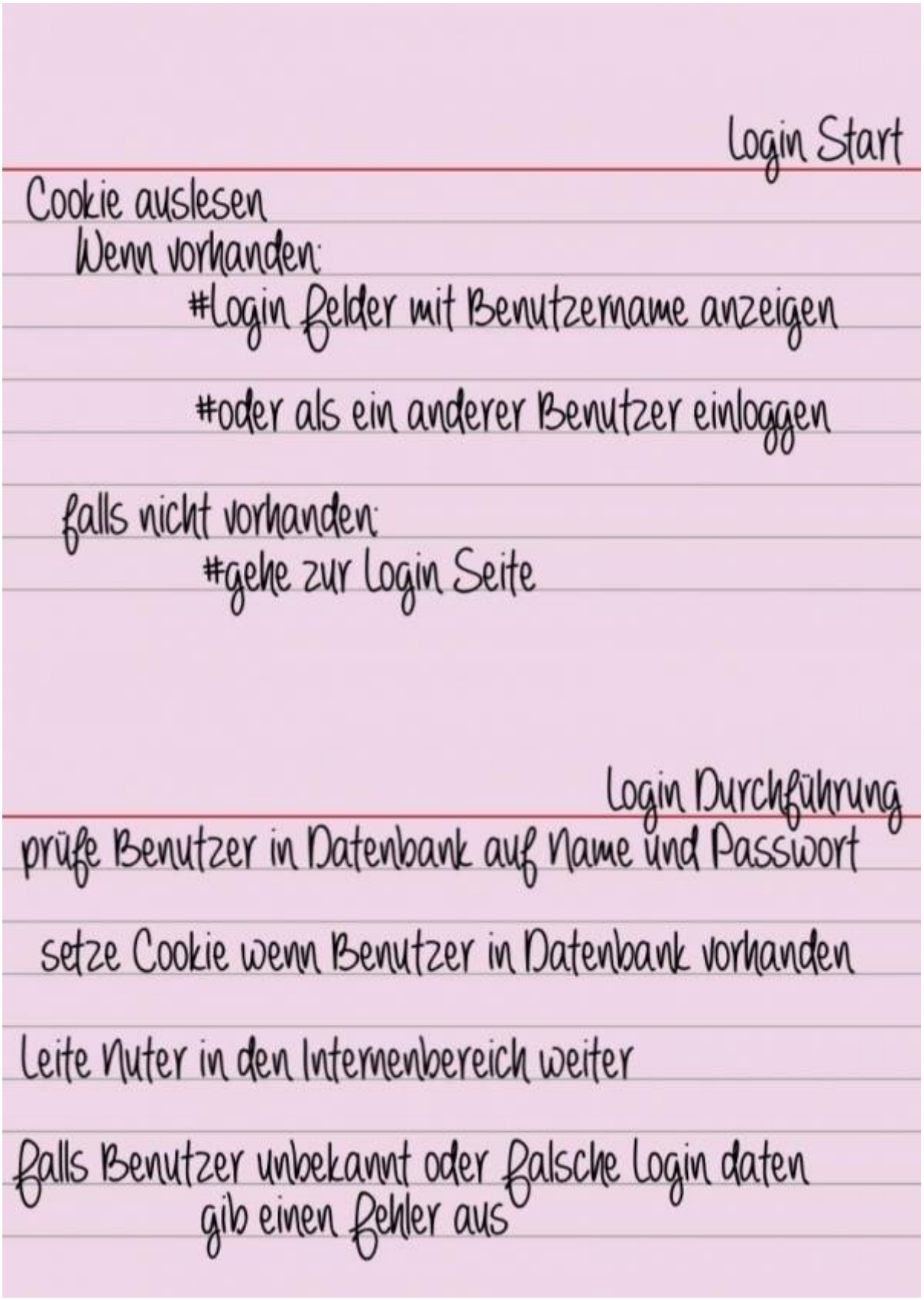
Eine nicht implementierbare Anforderung:



Eine große Anforderung, welche dann mehrere Kleine unterteilt wird:



Aufgeteilte Anforderung:



Leitfaden für gute Anforderungen

- Anforderungen müssen für den Kunden verständlich sein, die Profies sind in Ihrem Team und haben weniger Erklärungsbedarf
- jede Anforderungen muss dem Kunden etwas Wertvolles bieten
- Anforderungen müssen klein genug sein um mehrere davon in einem Iterationsschritt

durchführen zu können
- Anforderungen müssen unabhängig voneinander sein
- jede Anforderungen muss testbar sein

INVEST:

- **I**ndependent, Unabhängig,
- **N**egotiable, verhandelbar,
- **V**aluable, wertvoll,
- **E**stimable, abschätzbar,
- **S**ized appropriately, angemessen,
- **T**estable, überprüfbar.

Etablierte Vorlagen für User Storys

- Lange Vorlage: „Als <Anforderung> möchte ich <Ziel> , damit <Wert / benefit> erreichen“
- Kürzere Vorlage:“Als <Anforderung> möchte ich <Ziel / benefit> erreichen“

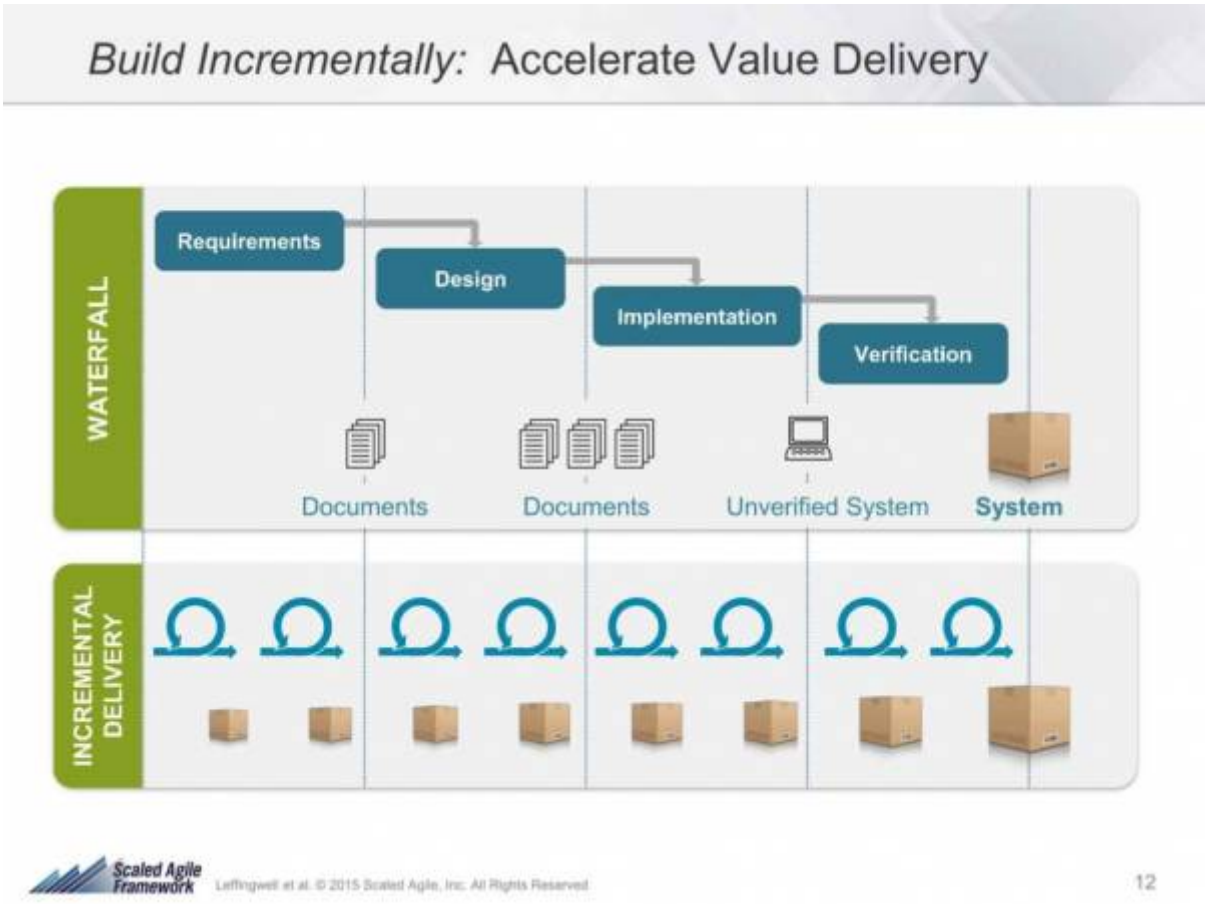
Beschreiben von User Storys

ID	**3**
Name	Login
Beschreibung	Als Administrator muss ich mich am System mit Benutzername und Passwort authentifizieren, um Änderungen vornehmen zu können
Akzeptanzkriterium	Der Dialog zum Einloggen wird korrekt angezeigt und es muss möglich sein, sich als Administrator zu authentifizieren. Ungültige Eingaben sollen ignoriert werden und normale Nutzer erhalten die Rolle als „Benutzer“ und nicht die Rolle des “Administrator”
Geschätzter Aufwand	3 Story Points
Entwickler	Raffael
Umgesetzt in Iteration	2
Tatsächlicher Aufwand (Std.)	6 Stunden
Velocity (Std./Story Point)	2
Bemerkungen	–

Scrum (kurze Übersicht)

- Scrum ist ein Projektmanagement-Framework
- Scrum verwendet einen iterativen, inkrementellen Ansatz zur Optimierung der Berechenbarkeit und Risikoabschätzung

Scrum vs Wasserfall



Quelle: infoq.com

Das Problem bei der Wasserfall Methode ist, dass Sie fast immer erst am Schluss des Projektes ein vorzeigbares Produkt haben. Zum Schluss kommt dann immer die „heisse Phase“, in der ein Produkt dann fertig wird oder schlimmstenfalls auch scheitern kann. Bei SCRUM oder einer vergleichbaren agilen Methode, haben Sie etappenweise kleine vorzeigbare Produkte. Sie laufen weniger Gefahr, dass Ihr Projekt zum Ende hin noch scheitern kann. Ihr Produkt wächst stetig und kann auch durch seine modulare Bauweise leichter angepasst werden.

Scrum Rollenverteilung

Product Owner

- verwaltet das Product Backlog

Development Team / Entwickler

- liefert funktionierende Software (Pakete), die den Anforderungen des Kunden entspricht
- das Team organisiert sich eigenständig

Scrum Master

- kümmert sich um den Prozess und
- stellt sicher, dass der Prozessablauf eingehalten wird

Scrum Events

Sprint

- Iteration bis zu einem Termin „Dead Line“
- Entwicklung erfolgt innerhalb eines Sprints
- beginnt mit einer Planungsphase und endet mit dem Sprint Review

Sprint-Planung oder Planungsphase

- Identifizierung der Aufgaben / Funktionen aus dem Produkt-Backlog
- während des Sprints erfolgt die Abstimmung zwischen dem Product Owner und den Entwickler
- Sie bestimmen das Design

Daily Scrum (oder Jour fix)

- ein kurzes, zeitlich begrenztes Meeting, um die Just-in-Time-Planung durchzusetzen

Sprint Review oder Sprint-Überprüfung

- Entwicklungsteam und Product Owner überprüfen die Ergebnisse
- Aktualisierung des Produkt-Backlog auf Grundlage der erzielten Ergebnisse

Rückblick oder Retroperspective

- prüfen, wie der letzte Sprint verlaufen ist und was entsprechend verbessert werden kann

Scrum Begriffe

Product Backlog

- eine priorisierte Liste der zu erledigenen Aufgaben, welche vom Product Owner geführt wird.
- der Product Backlog ist transparent
- die Priorität wird unter Berücksichtigung der Monetarisierung und dem verbunden Risiko für den Kunden festgelegt.
- Dinge mit hoher Priorität sollten so zerlegt werden, dass diese leicht und schnell umsetzbar sind

Sprint Backlog

- beinhaltet die umsetzbaren Elemente aus dem Produkt-Backlog

Anpassungsfähigkeit

Different types of systems need different development processes! Bleiben Sie anpassungsfähig!

Software, die in einem Flugzeug verwendet wird, muss anders entwickelt werden als eine Software für eine E-Commerce-Webseite. Ein Betriebssystem wird anders anders entwickelt als ein Prozessor. In großen Projekten können verschiedene Teile unter Verwendung verschiedener Prozessmodelle entwickelt werden.

Es gibt nicht den einen richtigen Weg, für alle Entwicklungen

Ziel

Ziel ist es systematisch kleinere (Software-) Pakete, mit maximaler Qualität einfach zu produzieren.

Um systematisch Software zu entwickeln, müssen Sie einem genau definierten Prozess folgen. Dieser entspricht den Bedürfnissen des in der Entwicklung befindlichen Projekts. Vergessen Sie niemals, es ist praktisch unmöglich, alle Anforderungen gleich zu Beginn einer Projektes zu kennen. Ein Projekt entwickelt sich durch neue Erkenntnisse in der Entwicklung und neue Anforderungen des Kunden.