

Software Engineering

Teil 7: Einführung in Agile Methoden -
Scrum als Beispiel für Agile Methoden

April 2018

Dr. Christian Bartelt



UNIVERSITATEA
BABEŞ-BOLYAI

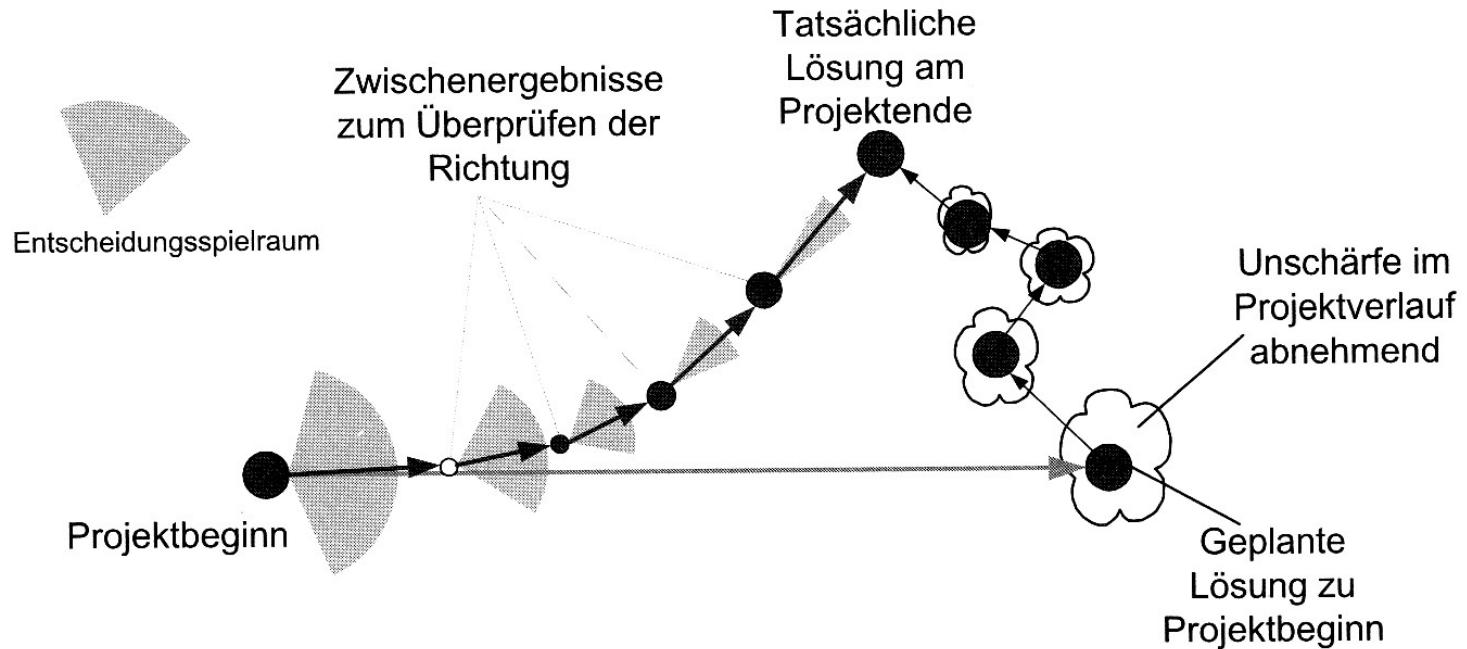
Universität Mannheim
Institut für Enterprise Systems InES
Schloss
68131 Mannheim / Germany

UNIVERSITY OF
MANNHEIM

Überblick

- Warum agile Methoden
- Agile Ansätze
- Scrum

Problem der Unsicherheit und Risiken



Grafik von Iteratec GmbH, aus Starke:
Effektive Softwarearchitekturen,
Hanser; 2002

Woher kommt die Unsicherheit?

- Kunde **weiss nicht genau**, was er braucht (kann er z.T. nicht)
- Nicht alle Anforderungen und Stakeholder bekannt
- Kunde hat widersprüchliche Anforderungen, z.T. wg. Politik
- Management Risiken
- Auftragnehmer **versteht nicht genau**, was der Kunde will
- Auftragnehmer unterschätzt / überschätzt Aufwände (Planung)
- Änderungen in den Prioritäten, Geschäftsprozessen etc. während des Projektes
- Projekt in komplexe Projektlandschaft eingebunden
- Technische Risiken, z.B. Infrastruktur hält nicht was sie verspricht

Antworten „traditioneller“ Entwicklungsprozesse

- Anforderungen
 - Am Anfang des Projekts vollständig abstimmen (Bei Risiken in Stufen vorgehen)
 - Genaue Spezifikation/Pflichtenheft erstellen (z.T. Vertragsgrundlage)
 - Änderungen nur über ein definiertes **Change Request Verfahren**
 - Ziele: **genaues Verständnis** der Anforderungen, **Anforderungsstabilität, (Verfolgbarkeit)**
- Architektur
 - In früher Projektphase entwickeln (mindestens Grobarchitektur)
 - Anforderungen späterer Phasen berücksichtigen
 - Ziele: Architektur = **stabiles Gerüst** der Software
- Dokumente
 - Alles wird dokumentiert (Prinzip der Schriftlichkeit), wichtig sind insbesondere **Spezifikationsdokumente** und **Architekturdokumente**
 - Für Offshore-Partner, Wartungsteam (wobei diese kaum lesen), das Teams
 - Ziele: **Klarheit, Review-Fähigkeit** bzw. **Abstimmbarkeit, Stabilität** und **Reduktion von Kommunikation**

Antworten „traditioneller“ Entwicklungsprozesse

- Risikomanagement
 - Projekte werden in der Regel in (wenigen) Stufen (3-12 Monate) durchgeführt
 - Aktiver (häufig unsystematischer) Umgang mit Risiken
(Einzelmaßnahmen wie technischer Durchstich, Risikoliste, ...)
 - Risiken häufig nur pauschal berücksichtigt (z.B. 20% „Risiko-Zuschlag“)
- Pläne
 - Am Anfang des Projektes grobe Planung (auch zur Preisfindung), Feinplanung bei Bedarf
 - Wenn's knapp wird: Weglassen/Verschieben unwichtiger Funktionen oder Qualitätseinbußen (z.B. Testaufwände kürzen)
 - Ziele nach Priorität : (1) Einhaltung des Termins, (2) Einhaltung des Budgets, (3) Qualität, (4) Vollständige Funktionalität
- Verträge
 - Meistens **Festpreisprojekte**
 - Großkunden haben **Einkauf** der entscheidet nach **Preis**
(Annahme: alle Anbieter sind gleichwertig, Gegenargumentation schwer)

Antworten „traditioneller“ Entwicklungsprozesse

- Kunde
 - integriert nach Verfügbarkeit (am Anfang, permanent, auch „on-site“)
 - Über **Anforderungsdefinition**, **Dokument-Reviews** und System- und **Abnahmetest** einbezogen
 - Entwicklung: teilweise gemischte Teams (z.B. bei sd&m, BMW)
 - Kommunikation zum Teil reglementiert durch Projektleiter („Ein-Ansprechpartner“-Modell) oder über Dokumente
 - Ziele: Langfristige Kundenbindung, Vertrauen, verbindliche Absprachen
- Prozesse
 - Aktivitäten / Workflows im Detail festgelegt
 - Viele Rollen (RUP / V-Modell ca. 30)
 - Viele, genau festgelegte Artefakte (Doku, Code und Co.)
 - Ziele: **Planbarkeit**, **Wiederholbarkeit**, **Verbesserbarkeit** (CMM)

Vorwürfe an „traditionelle Methoden“

- ***Meinung/Anforderungen ändern verboten / sehr teuer***
- ***„Software-Bürokratie“:***
Zu viele Dokumente werden erzeugt, insbesondere Berichtswesen zu umfangreiche Spezifikation / Architektur
- ***Zu spätes Feedback vom Kunden*** („ist doch alles Wasserfall“)
- Kunde bekommt eigentlich nicht, was er will, sondern das was ***im Vertrag*** steht
- Fördert ***überkandideltes Design*** („Chef-Architekten-Denkmaler“)
- Macht ***keinen Spaß***, nicht Menschen gerecht, vernachlässigt Kompetenzen der Entwickler
- Zu ***teuer***, zu schwergewichtig, zu ***träge***
- Traditionelle Prozesse werden ***nicht wirklich gelebt***
- „traditionelle“ Methode = RUP, V-Modell (XT), diverse Unternehmensinterne Prozesse (BMW-ITPM, sd&m-Prozess,...)

Überblick

- Warum agile Methoden
- Agile Ansätze
- Scrum

Prinzip von agilen Vorgehensmodellen

Agile Alliance 2001

„Wir entdecken bessere Wege zur Entwicklung von Software, in dem wir Software entwickeln und anderen bei der Entwicklung helfen. Dadurch haben wir gelernt:

- **Menschen und Kommunikation:**
 - Wichtiger als Prozesse und Werkzeuge
 - Wichtiger als umfangreiche Dokumentation
 - Wichtiger als Vertragsverhandlungen
 - Wichtiger als Pläne stur folgen
- **Lauffähige Software:**
- **Zusammenarbeit mit Kunde:**
- **Reagieren auf Veränderung:**

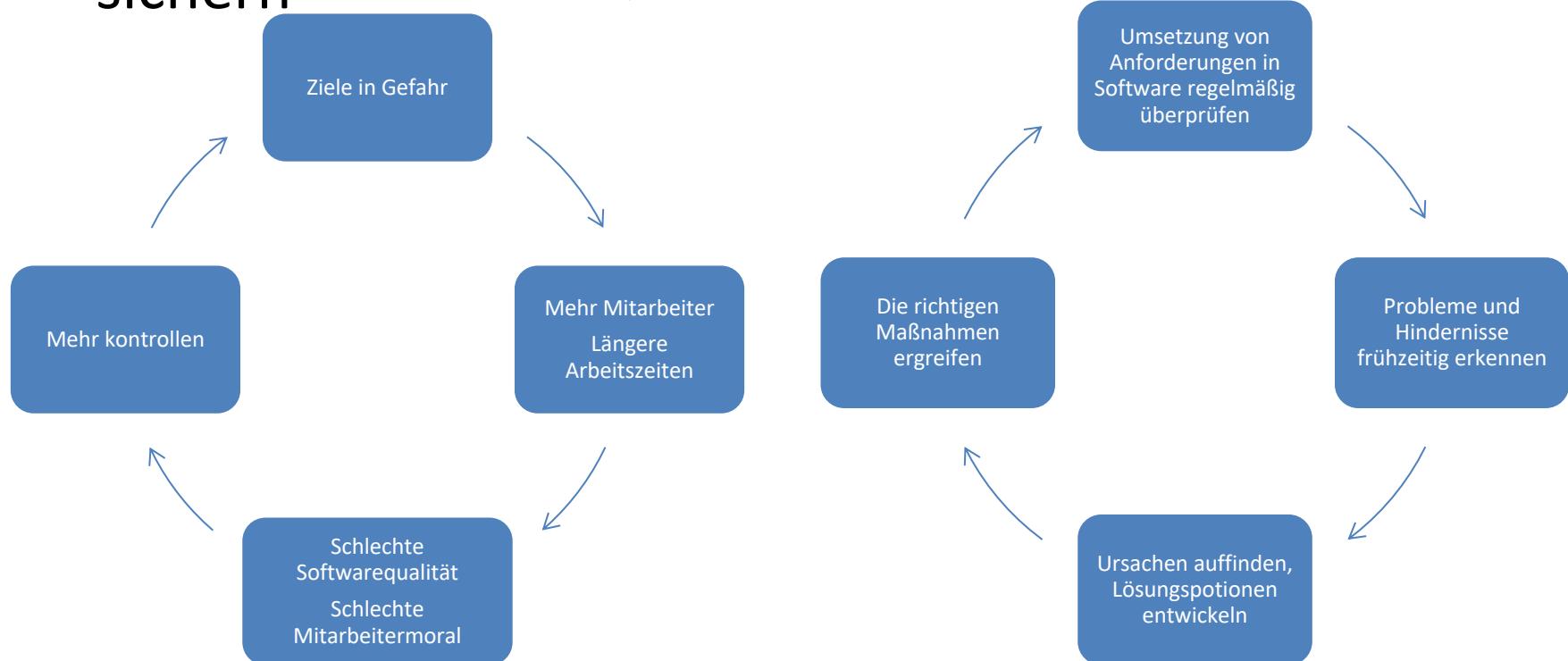
Natürlich sind auch die Dinge rechts wichtig, aber im Zweifel schätzen wir die linken höher ein.“

Quelle: Agile Manifesto: www.agilemanifesto.org

Agiler Kreislauf und Teufelskreislauf

- Probleme frühzeitig erkennen, Handlungsspielraum sichern

Reference: „Scrum-Agiles Projektmanagement erfolgreich einsetzen“, Roman Pichler, 2008

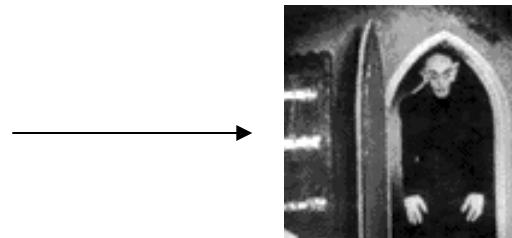
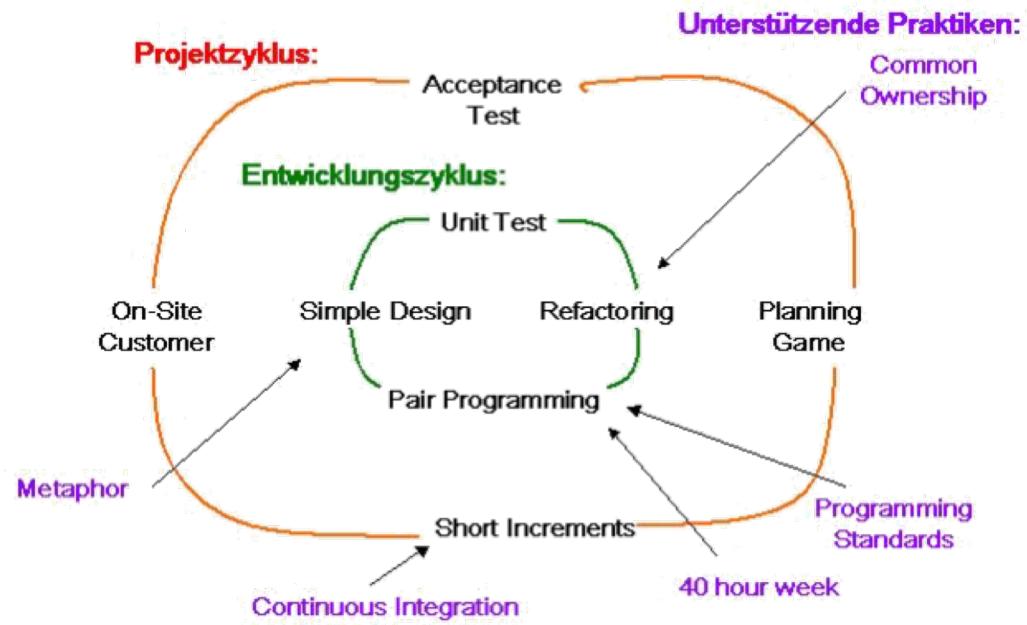


Teufelskreislauf

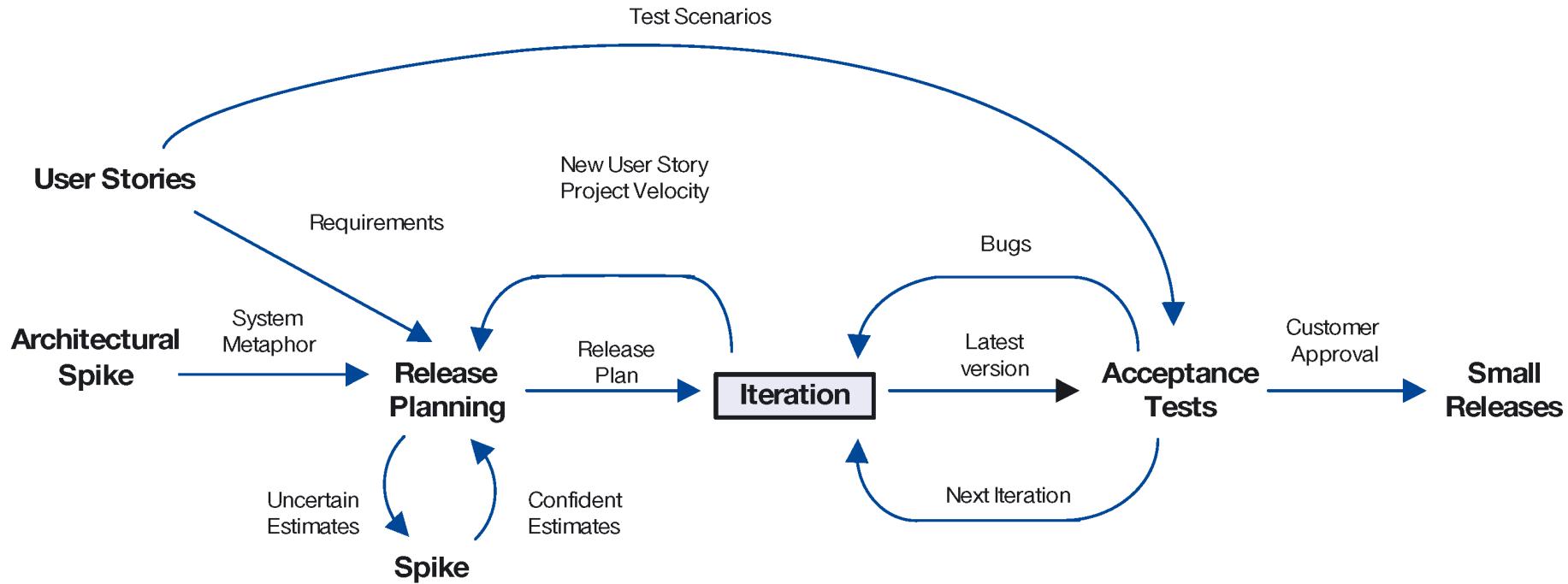
Agiler Kreislauf

Beispiel: eXtreme Programming – die Praktiken

- **Projektzyklus:**
 - Planning Game
 - On-Site Customer
 - Acceptance Test
 - Short Increments
- **Entwicklungszyklus:**
 - Simple Design
 - Pair Programming
 - Unit Test
 - Refactoring
- **Unterstützende Praktiken**
 - Metaphor
 - Collective Ownership
 - Continuous Integration
 - 40-hour week
 - Programming Standards



Beispiel: eXtreme Programming - Projektdurchführung

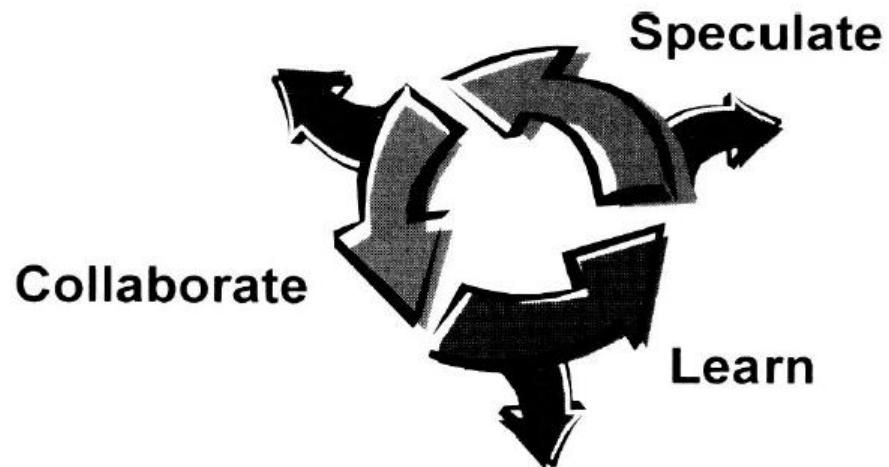


Beispiel: Adaptive Software Development (ASD)

Kurze inkrementelle Zyklen anstatt festhalten an einem starren Prozess

- Speculate
 - Anforderungen nicht bekannt und verändern sich
 - Spekulieren statt Planen
- Collaborate
 - Ein gutes Team erzeugt mehr als ein einzelnes Teammitglied
- Learn
 - Ergebnis der Lernphase dient als Input für die nächste Iteration:
 - Qualität der Software aus Kunden- und Entwicklungssicht
 - Verbesserung Entwicklungsprozess (Arbeitsmethoden & Praktiken)
 - Projektfortschritt

Adaptive Development Life Cycle



Aus: James A. Highsmith III: *Adaptive Software Development - A Collaborative Approach to Managing Complex Systems* [Hig00], Seite 41

Die ausscherenden Pfeile sollen betonen, dass jederzeit auch unkonventionelle Vorgehensweisen willkommen sind.

Agile Methoden

- Crystal Clear, -*Yellow*, -*Orange*, -*Red* (Alistair Cockburn)
- ASD (Jim Highsmith III)
- SCRUM (Ken Schaber, Mike Beedle, Jeff Sutherland)
- XP (Kent Beck, Ward Cunningham, Ron Jeffries)
- Diverse Spezialmethoden („eine pro Methodenberater“)
- RUP (Rational Unified Process)

Individuen und Interaktionen

Verantwortlichkeit des Einzelnen

- Jedes Teammitglied ist für das Ergebnis **verantwortlich**, nimmt Aufgaben/Risiken aktiv in Angriff
- Team organisiert sich während der Entwicklung weitgehend selbst
- Technik z.B. kurzes Meeting am Anfang jedes Tages (Scrum-Meeting,...)
- Kommunikation statt Dokumentation

Praktiken, Kultur und Adaptivität statt feste Prozesse

- Agile Prozesse schlank definiert, nur Rahmen
(wenige Rollen, wenige Artefakte, kaum Workflows)
- XP: Zusammenspielende Praktiken statt umfangreicher Workflows
(Refactoring, Collective Code Ownership, ...)
- Team gibt sich selbst einen Prozess und passt diesen während der Entwicklung an (z.B. „Empirie“, Methodology Tuning-Workshop)

Werte

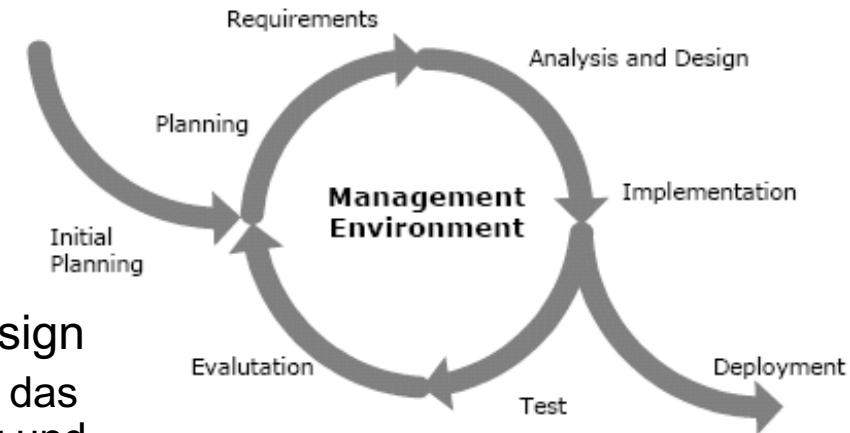
- Teamgeist, Team Ergebnisse (Collective Code Ownership, ...), Spaß (Cooperative Game), Mensch im Mittelpunkt (z.B. 40-Hour-Week)

Individuen und Interaktionen - Voraussetzungen

- **Verantwortlichkeit nur mit hoch qualifizierten Teams**
(nur qualifizierte Leute „wissen“, was zu tun ist)
- i.d.R. **kleine Teams** bis zu 10 Personen
(Große Teams führen zur Kommunikationsexplosion)
- Team braucht gemeinsamen Raum
- Eigenbrödler, Star-Architekten und Gurus nicht gefragt
- Management-Akzeptanz
 - Wiederholbarkeit der Ergebnisse nicht über Prozess gewährleistet
 - Planbarkeit, Verträge bei änderbarem Prozess?
 - Gutes Arbeiten von Chaos unterscheidbar?

Funktionierende Software (1)

- Kurze Releasezyklen (Iterativ, inkrementell)
 - Software wird schnell an den Kunden ausgeliefert, funktionierende Software wird damit erzwungen [Erziehungs-, Kulturfrage!]
 - Typische Releaselängen: 1-4 Monate, häufig Time-Boxing
 - Release wird jeweils durch den Kunden abgenommen
 - Schnelles Feedback, schnelle QS
- Evolutionäre Entwicklung, Einfaches Design
 - Start mit funktionierendem **Mini-Design**, das wächst und über **Refactoring** angepasst und verbessert wird, statt aufwändigem Architekturentwurf
 - Nur aktuelles Problem wird gelöst, nur aktuelle Anforderungen werden umgesetzt (KISS und YAGNI Prinzipien)



Funktionierende Software (2)

- Tests zentraler Bestandteil der Entwicklung
 - ***Test Driven Development***: Schreibe Testfall ***vor*** dem Code
 - Automatisierte Testsuiten (beim ***Nightly Build***, bei jedem Refactoring)
 - XP: Anforderungen -> Akzeptanztests
- Dokumentation soweit nötig
 - nicht keine Doku; Kunde bekommt das, wofür er bezahlt!
 - Crystal: Umfang abhängig von der Projektkritikalität und Mitarbeiterzahl (z.B. umfangreiche Spezifikation erst ab 10 Mitarbeiter wie in Crystal Orange zur Reduktion des Kommunikationsoverheads)
 - Statt Doku: Kommunikation im Team und mit dem Kunden
 - Alternative Doku: „Literate Programming“ mit JavaDoc?
- Teamarbeit führt zu Qualität
 - XP: Pair Programming, Collective Code Ownership, Aggressive Refactoring

Funktionierende Software - Voraussetzungen

- Kurze Releases nur wenn:
 - Akzeptanz / Zeit beim Fachdienst: Fachabteilung muss jedes Release (jeden Monat?) einzeln Abnehmen
 - Fachdienst muss Anforderungen priorisieren!
 - Anforderungen müssen in nützliche Releases partitionierbar sein
- Evolutionäre Architektur, nur wenn Änderungen billig
 - „Gute“ Start Architektur (Erfahrener Architekt)
 - Entwicklungsumgebung macht Änderungen leicht, z.B. mit Eclipse
- Die richtige Dokumentation weglassen
 - Entwicklungsteam == Wartungsteam?
(Problem des Know-How Transfers mit wenig Doku)
 - Anforderungen in geeigneter Tiefe / Vollständigkeit
(Erst Problem soweit wie nötig verstehen, dokumentieren dann Coden)
- Team-Ergebnisse möglich
 - Akzeptanz von Pair Programming, Collective Code Ownership

Kooperation mit den Kunden

- Kunde arbeitet im Team mit
 - On Site Customer
 - Ein befragbarer Kunde ersetzt Teile der Spezifikation
 - Schnelle Entscheidungen da **entscheidungsfreudiger** Kunde vor Ort
 - Schnelles Feedback, schnelle Klärung bei Problemen
- Kurze Releasezyklen: frühes Feedback vom Kunden
 - **Kunde bestimmt** Umfang des Releases nach Nutzwert (Prioritäten)
 - Entwickler schätzen Kosten, nennen Risiken
 - Gemeinsame Planung (z.B. Planning Game)
 - Explizit entweder Preis variabel (Aufwand) oder Umfang variabel
- Vertrauen
 - Vertrauen des Kunden ist wichtiger Bestandteil
 - Detailplanung, Vertrag wird durch **Versprechen** ersetzt,
immer das für den Kunden nützlichste zu tun (siehe oben)

Kooperation mit den Kunden - Voraussetzungen

- Kunde **arbeitet** im Team mit, Feedback:
 - Kunde vor Ort darf **entscheiden**
 - Kunde vor Ort ist **der Richtige** (andere Stakeholder werden weniger gefragt)
 - Kunde hat die **notwendige Zeit** (ein Fachdienst hat die häufig nicht)
- Vertrauen
 - Vertrauen beim Kunden, bei Management und Ansprechpartnern
 - Einkauf wird überrumpelt (Auswahl nach Festpreis geht nicht!)

Reaktion auf Änderungen

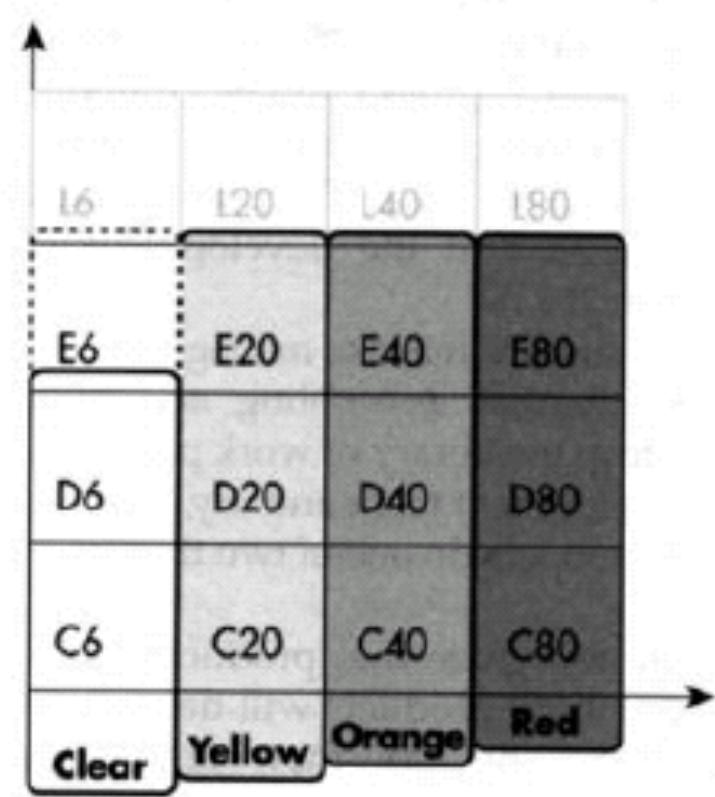
- Änderungen = Normalität
- Änderungen jeweils für jedes neue Release möglich
 - Selten während eines Releases
 - Kunde bestimmt Release-Umfang, bringt neue Anforderungen ein
- Adaptivität
 - Evolutionäre Architektur (wird über Refactoring angepasst)
 - Evolutionärer Prozess (wird von Entwicklern angepasst)
- Kein wirkliches Change-Request-Verfahren (mit CR = neuer Vertrag)
 - Änderungen werden in neue Releases eingeplant
 - Für eine Änderung wird etwas anderes nicht getan (Priorität), ggf. wird insgesamt später ausgeliefert

Reaktion auf Änderungen - Voraussetzungen

- Entwicklungswerkzeuge müssen leichte Änderungen erlauben
- Keine schweren Altlasten („da traut sich keiner mehr ran“)
- Keine Alternierenden Anforderungen (wg. politischer Konflikte)
- Verträge
 - Nach Aufwand (Abklären mit dem Einkauf schwierig)
 - Festpreis, aber vagem Funktionsumfang, ggf. Streichkandidaten ausweisen
 - Agil und öffentliche Ausschreibungen – Widerspruch!

Das Anpassungsproblem (Projektgröße, Kritikalität)

- Crystal: mehrere Methoden
 - Clear, Yellow, Orange
 - Nach Kritikalität: C = loss of comfort bis L = loss of life
 - Nach Projektgröße: 6, 20, 40
- XP – nicht Skalierbar
- Scrum: Hierarchie von Scrums
- RUP / V-Modell: Tayloring



Zusammenfassung

- Agile Methoden interessant
 - Öffnen den Blick für das Wesentliche in der Software-Entwicklung (andere Perspektive auf viele Probleme)
 - Sparen richtig eingesetzt Kosten (div. Erfahrungsbeispiele)
 - Verkürzen richtig eingesetzt Entwicklungszeiten
 - Machen Spaß
- Agile Methoden = andere **KULTUR**, anderes Wertesystem
 - Management Akzeptanz, Einkauf, Kunden, Entwickler
- Agile Methoden sind nicht immer und überall passend
 - Großes Team, kritische Anforderungen, feste Verträge (z.B. für Bund?)
 - Agiler Operations-Roboter?, Agiles Kernkraftwerk !?, Agiler Airbag !?
- Agilität häufig Geschäftemacherei
 - extrem Power-Pointing, extrem Labering, extrem Hyping
 - „Agil“ gerade Vertriebsargument

Überblick

- Warum agile Methoden
- Agile Ansätze
- Scrum

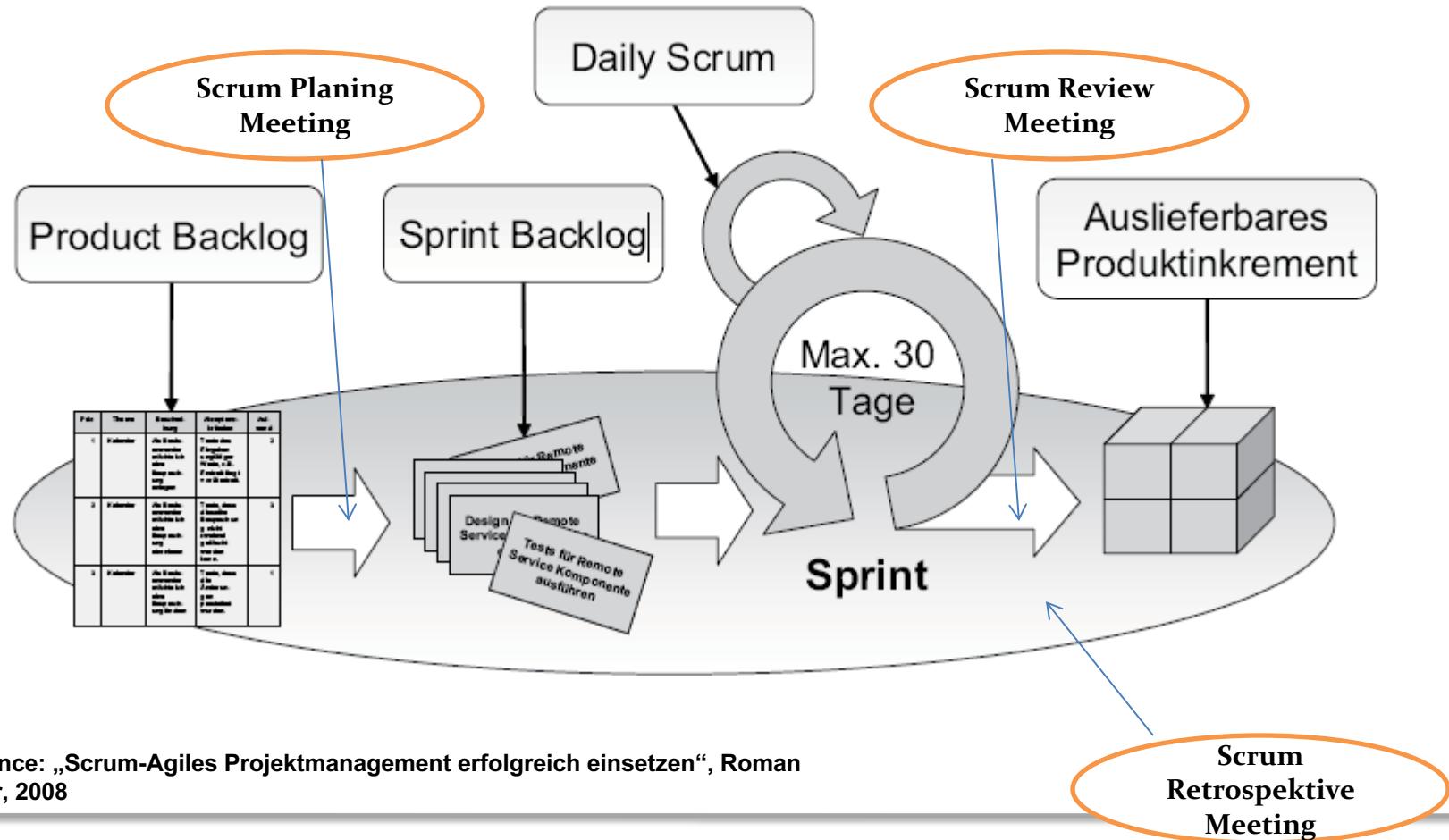
Was ist Scrum?-Scrum im Überblick

- Scrum ist ein agiles Managementframework zur Entwicklung von Software, das aus wenigen klaren Regeln besteht.
- Scrum Charakteristika
 - Leichtgewichtiger Management Prozess
 - Das Kernelement sind kurze Iterationen: Sprints
 - Regelmässige Produktinkremente
 - Kleine Teams- Selbst organisieren
 - Mehr Kommunikation und Kollaboration

Scrum- der Rahmen

Rollen	Meetings	Artefakte
<ul style="list-style-type: none">• Produkt-Owner• Team• ScrumMaster	<ul style="list-style-type: none">• Sprint-Planungssitzung• Daily-Scrum-Besprechung• Sprint-Review• Sprint-Retrospektive	<ul style="list-style-type: none">• Product Backlog• Sprint Backlog• Sprint-Burndown-Bericht

Ablauf von Scrum



Reference: „Scrum-Agiles Projektmanagement erfolgreich einsetzen“, Roman Pichler, 2008

Scrum- der Rahmen

Rollen

- Produkt-Owner
- Team
- ScrumMaster

Meetings

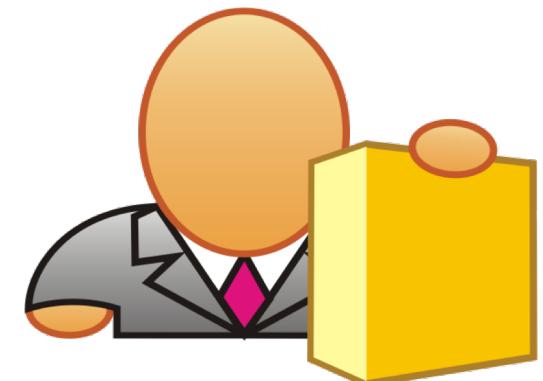
- Sprint-Planungssitzung
- Daily-Scrum-Besprechung
- Sprint-Review
- Sprint-Retrospektive

Artefakte

- Product Backlog
- Sprint Backlog
- Sprint-Burndown-Bericht

Der Product Owner

- Anforderungsbeschreibung und –management
 - das Erfassen der Kundenbedürfnisse und die Beschreibung der Anforderungen
 - Priorisiert Anforderungen
 - Eine enge und regelmäßige Abstimmung mit den Kunden
- Releasemanagement und Return on Investment
 - Entscheidet über Auslieferungszeitpunkt, Funktionalität und Kosten der Softwareversion
- Enge Zusammenarbeit mit dem Team
- Erfassung des Projektfortschritts



Das Team

- Klein
 - typischerweise 5-9 Personen
- Interdisziplinär besetzt
 - Funktionsübergreifend: QS, Programmierer, Architekt usw.
- Bevollmächtigt
- Selbstorganisiert
- Vollzeitteammitglieder
- Arbeitsplätze in unmittelbarer Nähe



Der ScrumMaster

- Verantwortlich für die Einhaltung von Scrum-Werten und -Techniken
- Beseitigt mögliche Hindernisse
- Stellt sicher, dass das Team vollständig funktional und produktiv ist
- Unterstützt die enge Zusammenarbeit zwischen allen Rollen und Funktionen
- Aber keine Autorität

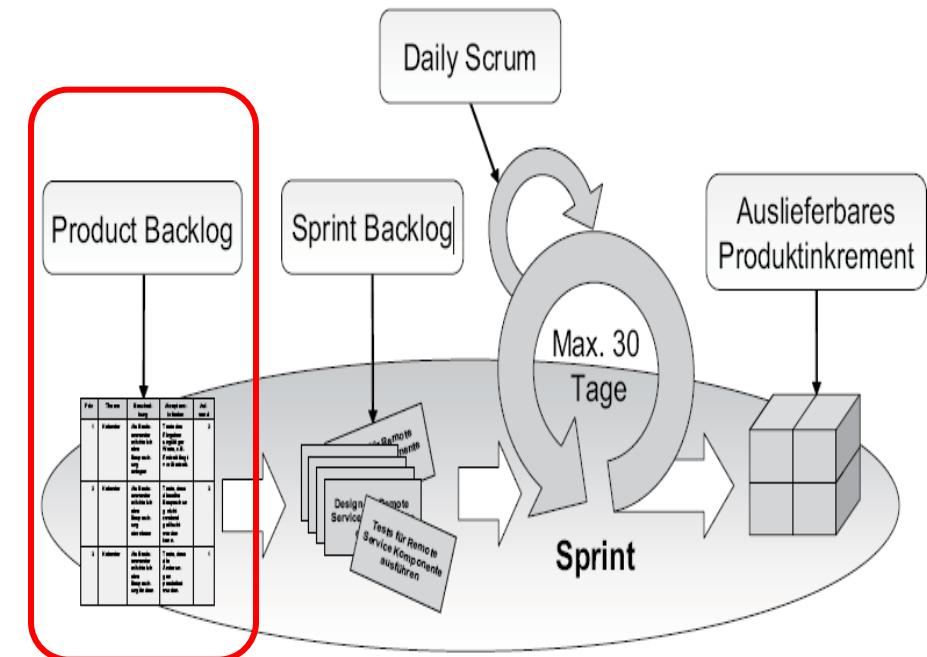


Scrum- der Rahmen

Rollen	Meetings	Artefakte
<ul style="list-style-type: none">• Produkt-Owner• Team• ScrumMaster	<ul style="list-style-type: none">• Sprint-Planungssitzung• Daily-Scrum-Besprechung• Sprint-Review• Sprint-Retrospektive	<ul style="list-style-type: none">• Product Backlog• Sprint Backlog• Sprint-Burndown-Bericht

Das Product Backlog

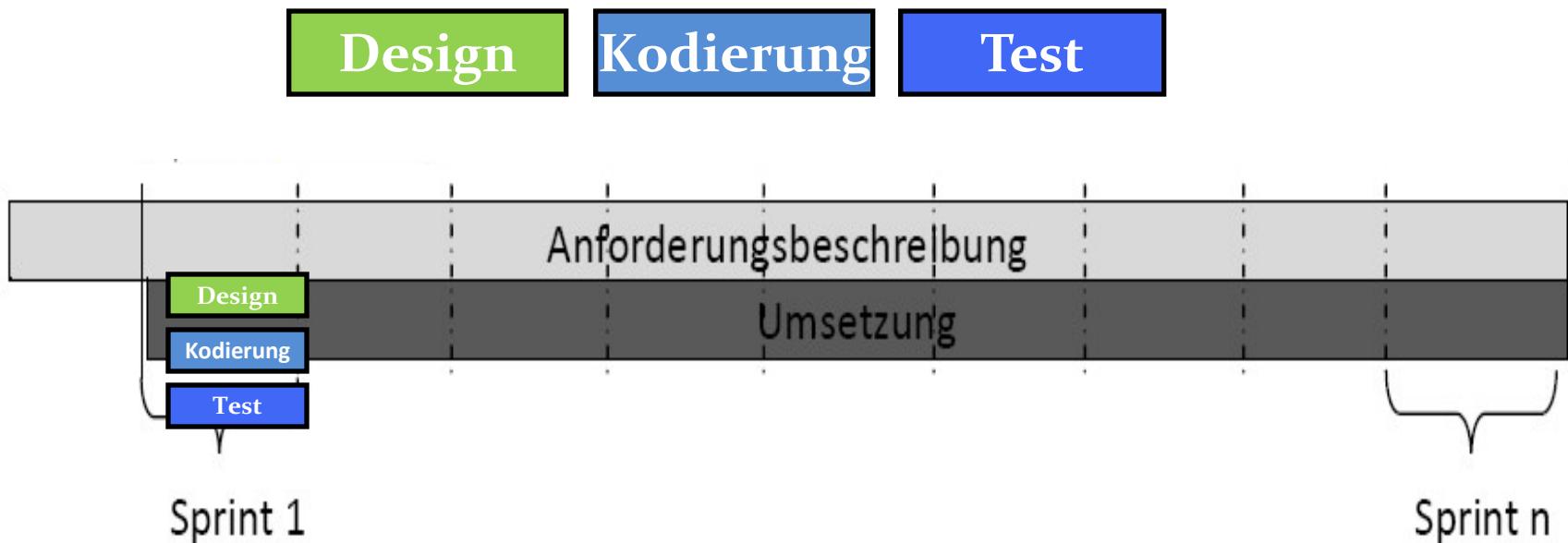
- Zum Erfassen und Managen von Anforderungen in Scrum
- Von der Product Owner erstellt und verfeinert wird
- Das Product Backlog ist ein lebendes Dokument
- Die Einträge sind priorisiert
- Die Einträge sind abgeschätzt



Reference: „Scrum-Agiles Projektmanagement erfolgreich einsetzen“, Roman Pichler, 2008

Anforderungen

- Definitions- und Umsetzungsphase in der traditionellen Softwareentwicklung
- Anforderungsbeschreibung in Scrum



Das Product Backlog -Beispiel

	Item #	Description	Est	By
Very High				
	1	Finish database versioning	16	KH
	2	Get rid of unneeded shared Java in database	8	KH
	-	Add licensing	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	Enforce unique names	-	-
	7	In main application	24	
	8	In import	24	A
	-	Admin Program	-	
	9	Delete users	-	JM
	-	Analysis Manager	-	-
		When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab		
	10		8	TG
	-	Query	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	Population Genetics	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	Add icons for v1.1 or 2.0	-	-
	-	Pedigree Manager	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	Explorer	-	-
		Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	21		4	T&A
	22	Delete settings (?)		

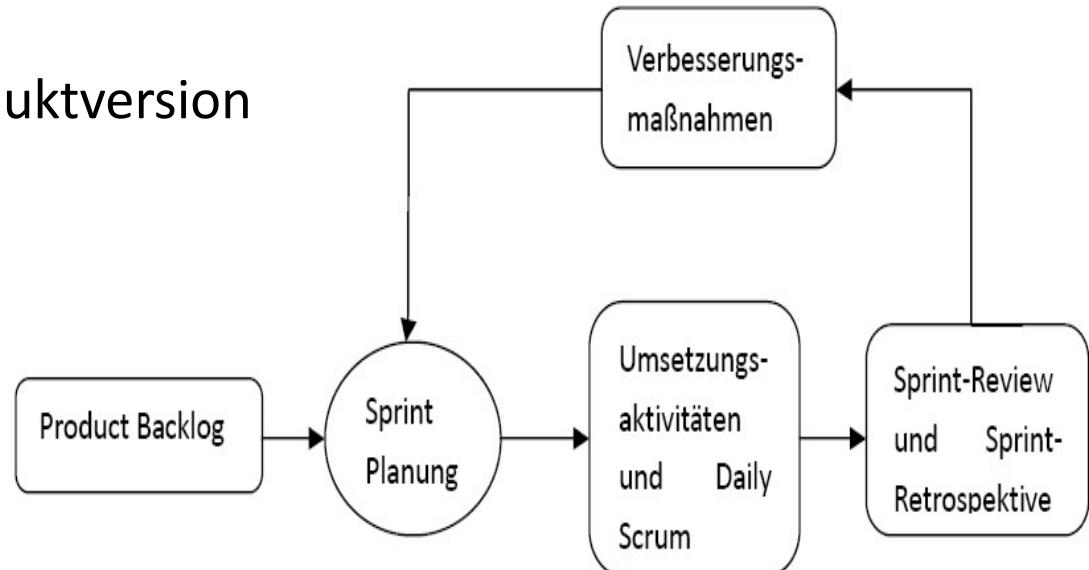
Reference: www.mountaingoatsoftware.com/product_backlog

Scrum- der Rahmen

Rollen	Meetings	Artefakte
<ul style="list-style-type: none">• Produkt-Owner• Team• ScrumMaster	<ul style="list-style-type: none">• Sprint-Planungssitzung• Daily-Scrum-Besprechung• Sprint-Review• Sprint-Retrospektive	<ul style="list-style-type: none">• Product Backlog• Sprint Backlog• Sprint-Burndown-Bericht

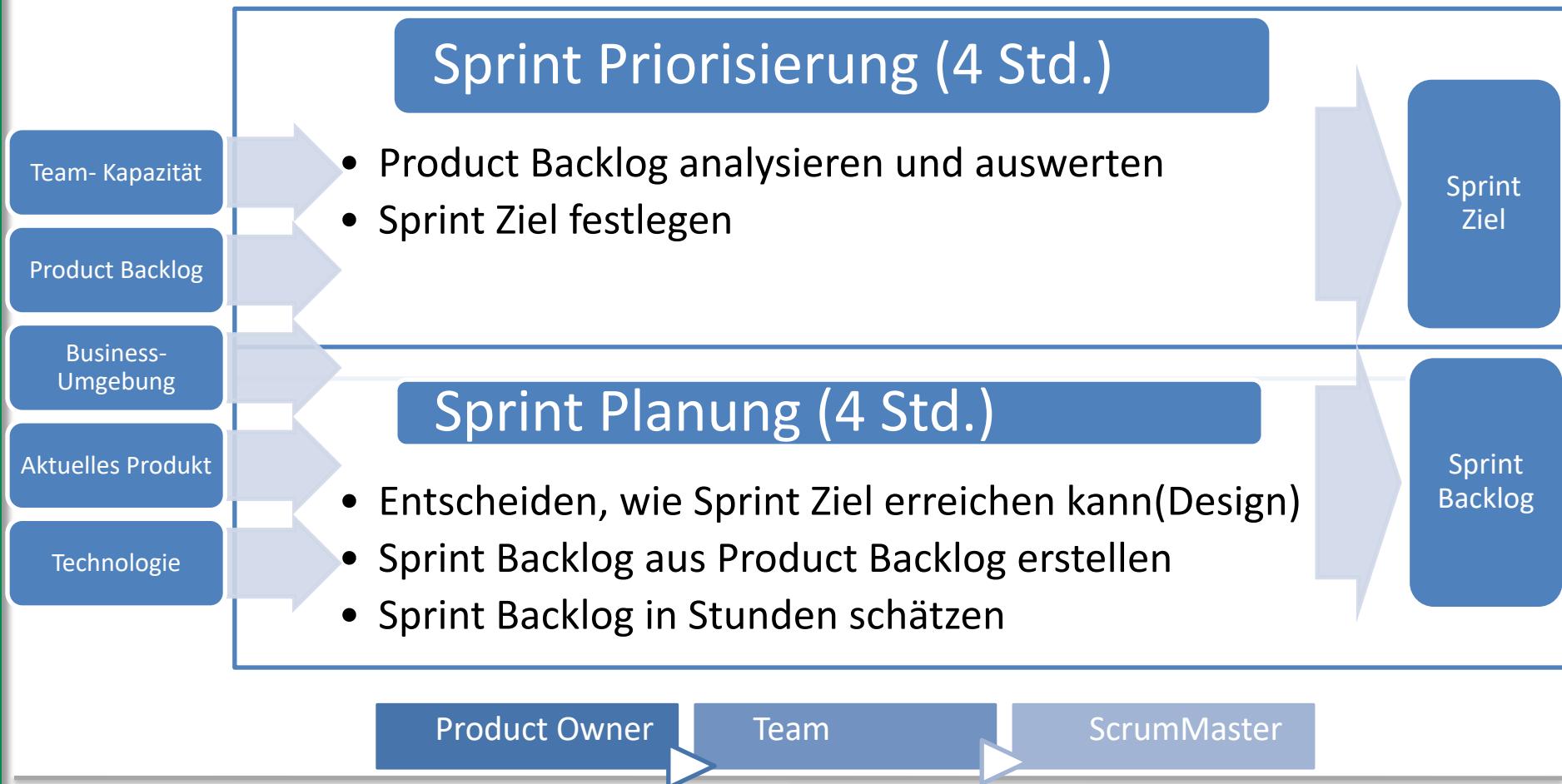
Sprints

- Der Sprint-Ablauf
 - Dauer: maximal 30 Tage
 - Team fokussiert auf Sprint Goal
 - Völlig autonom bzgl. Vorgehen
 - Das Produkt wird während des Sprints entworfen, kodiert und getestet
 - Produziert neue Produktversion



Reference: „Scrum-Agiles Projektmanagement erfolgreich einsetzen“, Roman Pichler, 2008

Die Sprint-Planungssitzung



Das Sprint Backlog

- Definiert Arbeiten/Tasks, die das Team in einem Sprint erledigt
- Die geschätzte restliche Arbeit wird täglich aktualisiert
- Jedes Team-Mitglied kann Tasks hinzufügen, löschen oder ändern, Nur Team kann/darf Sprint Backlog verändern
- Neue, für den Sprint benötigte Arbeit taucht auf
- Taskgröße zwischen 4 - 16 Stunden Aufwand

Das Sprint Backlog - Beispiel

Who	Description	Days Left in Sprint				
		15	13	10	8	
-	User's Guide					
SM	Start on Study Variable chapter first draft	1E	16	16	16	
SM	Import chapter first draft	4C	24	6	6	
SM	Export chapter first draft	24	24	24	6	
Misc. Small Bugs						
JM	Fix connection leak	4C				
JM	Delete queries	8	8			
JM	Delete analysis	8	8			
TG	Fix tear-of messaging bug	8	8			
JM	View pedigree for kindred column in a result set	2	2	2	2	
AM	Derived kindred validation	8				
Environment						
TG	Install CVS	1E	16			
TBD	Move code into CVS	4C	40	40	40	
TBD	Move to JDK 1.4	8	8	8	8	
Database						
KH	Killing Oracle sessions	8	8	8	8	
KH	Finish 2.206 database patch	8	2			
KH	Make a 2.207 database patch	8	8	8	8	
KH	Figure out why 461 indexes are created	4				

Reference: www.mountaingoatsoftware.com/sprint backlog

Die Daily-Scrum-Besprechung

- Ziel:
 - Unterstützung der Selbstorganisation des Teams
 - Identifizierung der Hindernisse
- Parameter:
 - Täglich
 - Maximal 15 Minuten lang
 - Stand-up
- Teilnehmer:
 - Alle sind eingeladen, aber nur Team-Mitglieder, der ScrumMaster, und der Produkt-Owner dürfen reden
- Informationsaustausch des Teams untereinander

Die Daily-Scrum-Besprechung

- Jedes Teammitglied beantwortet drei Fragen:

1

- Welche Aktivitäten habe ich seit der letzten Daily Scrum abgeschlossen?

2

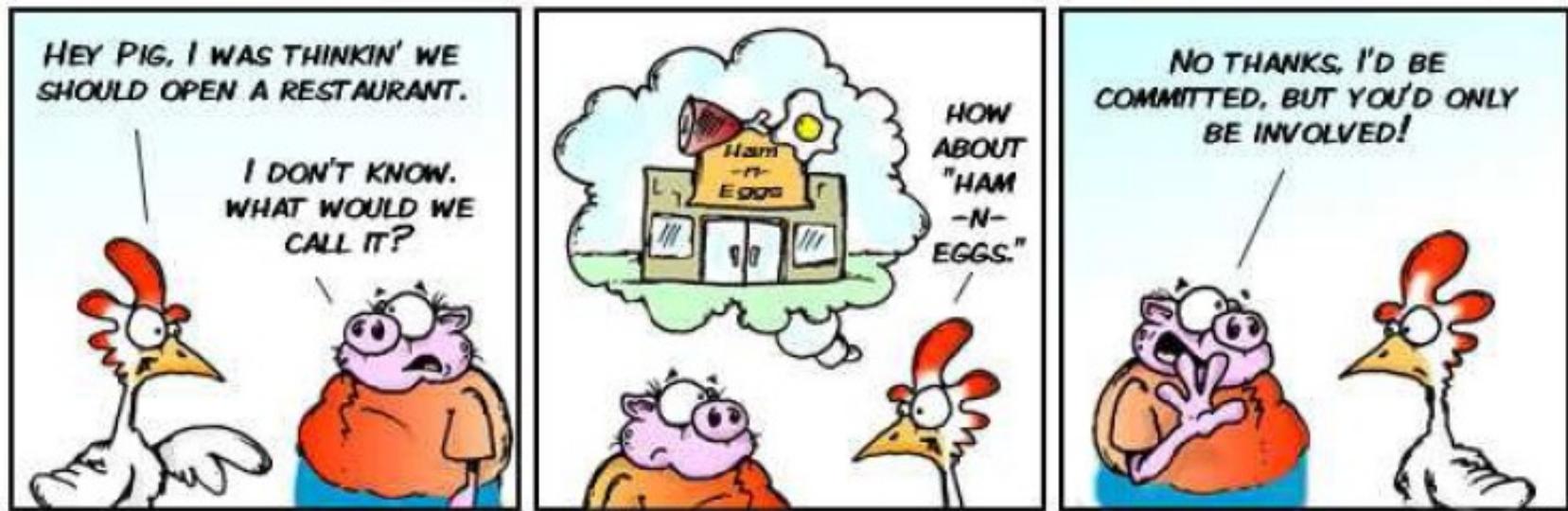
- Woran plane ich bis zur nächsten Daily Scrum zu arbeiten?

3

- Werde ich in irgendeiner Form an der Ausführung einer Aktivität behindert?

Die Daily-Scrum-Besprechung

- „Schweine“
 - Product Owner, Team und ScrumMaster
- „Hühner“
 - Alle anderen Interessenvertreter



@2006 implementingscrum.com

Das Sprint-Reviewmeeting

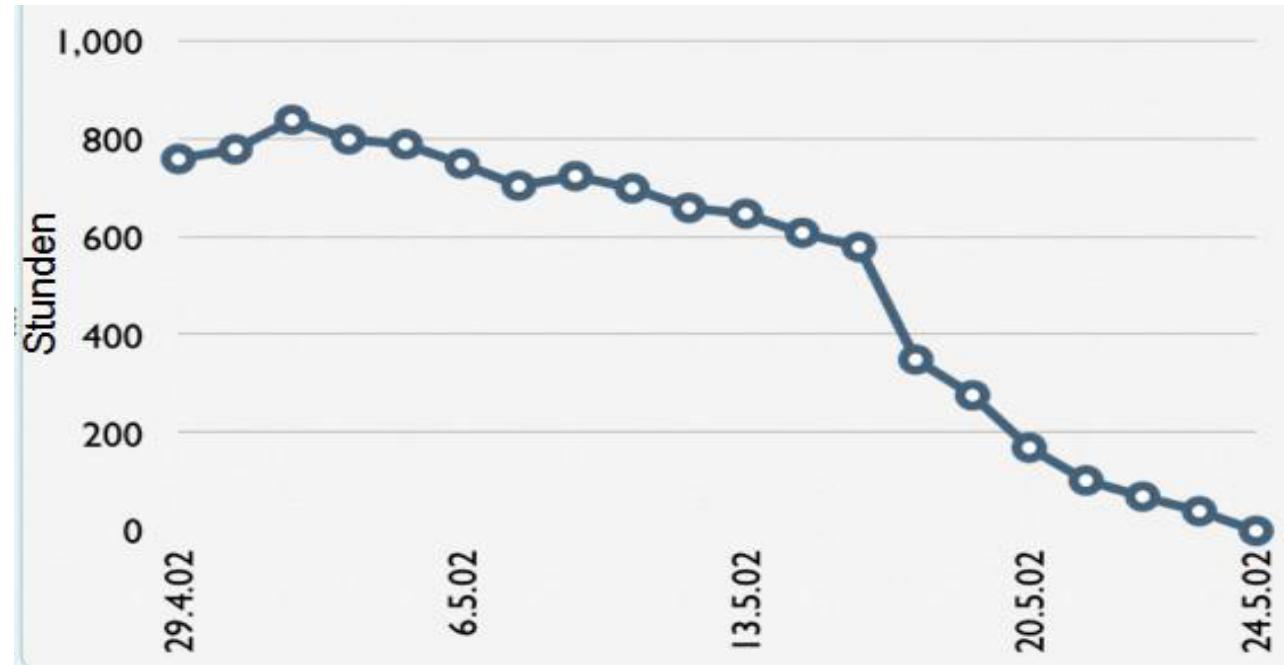
- Am Ende des Sprint findet statt
- Ziel
 - Gutachtung der entstandenen Arbeitsergebnisse
 - Projektfortschritt transparent machen
- Das Team stellt neue Produktversion vor und berichtet über Verlauf des Sprints
 - Typischerweise in Form einer Demo der neuen Features oder der zugrunde liegenden Architektur
- Informell
 - Ein bis zwei Stunden
 - Keine Folien
- Teilnehmer
 - Product Owner, Team und ScrumMaster nehmen teil
 - Laden andere Interessenvertreter
- Rückmeldung über die entstandene Software von Kunden und andere

Die Sprint-Retrospektive

- Unmittelbar im Anschluss an das Sprint-Review statt und schließt den Sprint ab
- Zwischen 1,5 – 2,5 Stunden
- Teilnehmer
 - Product Owner, Team und ScrumMaster
 - Nach Bedarf Endkunden oder anderen Personen
- Offene und ehrliche Meinungen über die Zusammenarbeit im Projekt
- Unterstützung eines kontinuierlichen Verbesserungsprozess
- Führung zur Steigerung der Produktivität, der Leistungsfähigkeit des Teams und der Softwarequalität

Der Sprint-Burndown-Bericht

- Eine tägliche aktualisierte graphische Darstellung des noch zu erbringenden Restaufwands für den aktuellen Sprint



Reference: agiletoday.com/scrumoverview.html