

# Projektmanagement

## Agiles PM



## Agiles PM

### **Vorgehensmodelle (zur Erinnerung)**

- Sequentielle vs. inkrementelle Modelle
- Umgang mit Komplexität

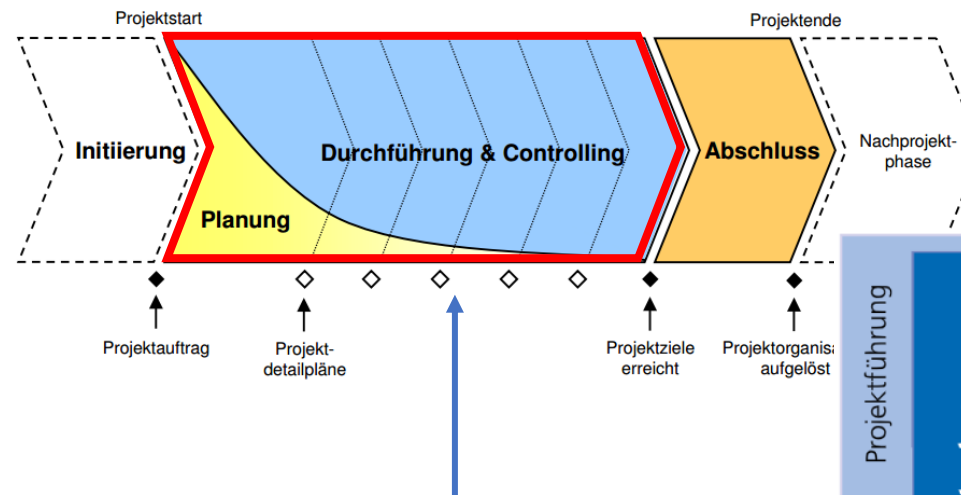
### **Scrum als Beispiel eines agilen Vorgehensmodell**

- Rollen, Prozesse, Artefakte
- Best practices in den Sprintphasen

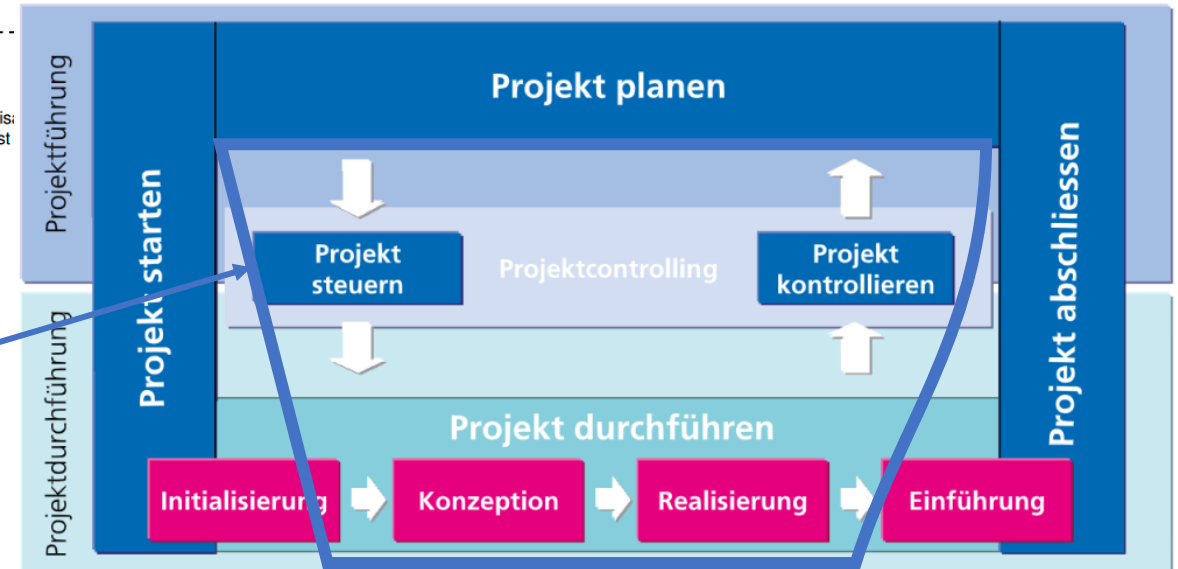
### **Erfahrungen und Überlegungen zur Einführung von Scrum**

- Unternehmenskultur und Erfahrungen

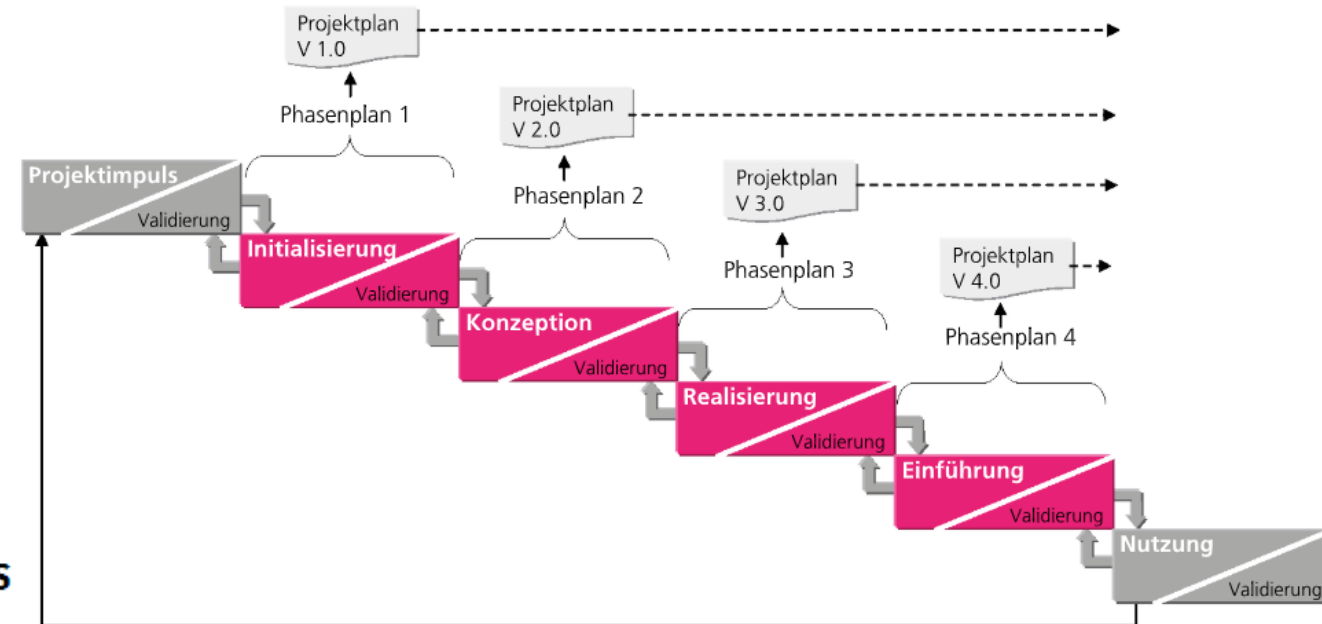
# Einordnung von agilem PM in die allgemeinen Projektphasen



Im Folgenden sprechen wir über das Vorgehen innerhalb der Planungs-, Durchführung & Controlling-Phase.



- Bei einem konstruktivistischen Vorgehensansatz laufen die Phasen linear ab.
- Vor Beginn jeder Phase muss die vorangegangene Phase abgeschlossen sein
- der Auftragnehmer wird meistens nur das realisieren, was der Auftraggeber explizit am Beginn verlangt hat.
- der „Plan“ als Artefakt wird in den Mittelpunkt gestellt, statt dem Prozess von Retrospektive und Change



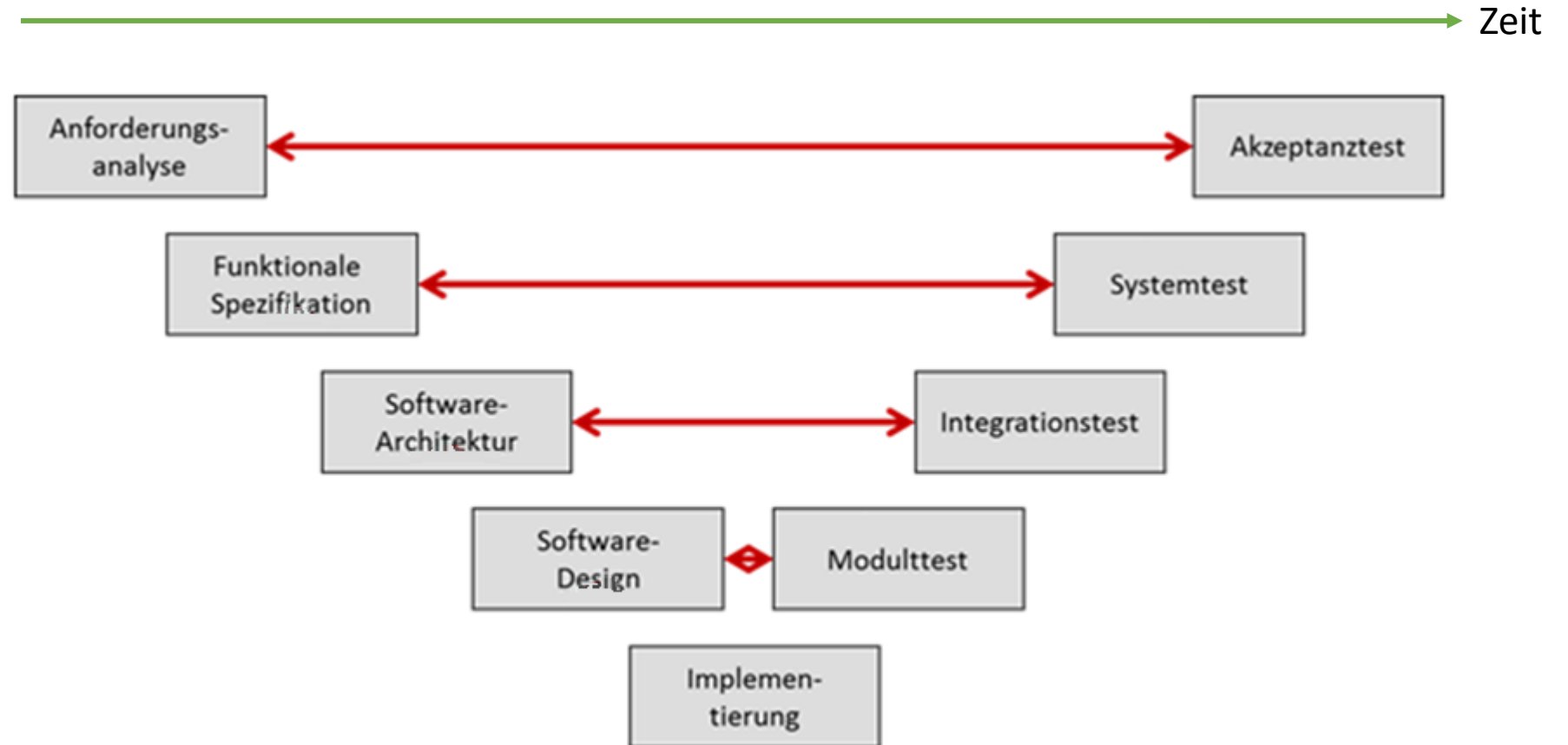
**BERTOLT BRECHT**

**BALLADE VON DER UNZULÄNGLICHKEIT MENSCHLICHEN PLANENS**

Ja, mach nur einen Plan!  
Sei nur ein großes Licht!  
Und mach dann noch‘nen zweiten Plan  
Gehn tun sie beide nicht.

(Jenny, 2014)

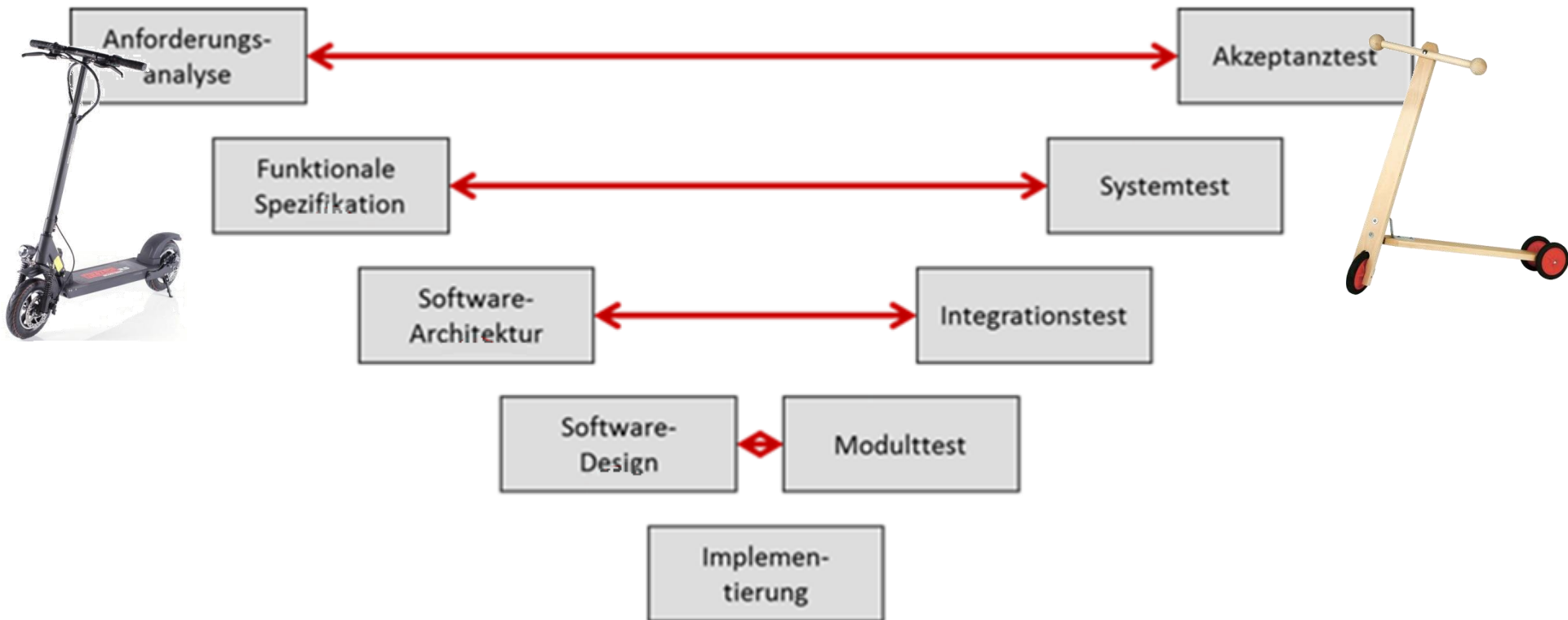
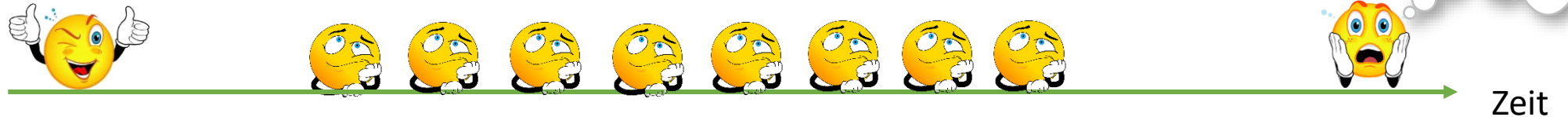
# V-Modell der deutschen Sicherheitsindustrie (Aviation, Automotive,...)



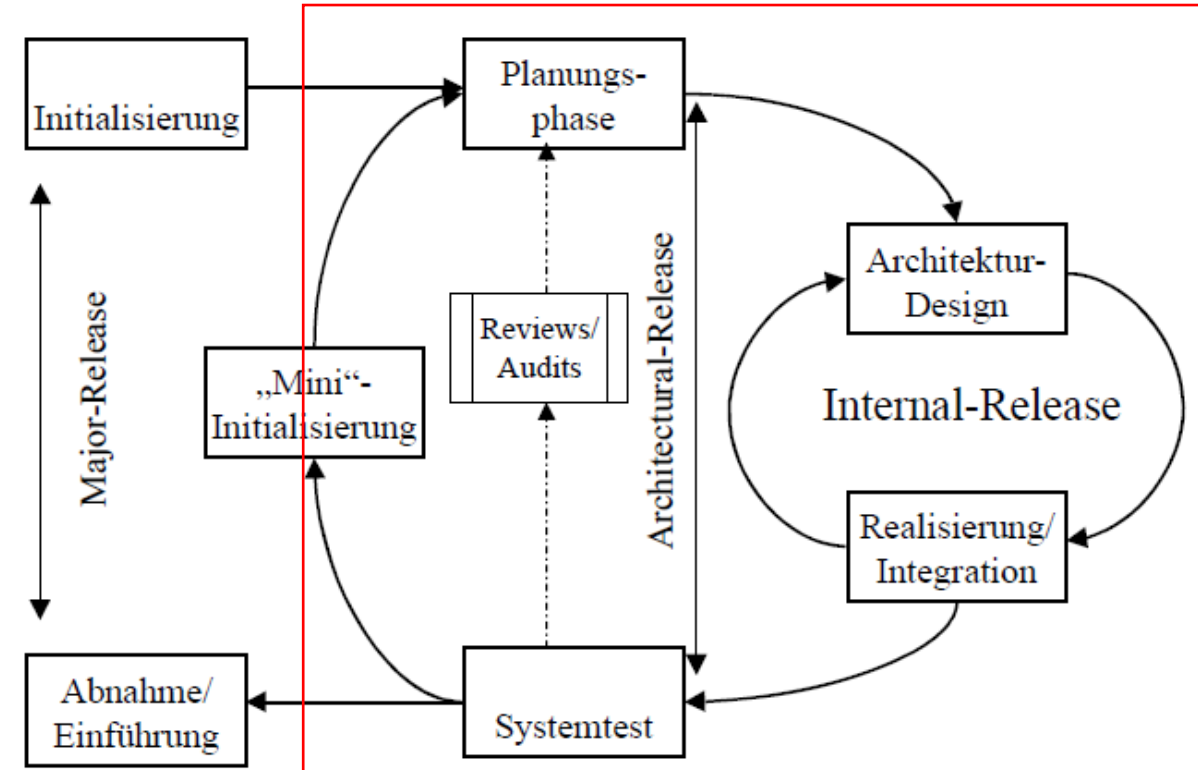


# Wasserfallentwicklung

## Frühes Commitment für späte Enttäuschung



- Vorgehen bedingt **keinen sequenziellen Projektablauf** mehr, sondern erlaubt **ein wiederholtes Zurückgreifen auf vorherige Ergebnisse** und Tätigkeiten früherer Phasen
- **Auf Veränderungen** von Zielen, Rahmenbedingungen und der Umgebung kann **rasch reagiert** werden
- **Sukzessive Bereitstellung** funktionstüchtiger Teilprodukte
- Vorgehensmodell **bedingt sehr sorgfältige Überlegungen** und stellt wesentlich höhere Anforderungen an die Planung als ein konstruktivistisches Vorgehensmodell



Agile Entwicklungsvorgehen wie Scrum basieren auf diesem Ansatz!

(Wieczorrek und Mertens, 2011)



# 2 x 3 Dinge aus der Praxis...

## 3 Dinge, von denen wir wünschen, sie wären wahr...

- Der Kunde weiß, was er will
- Die Realisierer wissen, wie es zu erstellen ist
- Nichts ändert sich im Projektverlauf

## 3 Dinge, mit denen wir leben müssen...

- Der Kunde entdeckt erst spät, was er will
- Die Realisierer entdecken erst, wie es am Besten zu erstellen ist
- Vieles ändert sich im Projektverlauf



(Komus, 2016)



# Weshalb agile Methoden?

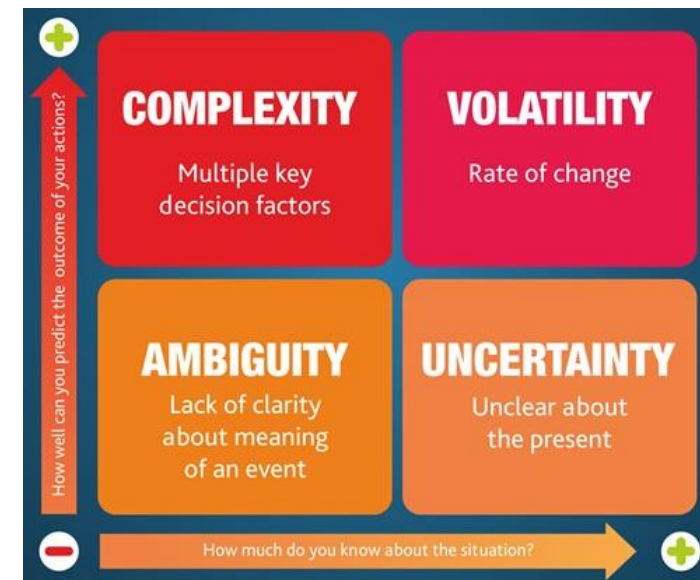
## Herausforderungen von Komplexität und Unsicherheit

### Management in der stabilen Welt (klassische Sicht, „Illusion of Control“)

- Dinge sind kompliziert, aber berechenbar
- Langfristige Planung ist im Detail möglich und sinnvoll
- Kennzahlen, Command and Control, Management im Detail (Micromanagement) ist legitim – dazu gehört auch das Micromanagement von Entwicklungsteams

### Management in der VUCA-Welt (Bennett und Lemoine, 2014)

- Langfristige Detailplanung nicht sinnvoll/möglich
- Eigenverantwortung notwendig (Schwarmintelligenz)
- Lernen und Experimentieren sind der Schlüssel



# Framework für Analyse: Komplexität des Projektkontextes

## Simpel

- Bekannt, beherrschbar
- Machen!

## Kompliziert

- Nicht einfach, aber Ursache-Wirkung berechenbar
- Planbar im Detail
- Lineares Vorgehen mögl. / agile Elemente

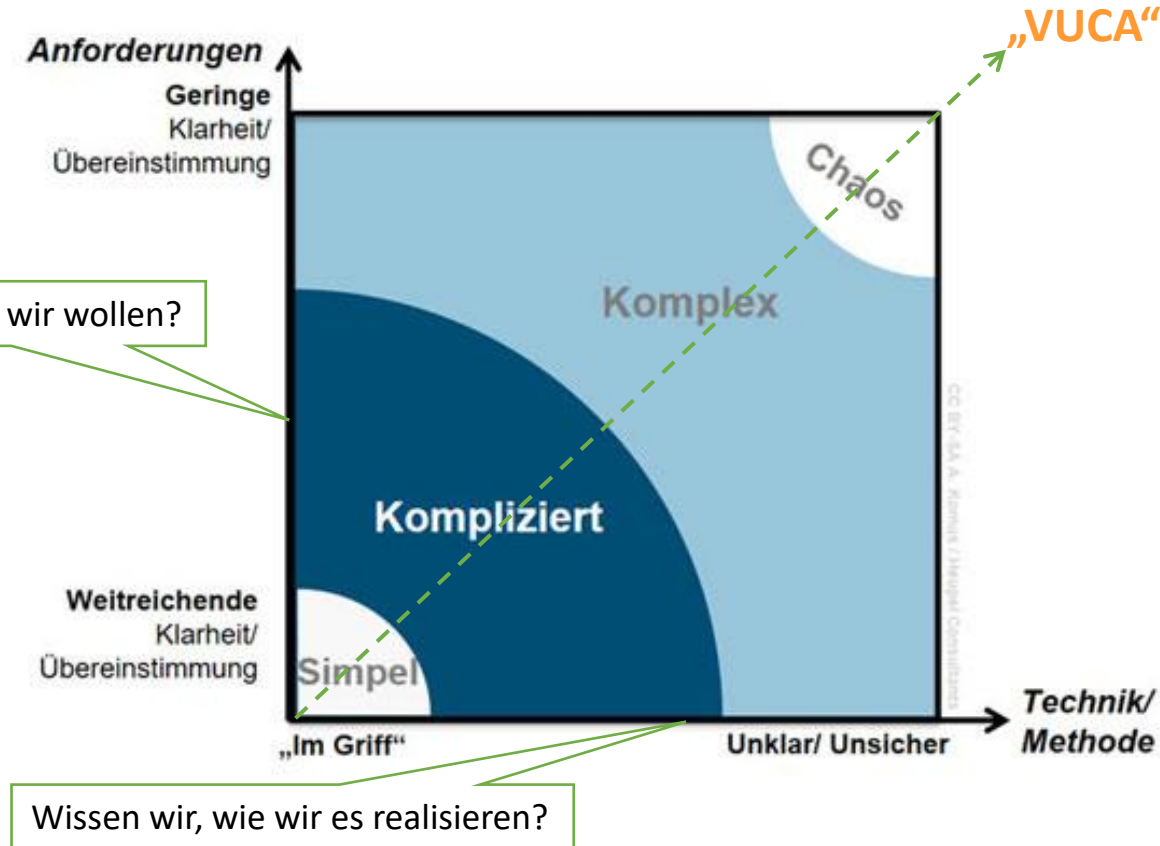
## Komplex

- Ursache Wirkung nicht mehr sicher nachvollziehbar
- agiles Vorgehen sinnvoll/notwendig
- „Inspect and Adapt“

## Chaotisch

- Systemverhalten unbekannt, nicht steuerbar
- Vermeiden...

Modifizierte „Stacey“-Matrix nach (Stacey, 1996)



# Nützliche Strategien im Umgang mit Komplexität

Unterscheidungen treffen zwischen **kompliziert, komplex und chaotisch**

- Vorgehen darauf abstimmen, eigene Muster erkennen

Gewisse **Flexibilität der Ziele** und Vorgehen

Ausprobieren in **kleinen Schritten**, Ergebnisse prüfen, justieren

Fehlertoleranz, **Fehlerkultur** aufbauen

- Bei Fehler keine Suche nach Schuldigen, sondern nach Lösungen

**Selbstorganisation („Subsidiarität“)**

- Da Top-Down-Steuerung nicht möglich/sinnvoll, Aufbau selbstregelnder Systeme, idealerweise mit einfachen Regeln
  - Bsp. Kreisverkehr: „Wer drin ist hat Vorfahrt“ → Verkehr regelt sich selbst
  - Mitdenken und Kreativität fördern statt Ausführung nach Vorschrift

"Totò disse che in tempo di crisi gli intelligenti cercano soluzioni mentre gli imbecilli cercano colpevoli.«

«Totò sagte, dass in Krisenzeiten kluge Menschen nach Lösungen suchen, während Dummköpfe nach Schuldigen suchen... «

Totò - bekannter italienischer Schauspieler





# 2001: Xtreme Programming und Agiles Manifest



- Agiles Ziel: Stop Starting - Start Finishing!

Frühes Ändern statt spätem Scheitern (agil = wendig, anpassungsfähig)

- Kunde hochfrequent involvieren, nach/ vor jedem Sprint
- Fokus auf lieferbaren Features



# Werte des Agilen Manifests



# Prinzipien hinter dem agilen Manifest

- Höchste Priorität ist es, den **Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software** zufrieden zu stellen.
- Heiße selbst **späte Anforderungsänderungen** 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 Entwicklungsteam zu übermitteln, ist im **Gespräch von Angesicht zu Angesicht**.



# Verbreiteter Irrtum: „Twice the work in half the time“

„Agiler“ heisst nicht: „Schneller“!

- Schlägt der Hase Haken, um schneller ans Ziel zu kommen???
- Frühe Variante agilen Vorgehens: die „Sowosamma-Neger“ (Fredl Fesl, 1985!):

Der bayerische Barde Fredl Fesl erzählte die Geschichte eines dunkelhäutigen afrikanischen Volkes, die in der Savanne lebten.

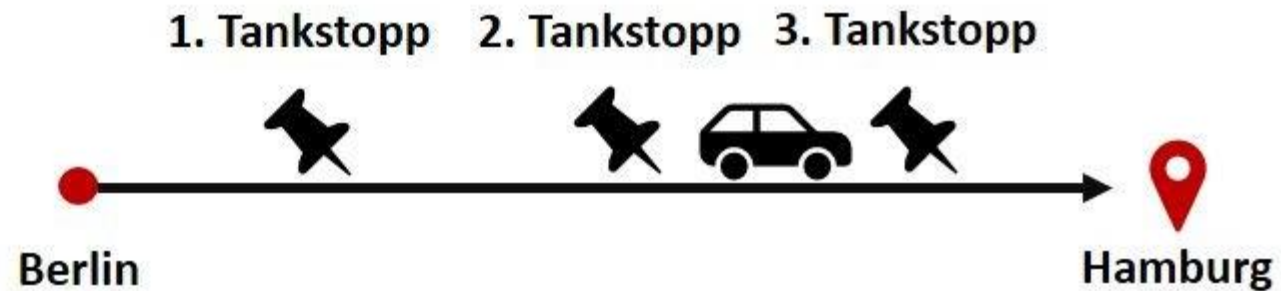
„...Ihr Problem war, dass das Savannengras so hoch war - oder sie zu klein ... jedenfalls mussten sie, wenn sie Ausschau halten wollten, immer in die Höhe springen und dabei riefen sie „So. Wo samma?“ Und so kam das Volk zum Namen der „Sowosamma-Neger“ ...“

(1985 war dieser politisch unkorrekte Begriff in der BRD noch gebräuchlich und gestattet)

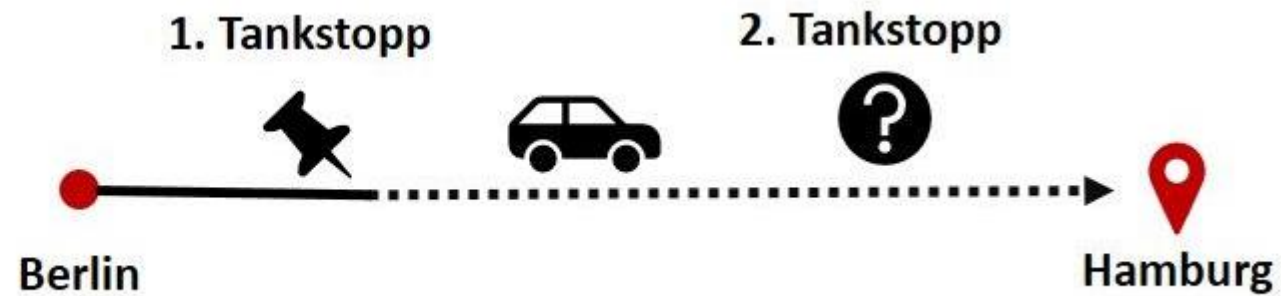


# Agile Planung

## Projektstruktur- planung

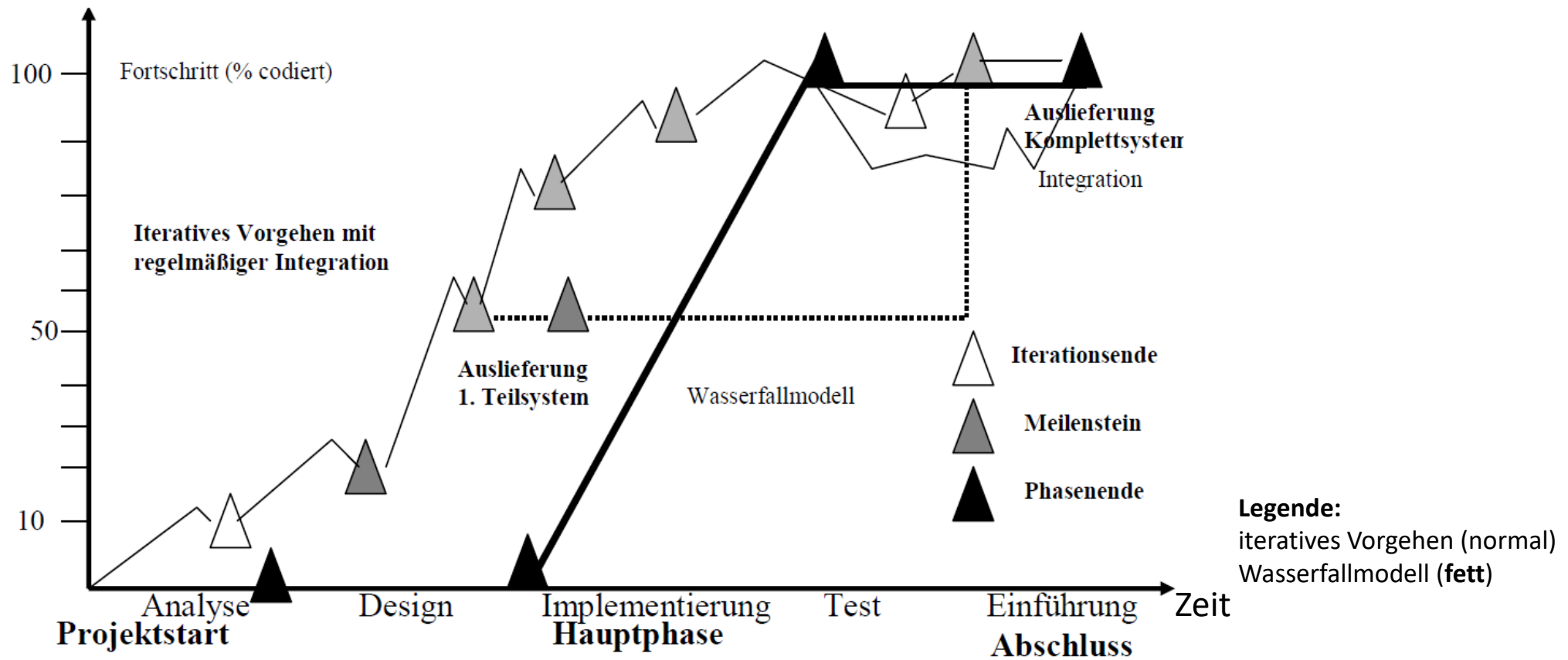


## Agile Planung





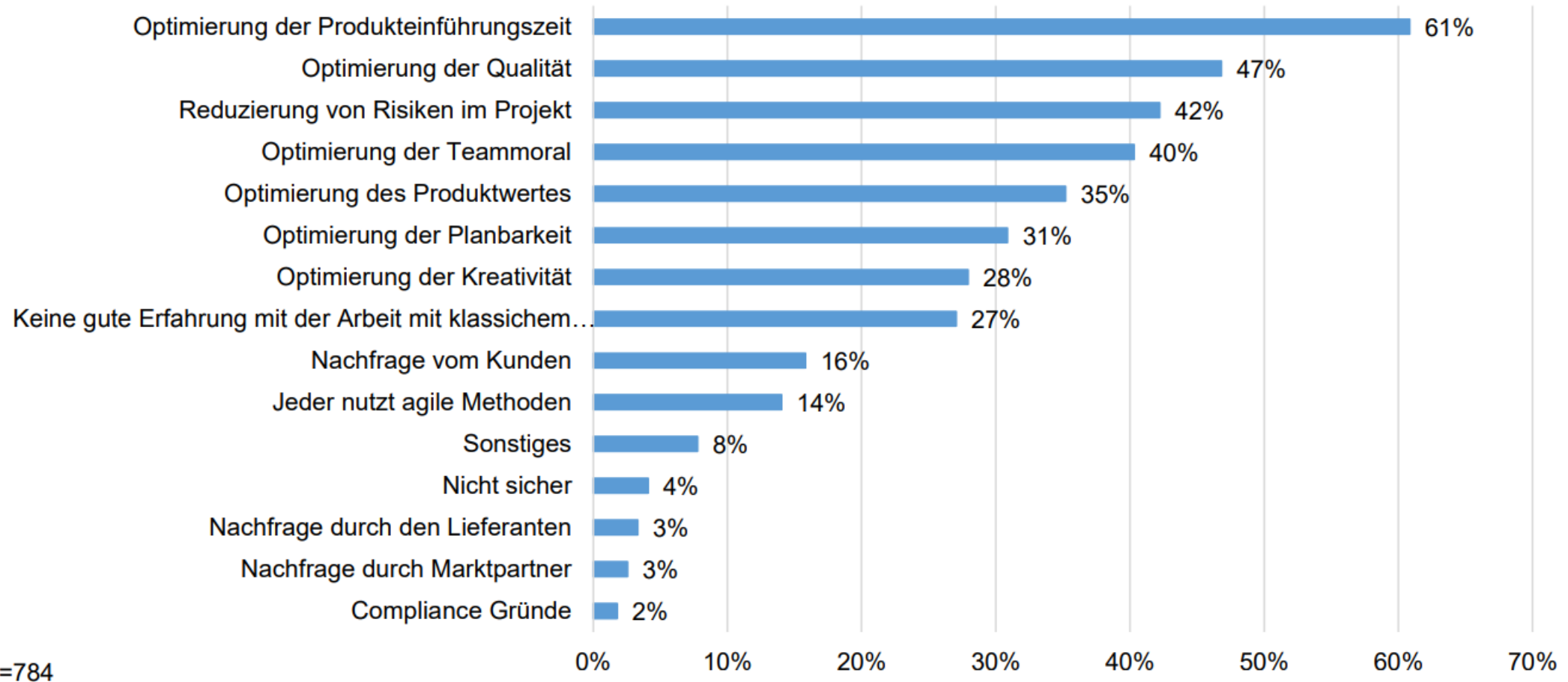
# Gegenüberstellung Agiles Vorgehen – Wasserfallmodell



(Wieczorrek und Mertens, 2011)

# Gründe für die Verwendung agiler Methoden

***Warum hat sich Ihr Unternehmen dazu entschlossen mit agilen Methoden zu arbeiten?***



## **Ermächtigung und Selbstorganisation**

- Teams sind eigenverantwortlich und selbstorganisiert.

## **Frühe, regelmäßige Lieferung**

- Frühe Lieferung von Inkrementen ermöglicht das frühe Einholen von Feedback.

## **Überprüfung und Anpassung (plan - do – act –change)**

- Durch die Teams werden in Reviews regelmäßig die Ergebnisse und Vorgehensweisen reflektiert, um sie effizienter zu gestalten

## **Transparenz**

- (Persönlicher) Austausch von Informationen im Team und nach außen

## **Festlegen von überschaubaren Inkrementen**

- Definierter Anfang und Ende fördert Disziplin, Fokus und pünktliche Lieferung

# Scrum als Beispiel eines agilen Vorgehensmodells

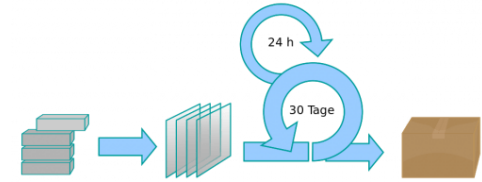
- Scrum
- Kanban („IT-Kanban“. Eigene Methodik neben dem klassischen Kanban in der Logistik)
- Extreme Programming
- Feature Driven Development (FDD)
- Lean
- Design Thinking
- Adaptive Software Development
- Agile Modelling
- Usability Driven Development
- Dynamic System Development Method
- Rational Unified Process (RUP)
- Crystal
- ...



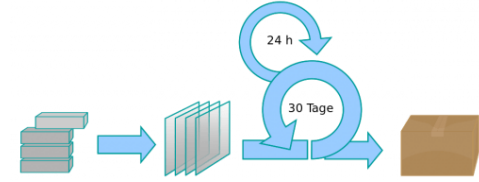
# Scrum - Gedränge



# SCRUM – Eigenschaften und Prinzipien



- Scrum (englisch für „Gedränge“) ist ein **Vorgehensmodell der Softwareentwicklung**
- Ansatz von Scrum: **empirisch, inkrementell und iterativ**  
(die meisten modernen Entwicklungsprojekte sind **zu komplex, um durchgängig planbar** zu sein)
- Scrum versucht, die Komplexität durch drei Prinzipien zu reduzieren:
  - **Transparenz:** Der Fortschritt und die Hindernisse eines Projektes werden täglich und für alle sichtbar festgehalten.
  - **Überprüfung:** In regelmäßigen Abständen werden Produktfunktionalitäten geliefert und beurteilt.
  - **Anpassung:** Die Anforderungen an das Produkt werden nicht ein und für alle Mal festgelegt, sondern nach jeder Lieferung neu bewertet und bei Bedarf angepasst.
  - **ABER: Während** eines Sprints werden die Anforderungen „eingefroren“ - damit ist eine „**Wasserfallartige**“ **Entwicklung im Sprint** möglich



Ziel ist die **inkrementelle und qualitativ hochwertige** Bereitstellung eines Produktes auf dem Weg zu einer zu Beginn formulierten Vision

- Die Umsetzung der Vision in das fertige Produkt erfolgt **nicht durch die Aufstellung möglichst detaillierter Anforderungslisten** ( Lastenheft / Pflichtenheft), die dann phasenweise umgesetzt werden.
- Statt dessen werden **klare Funktionalitäten aus Anwendersicht** formuliert, die dann in **zwei bis vier Wochen langen, sich wiederholenden Intervallen, sogenannten Sprints**, iterativ und inkrementell umgesetzt werden



# SCRUM – schrittweise Entwicklung in Sprints

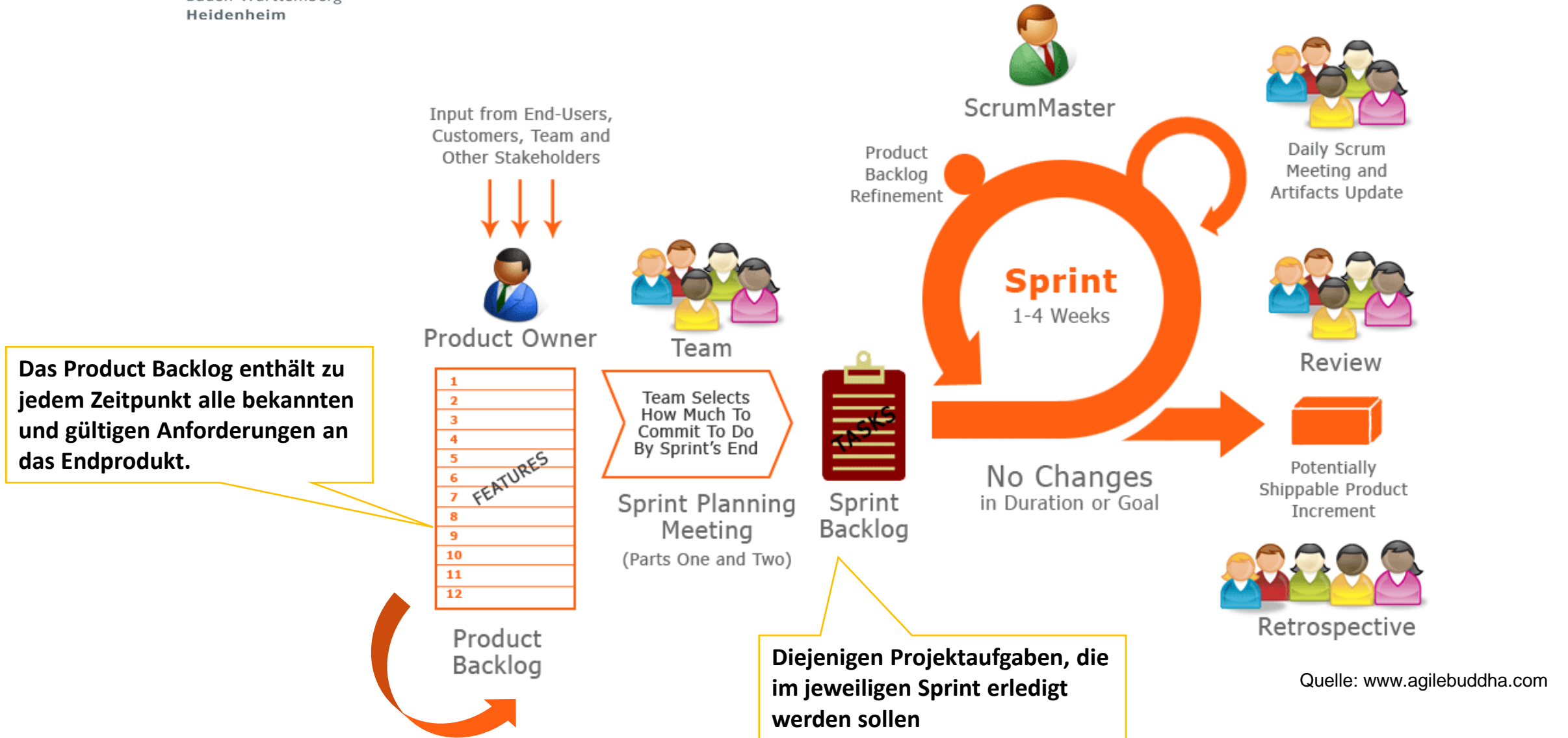
Die Anforderungen aus Anwender-Sicht werden meist als **User Stories** bezeichnet.

- Am Ende eines **jeden Sprints steht bei Scrum die Lieferung einer fertigen** (Software)-Funktionalität (**das Produkt-Inkrement**).
- Die neu entwickelte Funktionalität sollte in einem Zustand sein, dass sie **an den Kunden ausgeliefert werden kann** (potentially shippable code oder usable software).

Aufgrund seiner engen **Einbeziehung in das Projekt kann der Kunde unmittelbar Feedback** geben. So wird genau das entwickelt, was er wirklich benötigt.

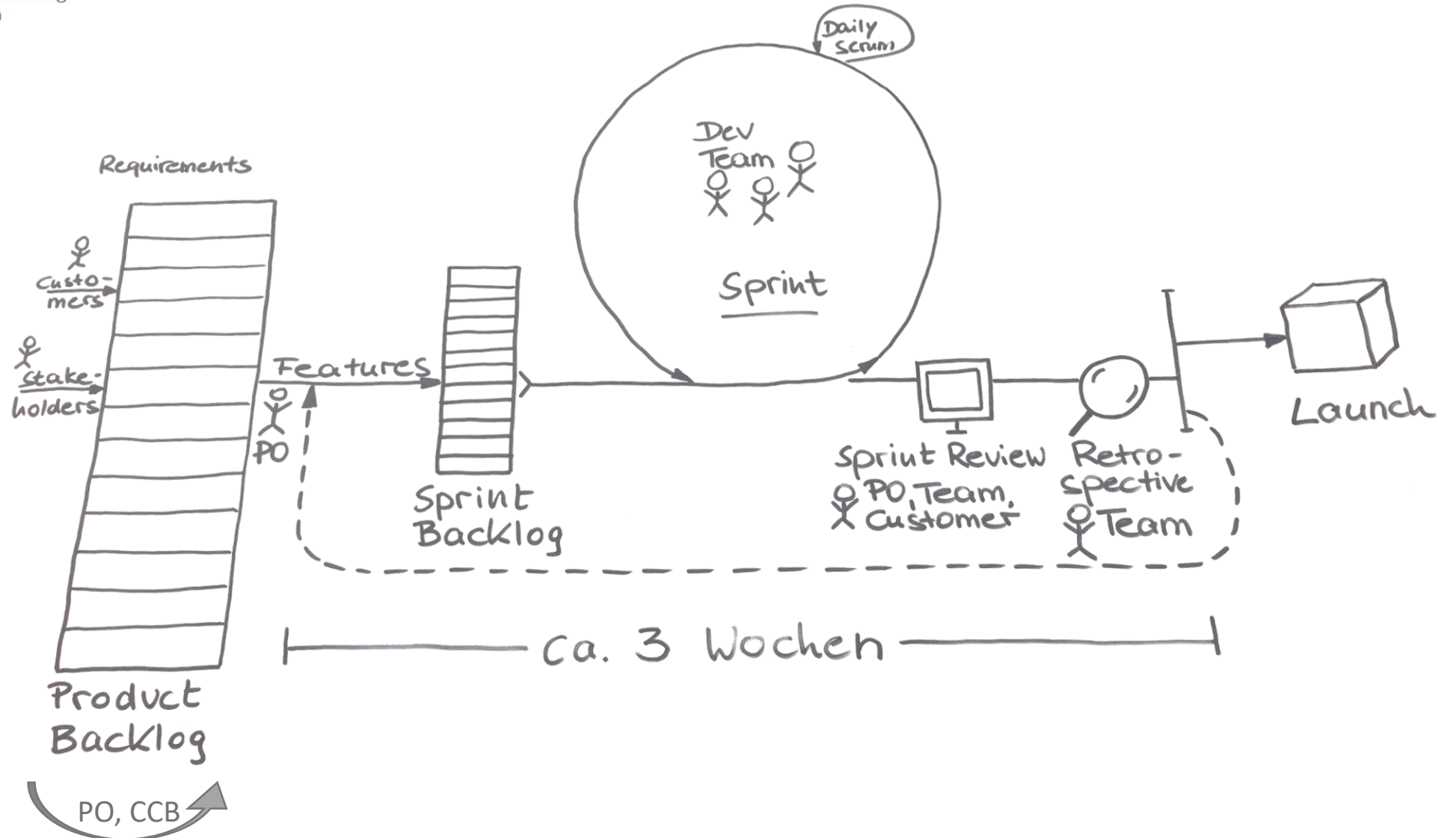
- In jedem Zyklus werden die am **höchsten priorisierten Anforderungen umgesetzt und damit die jeweils höchste Wertschöpfung** für den Kunden erzielt.
- Auf Änderungswünsche kann flexibel und kurzfristig reagiert werden.

# SCRUM - Framework im Überblick





# Scrum-Phasen



# SCRUM – Framework: Elemente im Überblick

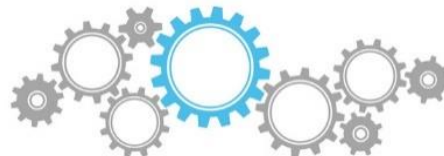
## ■ Tools + Repositories („Artefakte“)

- Product Backlog
  - Kundenwünsche
  - Backlog der Field issues und technischen Schulden
  - Tech Request (technisch notwendige Änderungen)
- Sprint Backlog
- Inkrement
- Definition of Ready/ Done



## ■ Organisation/Prozess/ Best practices

- Sprint Planung mit Slicing und Planning Poker
- Daily Scrum
- Sprint Review
- Sprint Retrospective



## ■ Rollen (der Personen)

- Kunde/Auftraggeber
- Product Owner
- Scrum Master
- Entwicklungsteams + CoCs
- CCB



# SCRUM – Bugbewertung, Einordnung



$$\text{Bug severity} = \text{Customer impact} * \text{Probability}$$

- **Customer impact:**

- 1= no impact
- 2 = minor impact: usability issues, komplizierte Bedienung, Typos, temporäre Performanceschwächen,...
- 3= medium impact: Fehlverhalten/ Unbenutzbarkeit neuer Features, schlechte Bedienbarkeit
- 4: major impact: System crash, kurze (<1 min.) Nichtverfügbarkeit, Anmeldeprobleme, kurzzeitige Verbindungsverluste,....
- 5= serious impact: Downtime, Verlust von Leben oder Kontrolle, Datenverlust, Sicherheitslücken

- **Probability:**

- 1= nicht reproduzierbar
- 2 = sehr geringe Wahrscheinlichkeit (<10% der Systeme)
- 3= mittlere Wahrscheinlichkeit (50% der Systeme)
- 4= hohe Wahrscheinlichkeit (>80% der Systeme)
- 5= immer

**Severity = 25 (5x5): ShowStopper – sofortiger Fix notwendig, muss in allen freigegebenen Versionen gefixt werden**

**Severity = 16/ 20: Severity höher als alle neuen Features, Fix für alle betroffenen freigegebenen Versionen**

-----  
**Severity = 12/10: müssen vor dem Start des Systemtest gefixt sein**

**Severity <= 9: eventuell nicht beheben**



## Product Owner (PO)

- Der Product Owner ist für die **Eigenschaften und den wirtschaftlichen Erfolg des Produkts** verantwortlich, vertritt den Kunden gegenüber dem Team.
- Er **definiert und priorisiert die zu entwickelnden Produkteigenschaften** und nimmt diese am Ende eines Sprints ab.

## Scrum Master

- Der Scrum Master **fungiert als Moderator** und Dienstleister („servant lead“).
- Er **unterstützt das Entwicklungsteam bei der Einhaltung der Regeln**, er schafft geeignete Rahmenbedingungen, **organisiert Ressourcen, räumt Hindernisse aus dem Weg**, moderiert bei Konflikten, **organisiert die Kommunikation mit der „Außenwelt“**.



## Entwicklungsteam

- Das Entwicklungsteam **organisiert sich selbst und verantwortet die Lieferung** der Produktfunktionalitäten in der vom Product Owner gewünschten Reihenfolge gemäß den **vereinbarten Qualitätsstandards**.
- Kompetenzen/ Verantwortung/ Rollen:
  - Produktdesigner (arbeitet evtl. in mehreren Teams mit)
  - Architekt (arbeitet evtl. in mehreren Teams mit)
  - Entwicklung
  - Test designer/ Tester
  - Tech Writer (Customer documentation) (arbeitet evtl. in mehreren Teams mit)
  - Supporter (arbeitet evtl. in mehreren Teams mit)



## Architekt/ CoC

- Centers of Competence und/ oder Architekten verantworten einzelne kritische Architekturbereiche oder die Systemarchitektur
- Sie werden an Konzeptreviews und Architekturentscheidungen beteiligt oder stoßen diese an

## Change Control Board

- Mitglieder: PO, Support, Tester, Architect, CoC...
- Aufgabe: Pflege des Backlogs mit besonderem Fokus auf Customer issues (Grooming/ Refinement)
- Analyse, Priorisierung und Einordnung der aus dem Feld gemeldeten Bugs ins Produkt backlog (ins Sprint backlog nur bei Show stoppern)
- Meetings: täglich (Systemtest) ... 1x/ Woche (Neuprodukte)
- Ziel: „no faults forward“!

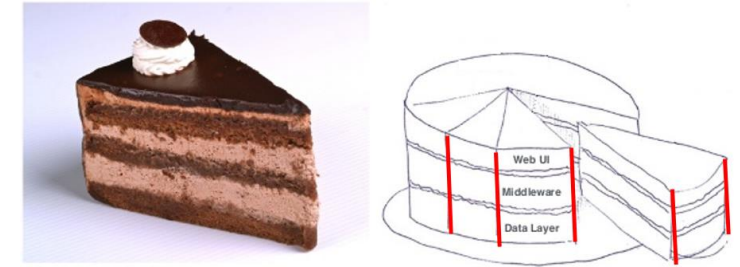




## Team-Setup-Varianten:

### a) Featureteam/ Scrum Team:

- Team bearbeitet eine komplette User story, quer über die Architektur des gesamten Produkts
- Architekt oder CoC oft beratend/ reviewend tätig

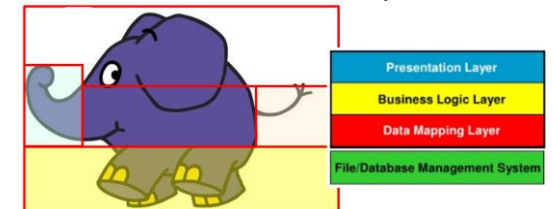




## Team-Setup-Varianten:

### b) Fokusteam/ Center of Competence (CoC):

- Team verantwortet einen bestimmten Architekturbereich eines Produkts: einen oder mehrere Clients, oder Backend-Komponenten (Datenbankanbindung, Business-Logik,...), oder remote Kommunikation, oder Konfiguration, oder...
- klassische Teamaufteilung in Wasserfallprojekten
- Arbeit an einem Features wird auf die Teams der Architekturschichten verteilt: langsamstes Team bestimmt den möglichen Lieferzeitpunkt des Features
- meist gleichzeitige Arbeit an mehreren Features für gleichmässige Auslastung aller Teams
- in agilen Projekten: CoC oft beratend tätig, oder verpflichtend bei Reviews einzubinden



# Scrum-Prozess

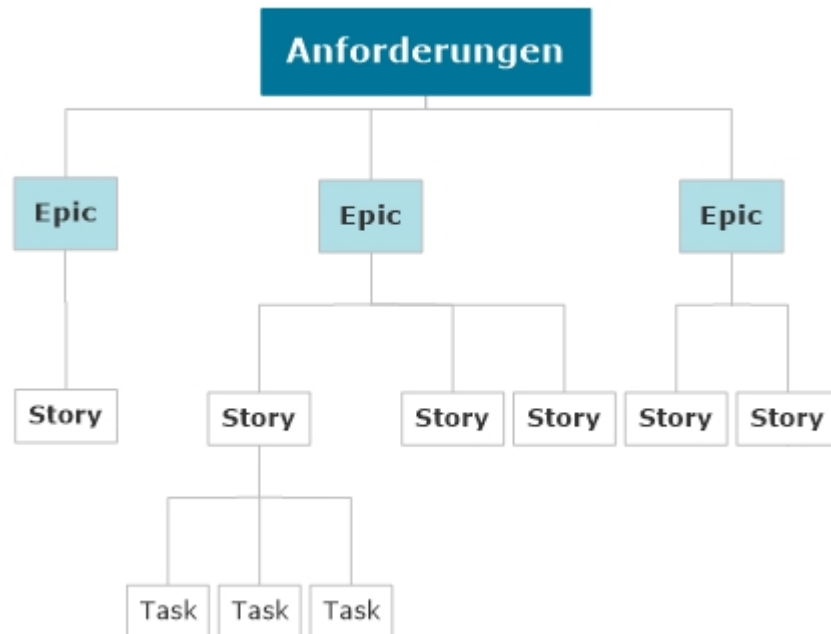
## 0) Projektinitiierung

- Teamzusammenstellung mit entspr. Skills, auch in QA-Methoden (z.B. *ISTQB®-Ausbildung zum Certified Agile Tester*) und Fakultäten (nicht unbedingt **dedizierter Tester**, aber diese Rolle)
- **Product Owner** mit fachlicher Kompetenz und Handlungsspielraum
- **Scrum Master** als Roadblock-/Barrieren-Entferner
- Definition of Ready (DoR) und Definition of Done (DOD) initial festlegen
- Iteration Zero (Sprint 0): initiale/ prototypische Architektur, Testumgebung (Virtualisierung, Container), Buildumgebung, Deployment auf Testsysteme etc.
- Bugmanagement aufsetzen

# Scrum-Prozess

## a) Product Backlog grooming

Product backlog items: **User stories**



### Epic

Epic ist eine User Story auf höchster Abstraktionsstufe. Für einen Sprint ist sie zu groß. Der Product Owner zerlegt sie deshalb in mehrere kleinere User Stories.

### Story

Die Story ist eine User Story und Teil einer Epic. Eine Story enthält die Anforderungen, geschrieben als kleine handhabbare Texte. Auf dieser Basis schätzt das Team den Realisierungsaufwand ein.

### Task

Entwickler und Tester zerlegen eine Story in einzelne Aufgaben (Tasks). Wenn eine Aufgabe eine vereinbarte Zeitspanne überschreitet, kann sie in zusätzliche Aufgaben aufgeteilt werden. Eine Story ist dann erledigt, wenn alle Aufgaben der Story erledigt sind.

# Grooming/ Refinement für PBIs (PO, Team) (Product backlog items)

Alle PBIs haben einen Lebenszyklus. Wichtige Meilensteine des Groomings:

**Requirement-/Epic-Ebene:** „Ready“  
Feature-/User story-Ebene: „Done“

**Definition of „Ready“ (DoR)/** Definition of „Done“ (DoD):

- Zielt auf gemeinsames Verständnis ab, wann eine Aufgabe (Epic) gestartet werden kann.
- Entwicklungsteamspezifisch
- Umfasst mit zunehmender Erfahrung des Entwicklungsteams mehr Aktivitäten -> Maß für die Reife eines Entwicklungsteams

„Die Vereinbarung des Teams, was für das Erreichen dieser beiden Zustände getan sein muss, bestimmt die Qualität des Artefakts!“

Chris Rupp (Sophisten)

# Definition of „Ready“ (für PBIs)

- Funktionale Requirements sind verhandelt
- Nichtfunktionale Requirements sind erfragt
- Erfolgskriterien sind definiert

**Analysiert**

- Lösungsoptionen sind mit Pros & Cons erstellt

**Beschrieben**

- Alle Stakeholder wurden einbezogen
- Lösungsoptionen wurden diskutiert

**Reviewed**

- Die gewählte Lösung wurde in lieferbare Features (User stories) unterteilt

**Verfeinert**

# Scrum-Prozess

## b) Slicing der Epics in User stories

Eine User Story ist wie folgt aufgebaut:

Als [Nutzer] möchte ich [Funktion], damit / um / weil [Wert].

Das heißt, eine User Story beantwortet die Fragen: WER möchte WAS und WARUM.

Als User stories formulierte Anforderungen sollen als Produktinkremente einen sichtbaren, neuen Kundennutzen bringen!

# Scrum-Prozess

## b) Slicing der Epics in User stories

### Qualitätskriterium beim Slicing: „INVEST“

- **Independent:**  
Die User story ist **unabhängig** von jedem anderen Artefakt (3rd party library, andere User Stories), das am Beginn des Sprints noch nicht verfügbar ist.
- **Negotiatable:**  
Das Team und der PO/ PM sind sich einig, daß die Story eine gute Basis für **weitere Diskussionen** ist.
- **Valuable:**  
Das Team und der PO/ PM sind sich einig, daß die Story einen Wert zum Systemnutzen beiträgt und **geliefert** werden könnte.
- **Estimable:**  
Das Team hat genug Input für eine **Aufwandsabschätzung**.
- **Small:**  
Das Feature paßt in **einen Sprint** (bis und einschließlich DoD).
- **Testable:**  
Das Team kann **testen**, ob die Story erfolgreich umgesetzt wurde.



# Scrum-Prozess

## b) Slicing der Epics in User stories

WWW.AGILE42.COM

### Story Splitting Cheat Sheet

#### The INVEST Model

Stories should be:

**Independent, Negotiable, Valuable, Estimable, Small and Testable.**

#### Patterns for Splitting Stories

##### Workflow Steps

As a content manager, I can publish a news story to the corporate website.

...I can publish a news story directly to the corporate website.

...I can publish a news story with editor review.

...I can publish a news story with legal review.

##### Business Rule Variations

As a user, I can search for flights with flexible dates.

...as "n days between x and y."

...as "a weekend in December."

...as "± n days of x and y."

##### Major Effort

As a user, I can pay for my flight with VISA, MasterCard, Diners Club, or American Express.

...I can pay with one credit card type (of VISA, MC, DC, AMEX).

...I can pay with all four credit card types (VISA, MC, DC, AMEX).

##### Simple/Complex

As a user, I can search for flights between two destinations.

...specifying a max number of stops.

...including nearby airports.

...using flexible dates.

...etc.

#### Variations in Data

As a content manager, I can create news stories.

...in English.

...in Japanese.

...in Arabic.

...etc.

#### Data Entry Methods

As a user, I can search for flights between two destinations.

...using simple date input.

...with a fancy calendar UI.

#### Defer Performance

As a user, I can search for flights between two destinations.

...(slow - just get it done, show a "searching" animation).

...(in under 5 seconds).

#### Operations (e.g. CRUD) (Create, Read, Update, Delete)

As a user, I can manage my account.

...I can sign up for an account.

...I can edit my account settings.

...I can cancel my account.

#### Break Out a Spike

As a user, I can pay by credit card.

Investigate credit card processing.

Implement credit card processing  
(as one or more stories).



# HOW TO SPLIT A USER STORY

## 1 PREPARE THE INPUT STORY



\* INVEST - Stories should be:  
Independent  
Negotiable  
Valuable  
Estimable  
Small  
Testable

## WORKFLOW STEPS

Can you split the story so you do the beginning and end of the workflow first and enhance it with stories from the middle of the workflow?

Can you take a thin slice through the workflow first and enhance it with more stories later?

## DEFER PERFORMANCE

Could you split the story to just make it work first and then enhance it to satisfy the non-functional requirement?

Does the story get much of its complexity from satisfying non-functional requirements like performance?

## SIMPLE/COMPLEX

Could you split the story to do that simple core first and enhance it with later stories?

Does the story have a simple core that provides most of the value and/or learning?

Could you group the later stories and defer the decision about which story comes first?

## MAJOR EFFORT

When you apply the obvious split, is whichever story you do first the most difficult?

Can you split the story to handle data from one interface first and enhance with the others later?

## INTERFACE VARIATIONS

Does the story get the same kind of data via multiple interfaces?

Is there a simple version you could do first?

## OPERATIONS

Can you split the operations into separate stories?

Does the story include multiple operations? (e.g. is it about "managing" or "configuring" something?)

## BUSINESS RULE VARIATIONS

Can you split the story so you do a subset of the rules first and enhance with additional rules later?

Does the story have a variety of business rules? (e.g. is there a domain term in the story like "flexible dates" that suggests several variations?)

## VARIATIONS IN DATA

Can you split the story to process one kind of data first and enhance with the other kinds later?

Does the story do the same thing to different kinds of data?

## BREAK OUT A SPIKE

Are you still baffled about how to split the story?

Can you find a small piece you understand well enough to start?

Write that story first, build it, and start again at the top of this process.

Can you define the 1-3 questions most holding you back?

Write a spike with those questions, do the minimum to answer them, and start again at the top of this process.

Take a break and try again.

## 3 EVALUATE THE SPLIT

Are the new stories roughly equal in size?

Is each story about 1/4 to 1/2 of your velocity?

Do each of the stories satisfy INVEST?

Are there stories you can deprioritize or delete?

Is there an obvious story to start with that gets you early value, learning, risk mitigation, etc.?

You're done, though you could try another pattern to see if it works better.

**NO** → Try another pattern on the original story or the larger post-split stories.

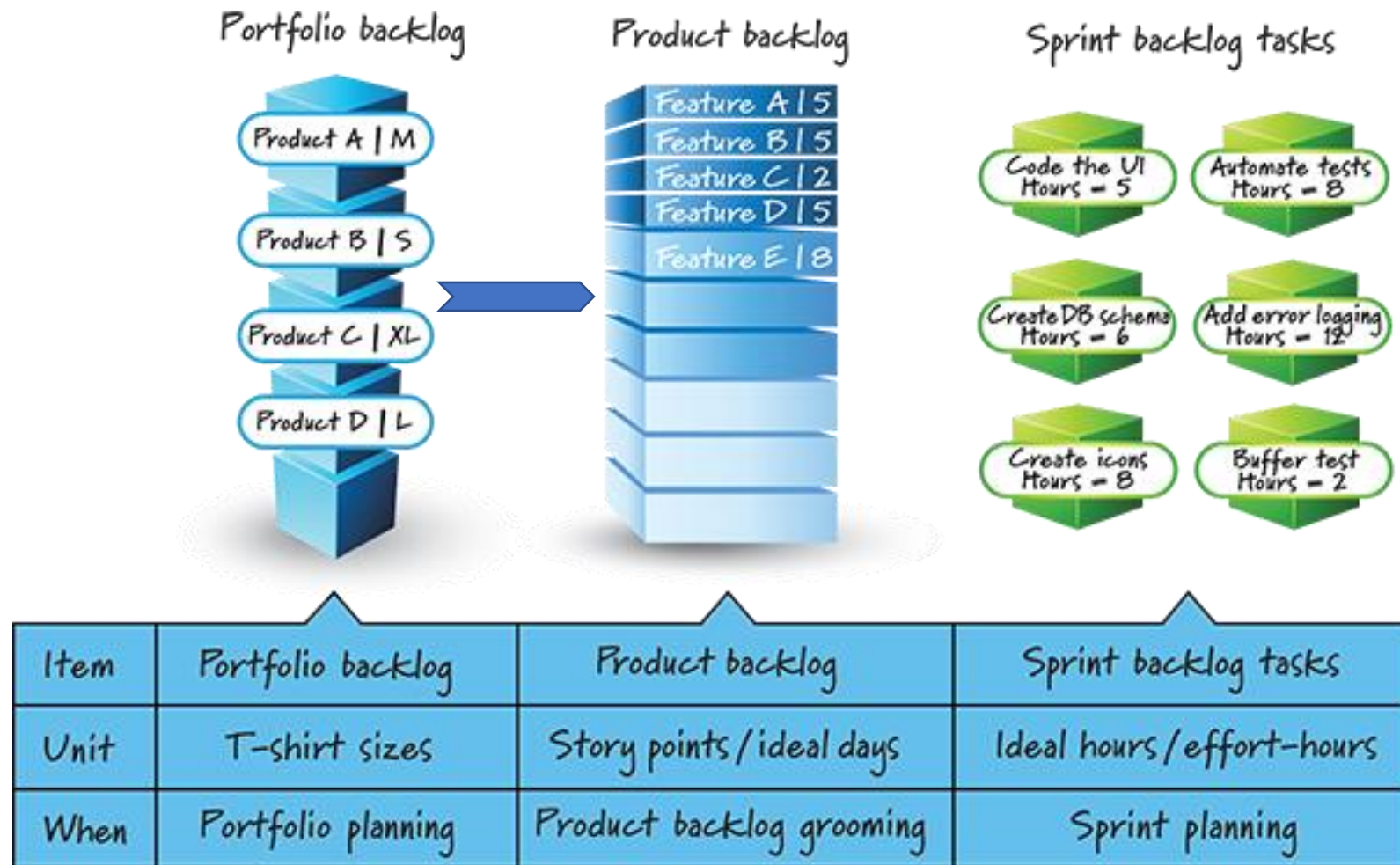
Try another pattern.

Try another pattern. You probably have waste in each of your stories.

Try another pattern to see if you can get this.

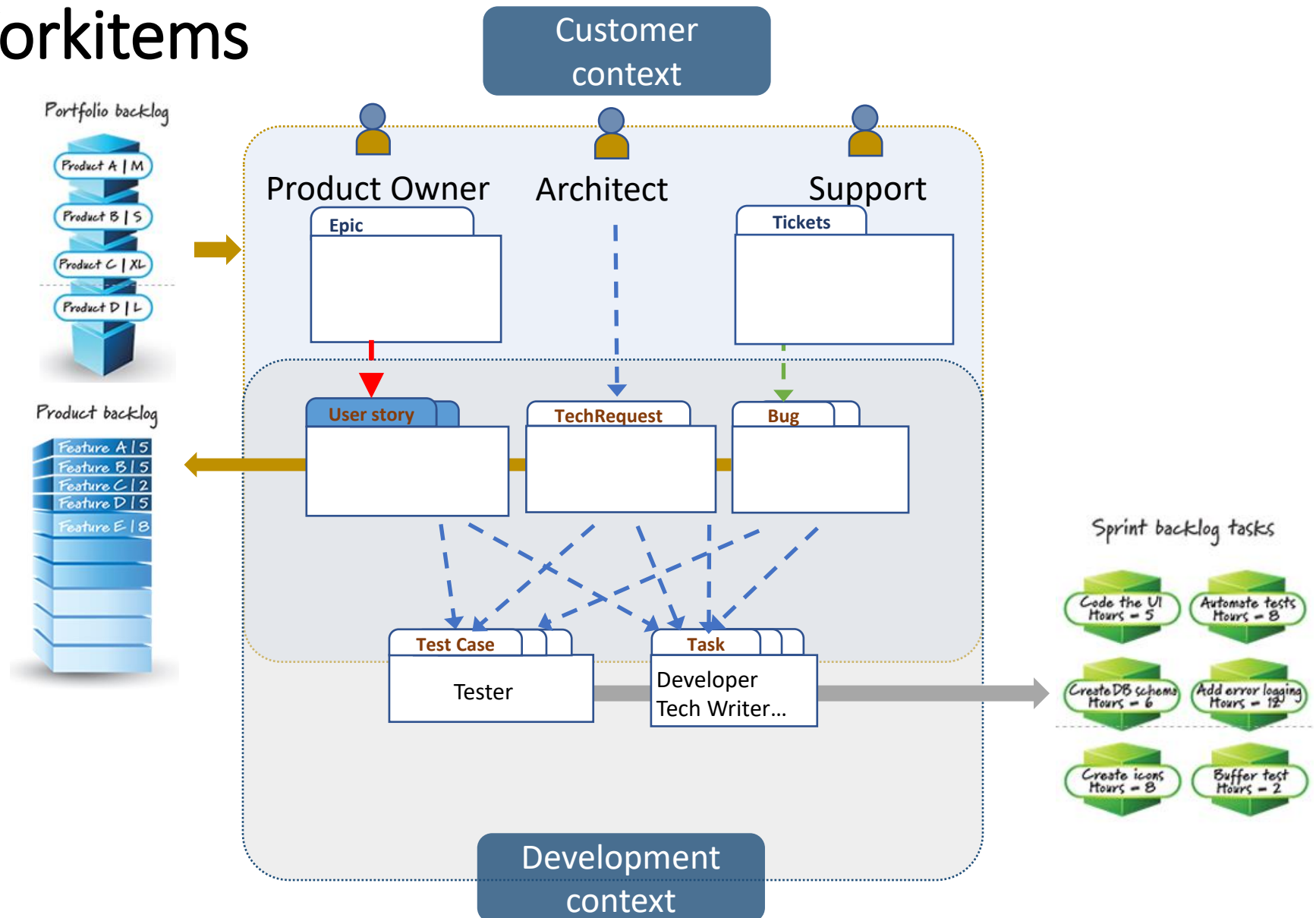
# Scrum-Prozess

## b) Slicing der Epics in User stories/ Features



2012, Kenneth S. Rubin and Innolution, LLC.

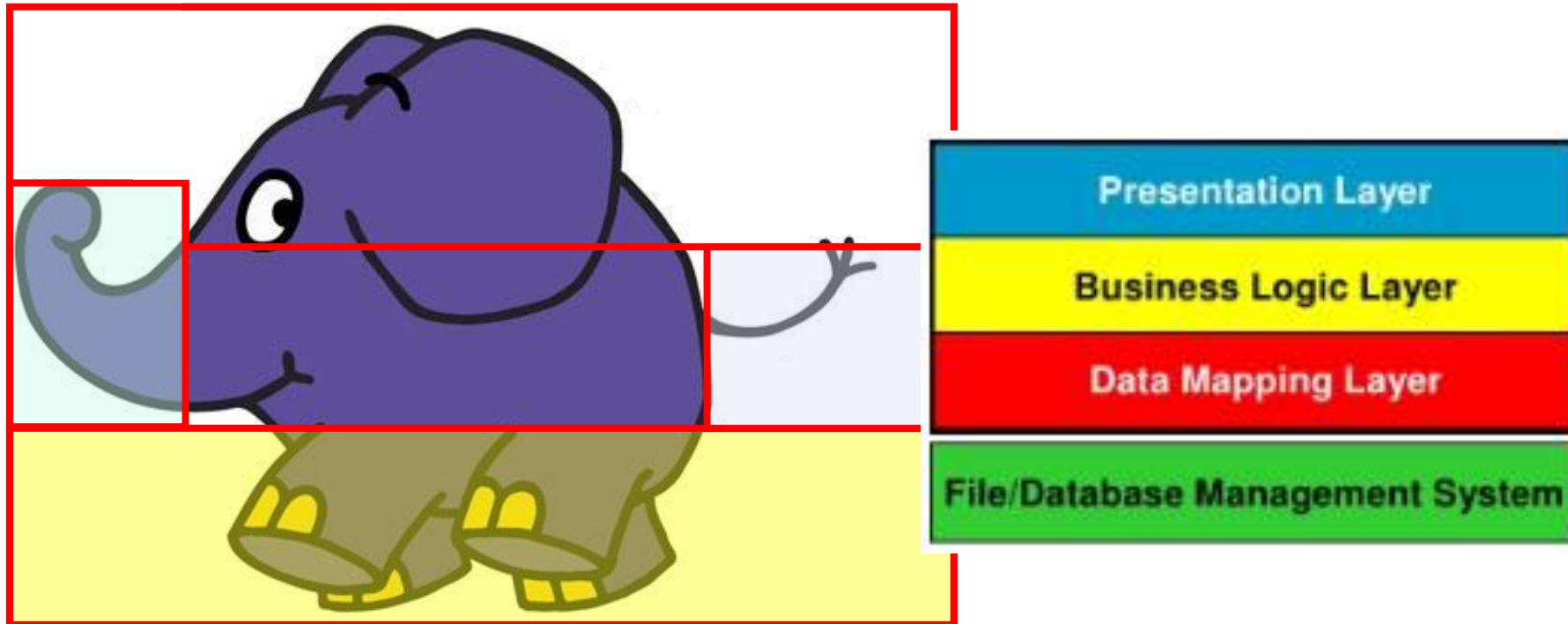
# Workitems





# Elefantencarpaccio

Horizontales Schneiden mit dem Fokus auf Architekturkomponenten?





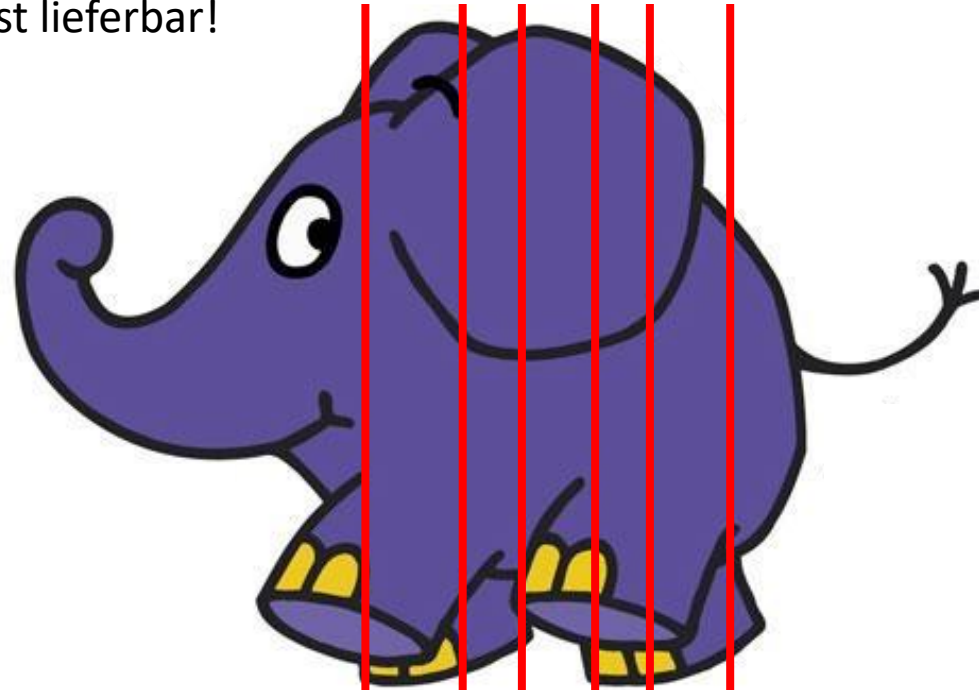


# Vertikales Schneiden

Oder vertikal schneiden?

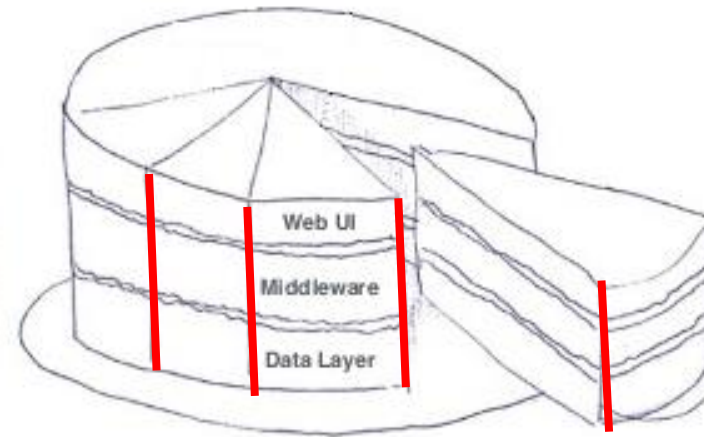
Feature-Team-Cut (fokussiert auf vertikale Deliverables):

- Lösung als Ganzes verstehen und durchdringen
- Scheibchenweise funktional umsetzen
- Ziel: jedes Feature ist lieferbar!



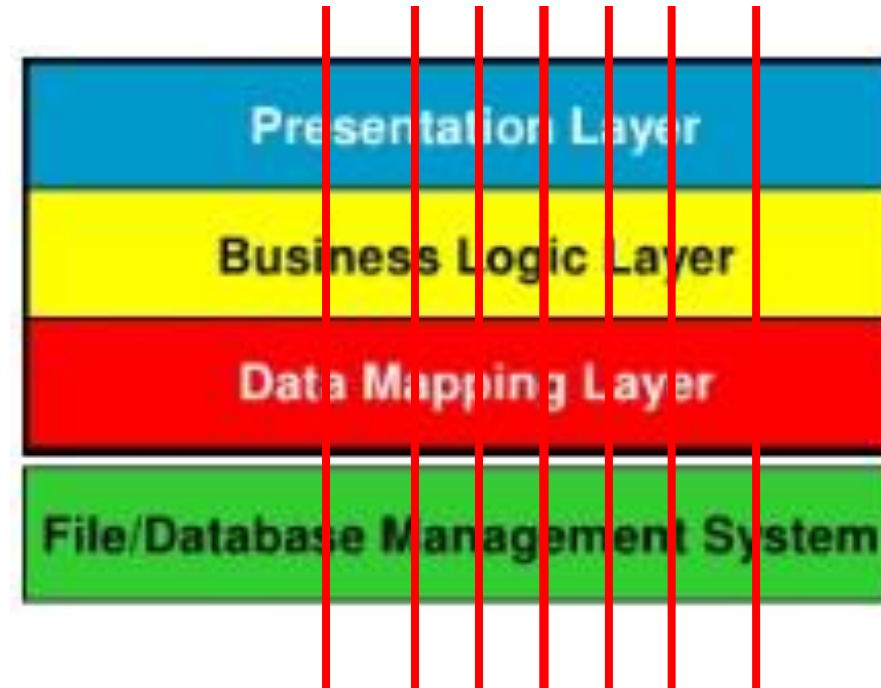


# Elefantenfreundlicher...



# Architektur-Elefanten

Software-Architektur:



Voraussetzung für erfolgreiches Elefantencarpaccio: Interdisziplinäre Teams!

- Die Teams brauchen die Kompetenz, am Gesamtsystem Änderungen vorzunehmen
- Tester sind Bestandteil des Teams und arbeiten schritthaltend
- Evtl. zusätzliches Architekturteam als Berater-/Prototyper/ CoCs
- Bildung von DevOps für Kundeninfrastruktur-abhängige IT-Systeme



# Gegenüberstellung

Bedeutung für:	Horizontales Schneiden (Architekturbezug)	Vertikales Schneiden (Featurebezug)
Teamstruktur	Fokussierte, architekturbezogene Teilkompetenz ausreichend	Interdisziplinäre Team-Zusammenstellung nötig
Kommunikation	Alle Schnittstellenabsprachen mit anderen Teams	Innerhalb des Teams für alle Schnittstellen des Features
Architektur-Governance	Spezialisierte Teams mit dedizierter Verantwortung	Verteilt; evtl. zusätzl. Architekturteam oder COCs
Fertigstellung	Fertig, wenn letztes Team fertig ist	Vom Team vollständig bearbeitbar (bis „Done“)
Abschätzung	Summe aller Teams + kritischer Pfad + Reibung/ Rework an Interfaces	Ein Team schätzt
Ergebnis-verantwortung	Verteilt, schwierige Stabilisierung der Qualität	Team selbst, inkl. Qualität

# Scrum-Prozess

## c) Sprint planning

Nächster Schritt: Abschätzung mit Planning poker für User stories



# Scrum-Prozess

## c) Sprint planning

**Ziel: Definition of Done (DoD) muss für eine User Story in einem Sprint erreichbar sein**



# Scrum-Prozess: Sprint planning: DoD (real world example)

## Development

- All dedicated Tasks are closed
- SC updated, reviewed for integrity and completeness
- New CTQ's defined
- Build without Errors\Warnings
- Open source scan done
- Potential patent infringements considered
- Use of outside patents/licenses approved
- Forward integration done
- Development done according to coding/checkin guidelines
- All checkins reviewed acc. to code review guidelines
- No Code refactoring pending
- Unit tests implemented
- Feature integrated into Setup
- Critical feature bugs  $\geq 12$  fixed
- Uncritical Bugs  $< 12$  closed as accepted
- Planned UX activities done
- Acceptance Test\Demo done with PO at sprint review meeting

## Test

- "How to test" answered by development
- Test cases added to test case repository
- Test cases reviewed again
- Test environment available  
(incl. required tools\triggers)
- Regression test done
- Integration test done
- New CTQ's tested

## Product documentation

- Online manuals updated
- Written product documentation updated

# Scrum-Prozess

## c) Sprintplanung

- Quality gate
- Basierend auf zyklischen Grooming des Product backlogs
- Wichtung von TechRequests (abtragen technischer Schuld und Architekturarbeiten) gegenüber neuer Funktionalität vor jedem Sprint, ebenso Field issue-Behebung, evtl. unter Einfügen von Spikes
- Funktionierende SW über Dokumentation: Testfälle sind die beste Dokumentation des Systemverhaltens
- Planung von Simulatoren, Stubs,...
- Schneiden von User stories in Tasks (Entwicklungsteam), Aufwand/ Scope: 1-2 PT



## Sprint execution

# Scrum-Prozess

## d) Sprint execution

### Daily stand ups/ Daily scrum (Entwicklungsteam-intern):

- max. 15 min. (im Stehen)
- Ziel: mit Blick aufs Task-Board Arbeiten aufeinander abzustimmen, einen Plan für die nächsten 24 Stunden zu haben
- Erreichen des Sprintziels wird wahrscheinlicher
- fester täglicher Zeitpunkt und Ort
- jeder Teilnehmer beantwortet 3 Fragen:
  - Was habe ich seit dem letzten Daily Standup erreicht?
  - Was werde ich heute erreichen?
  - Erwarte ich Hindernisse und kann das Team mir dabei helfen?

# Scrum-Prozess

## d) Sprint execution

Warum sind Daily Standup-Meetings wichtig?

- Sie verbessern die Kommunikation.
- Das Daily Standup-Meeting sorgt dafür, dass andere Treffen überflüssig werden.
- Hindernisse werden identifiziert.
- Eine schnelle Entscheidungsfindung wird stimuliert.
- Die Kenntnisse des Scrum-Teams werden erweitert.
- Fortschritte in Bezug auf das Sprintziel werden transparent gemacht.

Die Teammitglieder können Nachlässigkeiten daher nur schwer verbergen.

An diese Transparenz müssen sich einige erst gewöhnen, für andere hingegen ist sie eine Erleichterung.

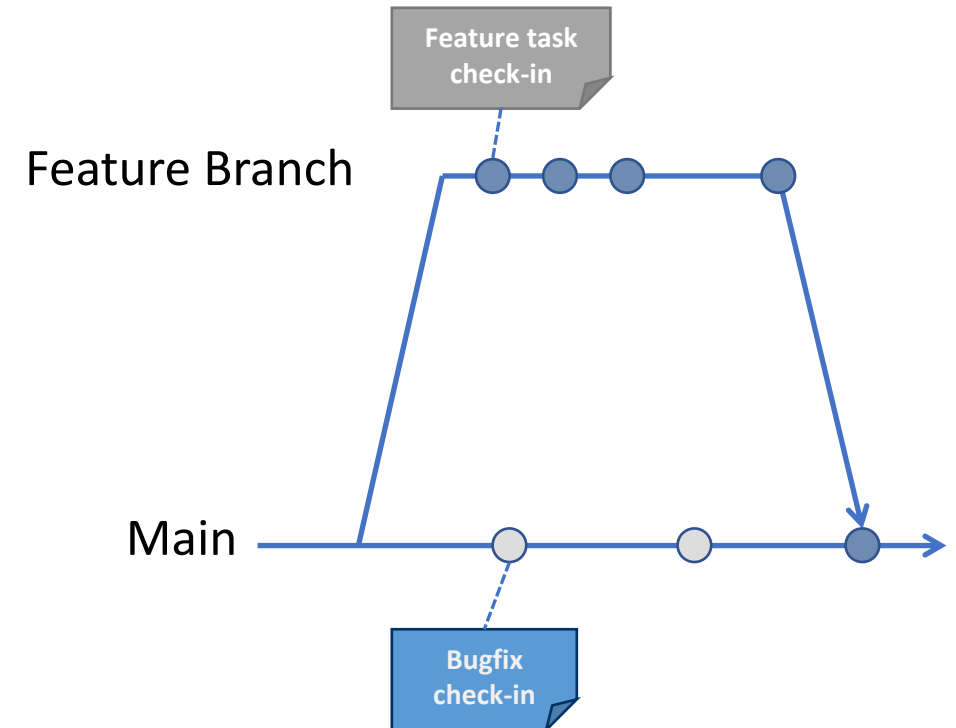
## d) Sprint execution: weitere Best practices

- Pair Programming
- Ein reifes Team gibt sich ab der Designphase Architekturregeln, Programmier- und Designrichtlinien (Entwurfsmuster, Styleguides - automatisch prüfbar)
- Test-Driven/ Test-first Development mit automat. Ausführung von Regressions-Tests
- Peer Code reviews
- Zeitnaher Test der integrierten neuen Funktionalität innerhalb des Sprints (pro Feature: forward integration -> build + Integrationstest auf Feature branch -> schnelles Feedback ans Team zur Produktqualität)
- Continuous build + integration (autom. Deployment auf Testsysteme)
- Hoher Grad an Testautomatisierung notwendig (automatische Regressionstests der Funktionalität und der CTQs)

# Scrum-Prozess: Sprint execution

## Feature branches

- Feature Check-ins nur auf den Feature Branch
- Explizite Entscheidung: wann zurückmergen?
- Bug fixes noch auf Main, wenn:
  - Unabhängig von der Feature-Entwicklung
  - Niederes Risiko angenommen





# Scrum-Prozess: Sprint execution

## Feature branches

### Erwartete Vorteile:

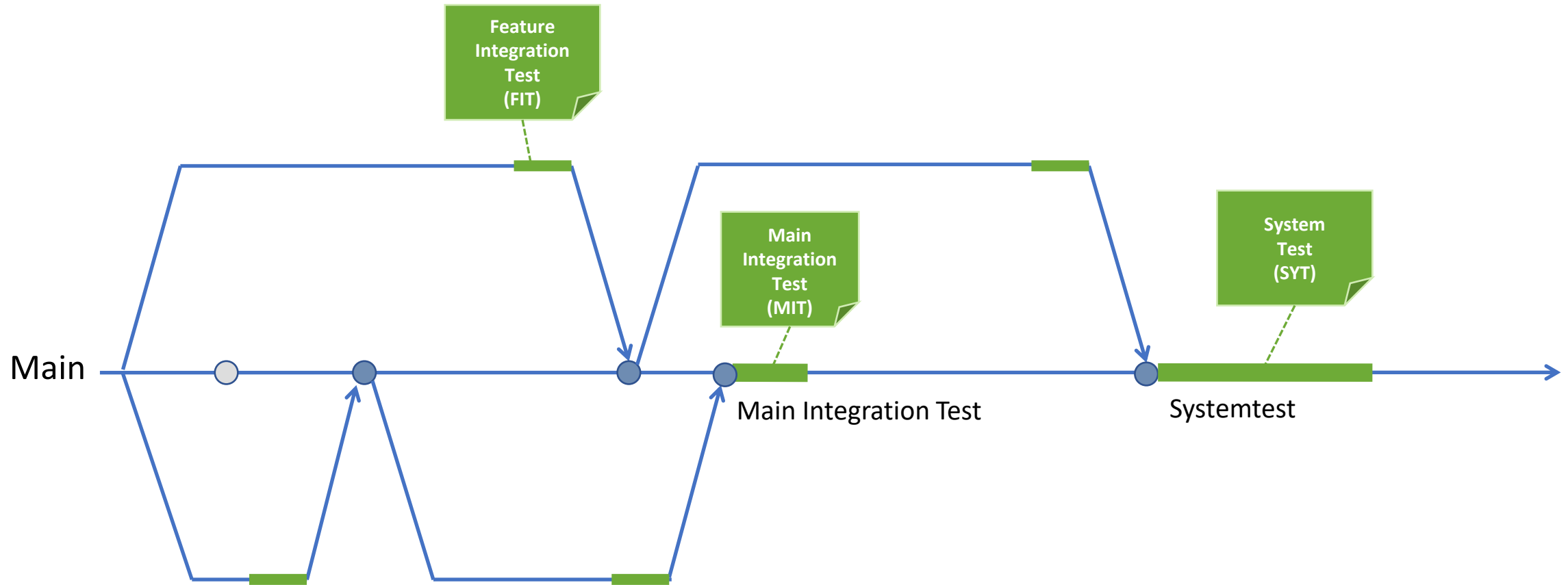
- Main Branch bleibt gesund
- Kürzere Releasezyklen (mit weniger Features!) möglich
- Releaseplan besser zu managen.

### Mögliche Probleme:

- Zusätzlicher Merging-Aufwand (forward-/backward-Integration)
- komplexere Infrastruktur



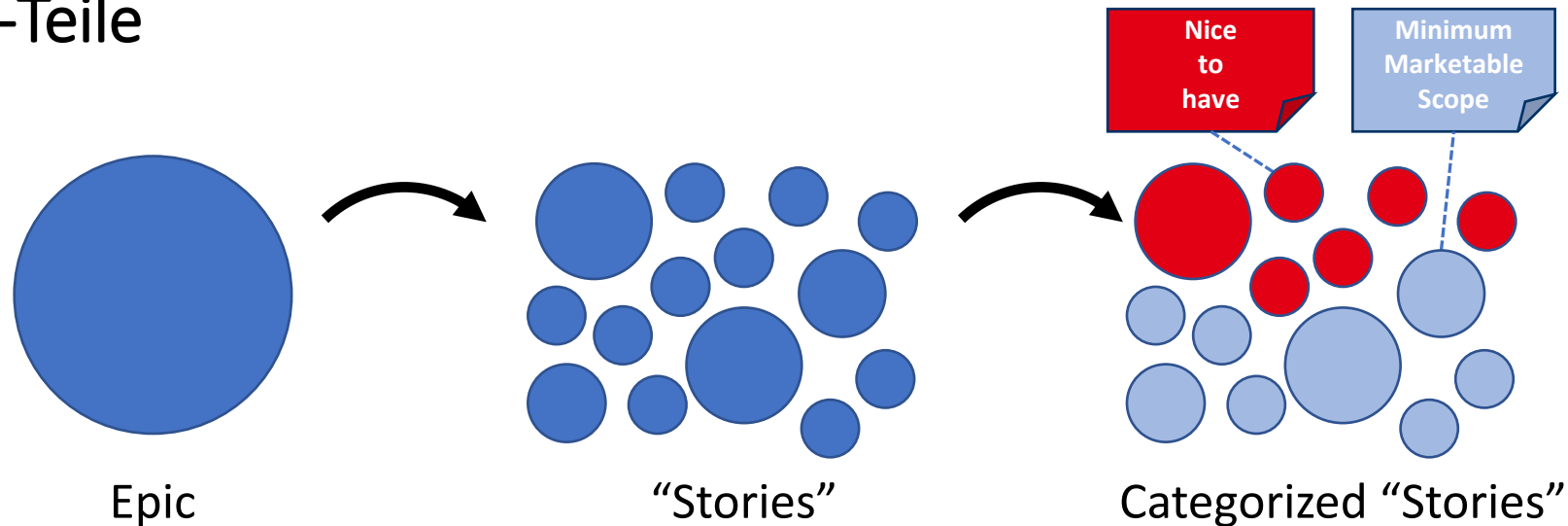
# Scrum-Prozess: Feature branches Workflow und Test II



# Scrum-Prozess: Feature branches

## Was lohnt sich, zurückzumergen?

### Epic-Teile



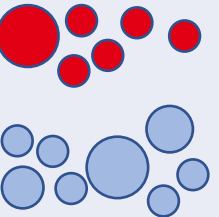

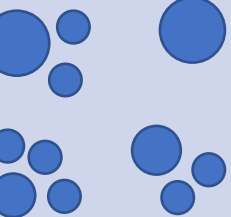





### Mögliche Merge-back-Einheiten:

- Ganzes Epic (1 große Portion)
- Minimum Marketable Scope, dann "Nice to have" (2 große Portionen)
- n Stories in beliebiger Anzahl (n kleine Portionen)
- Eine einzelne Story (n sehr kleine Portionen)

# Scrum-Prozess: Feature branches

## Feature branch merging

	Teilgröße		Potentielle Probleme
	Ganzes Epic		<ul style="list-style-type: none"> <li>- Verursacht späte Integration</li> <li>- Spätes Feedback</li> <li>- Hoher Merge-Aufwand</li> <li>- Späte Tests auf dem Main branch</li> </ul>
	1. MMS 2. "Nice to have"		<ul style="list-style-type: none"> <li>- MMS muss vorab definiert sein</li> <li>- Relativ späte Integration/ Test auf Main</li> <li>- Relativ hoher Merge-Aufwand</li> </ul>
	Stories in beliebiger Anzahl		<ul style="list-style-type: none"> <li>- In Main sichtbar, obwohl noch nicht vermarktbare, reduzierte Flexibilität (Release Feature A, Feature B noch nicht, aber Teile von B sind schon enthalten)</li> </ul>
	Einzelne Story		<ul style="list-style-type: none"> <li>- Hochfrequente Integrationen, die viel Overhead verursachen</li> </ul>

Mögliche Abfederung: Sichtbare Teile noch nicht vermarktbarer Features im UI verstecken

# Scrum-Prozess:

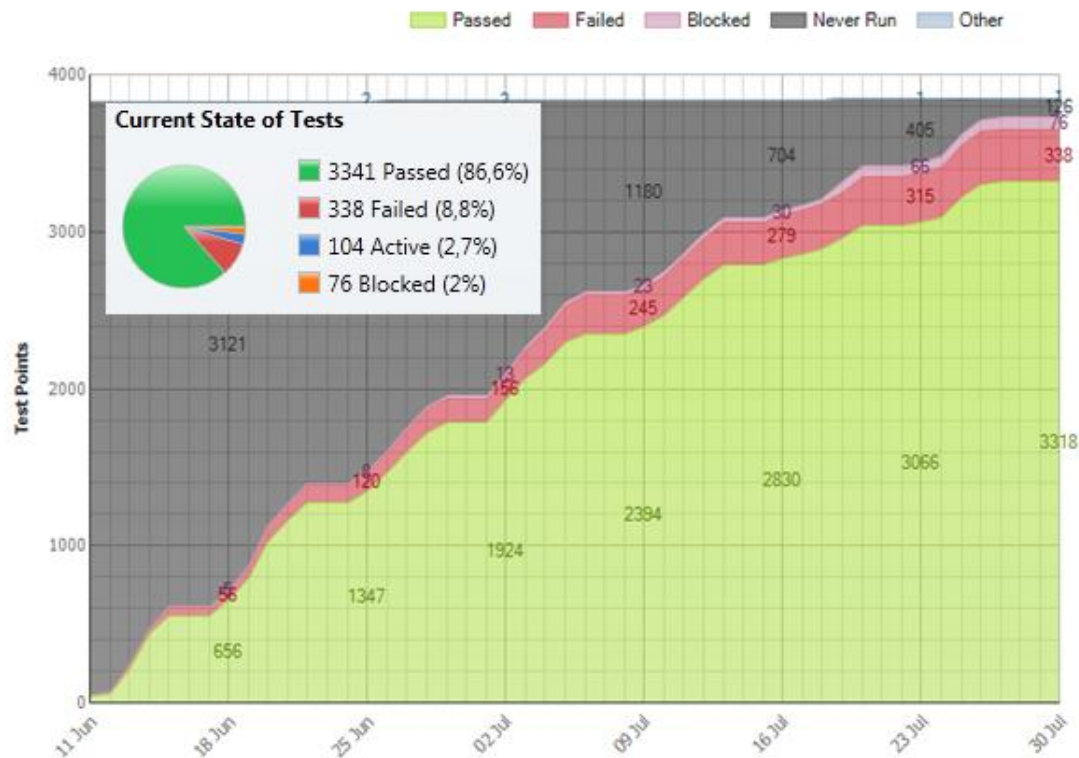
## e) Sprint review

- Quality Gate unter Kunden-/ Stakeholderbeteiligung
- Tester des Teams demonstrieren neue Features, haben das größte Produkt-KnowHow (einsatzbar bei Schulungen, Support...)
- Akzeptanzkriterien der DoD werden bestätigt – Konzentration nicht nur auf die für den Anwender sichtbare Funktionalität!
- Verantwortlich für Abnahme des Sprintergebnisses: PO + Stakeholder und QA
- Nächster Sprint erst nach Erledigung aller Hausaufgaben – und auf Basis der Kundenzufriedenheit mit dem Zwischenergebnis

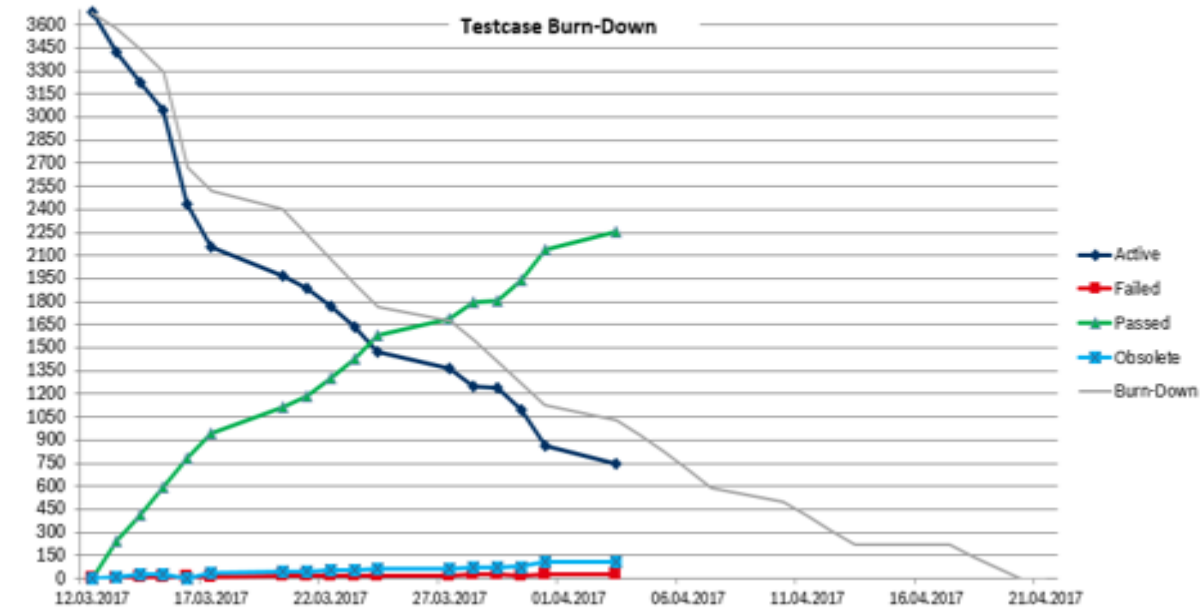
# Scrum-Prozess: e) Sprint review - Reporting

## Testfortschritt:

Testcases executed



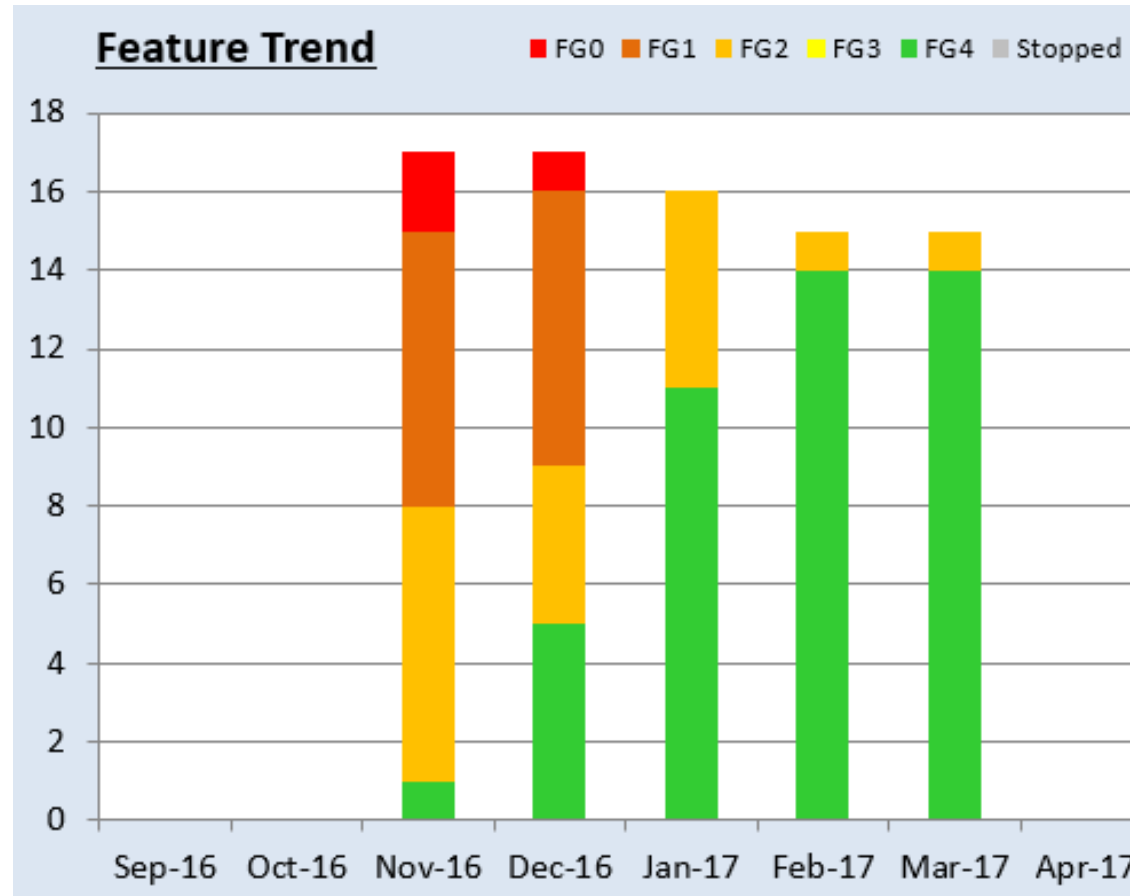
## BVMS Systemtest Burndown





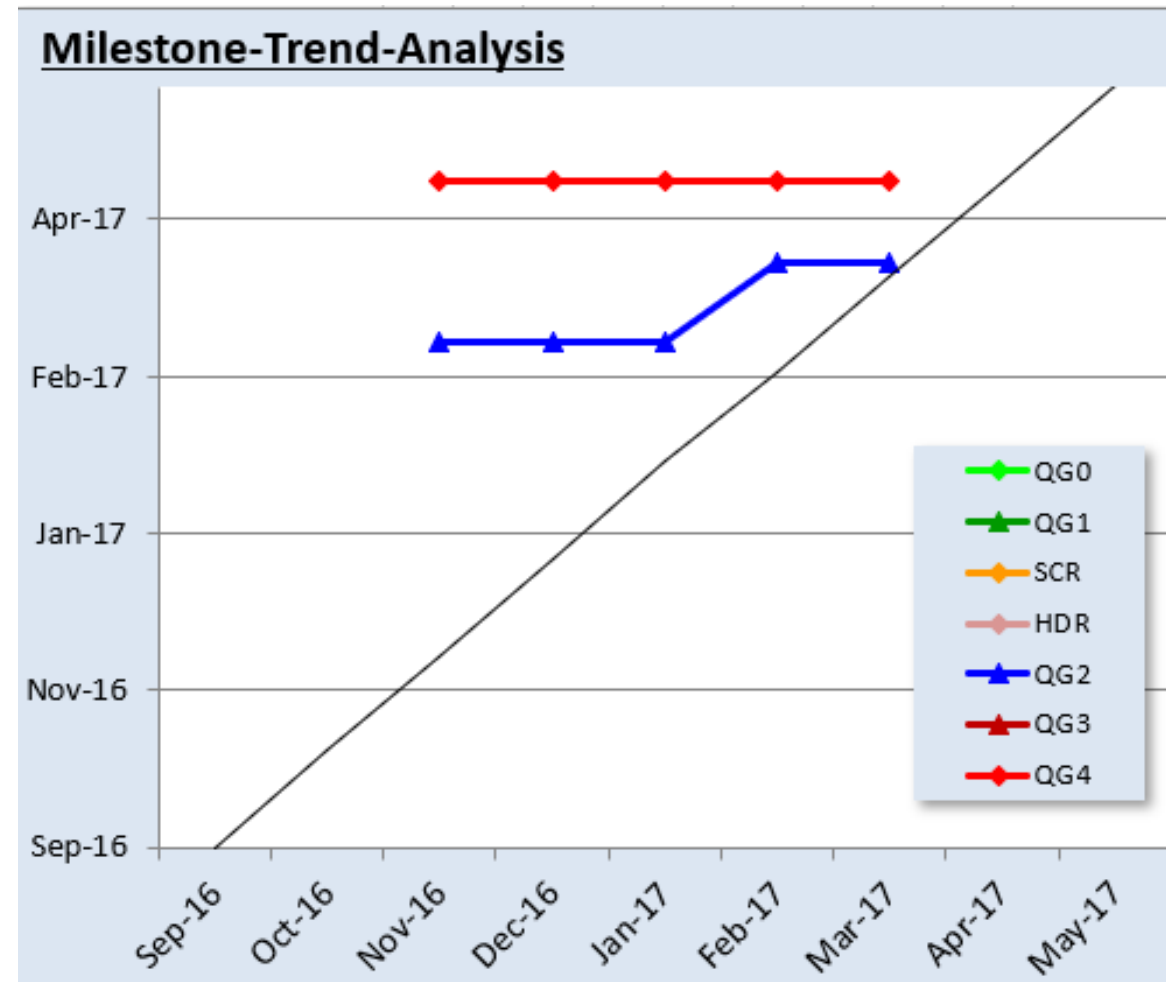
# Scrum-Prozess: Sprint review

## Feature burndown chart



# Scrum-Prozess: Sprint review

## Meilenstein-Trend-Analyse



### Retrospective:

- das Team analysiert seine Arbeitsweise und nimmt Prozessverbesserungen vor
- es klassifiziert Prozessänderungen in:
  - „More of...“
  - „Less of...“
  - „Continue...“
- üblicherweise durchgeführt an Metaplantafeln mit Karteikarten

# Scrum-Prozess: g) Launch

## Launch:

- Finale Release-/ Systemtests vor Auslieferung möglich!

# Einführung von agilen Vorgehensmodellen: Wie verankere ich agiles PM in einer Organisation?

## Erfahrungen am Beispiel SCRUM

- In der Praxis stellt sich die Einführung von SCRUM oder anderen agilen Methoden **oft als schwierig** dar.
- Häufiger Grund: die der **Methode zugrunde liegenden Werte passen zunächst nicht (ohne Weiteres) zur Kultur** des Unternehmens, und dies wird nicht hinreichend reflektiert und berücksichtigt.

## Die Einführung einer Methode wie SCRUM ist ein Projekt auf mehreren Ebenen:

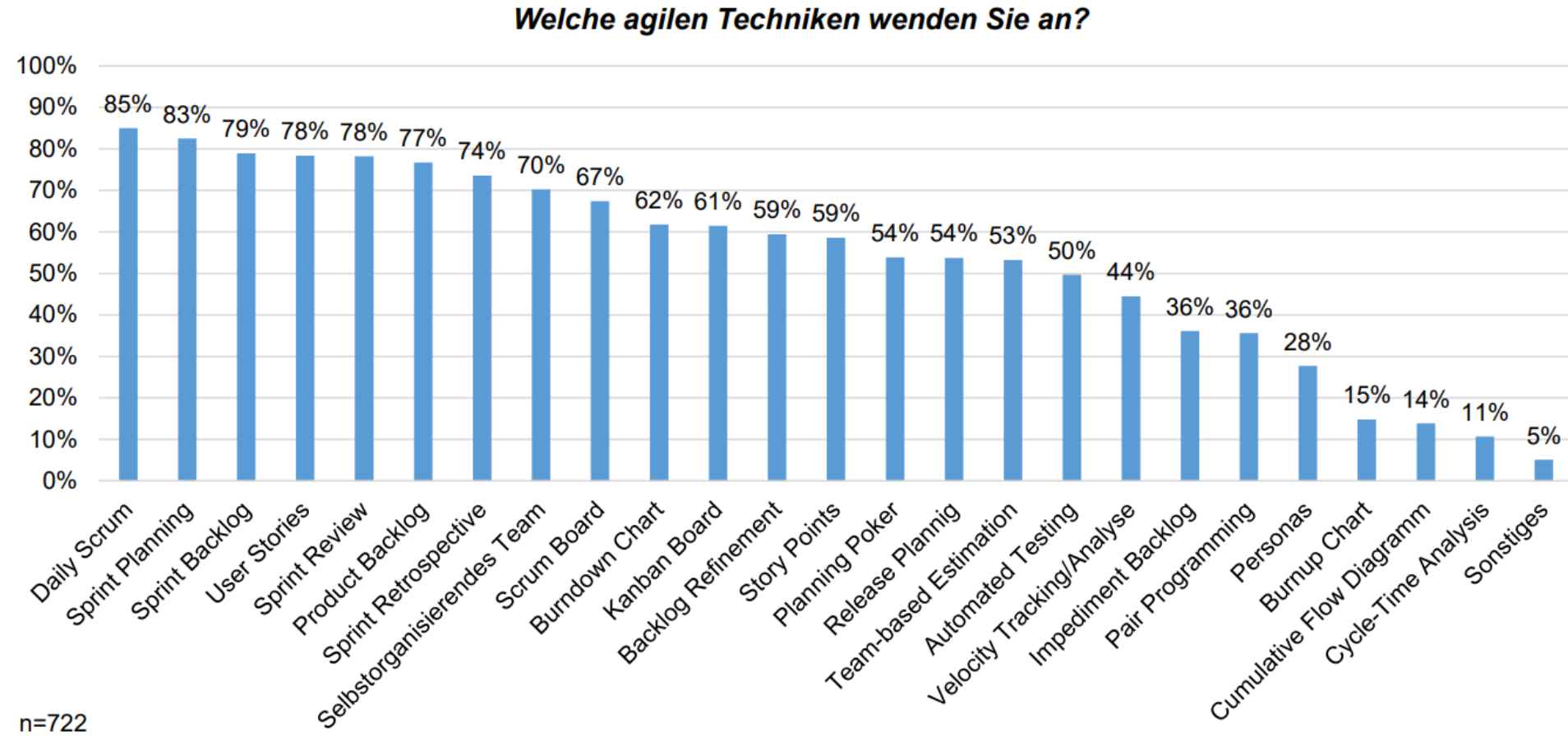
- Einführung der **agilen Methodik** im *Fachprojekt*
  - Training, Übung für die Teammitglieder
  - Methodik selbst ist prinzipiell einfach, bedarf aber der Übung/Unterstützung
- Einführung von **agilen Werten** im *Unternehmen*, Veränderung der Unternehmenskultur, ist also ein Organisations-Entwicklungsprojekt!
  - Dieses Change-Management wird oft vernachlässigt, oder Notwendigkeit ist gar nicht bewusst

# Einführung von agilen Vorgehensmodellen – Anwendung der Werte des Agilen Manifestes

- Methoden (wie SCRUM) und deren **zugrundeliegende Werte müssen zum jeweiligen Kontext**, in dem sie verwendet werden **passen** (Unternehmen, Kultur, Kompetenzen, Technologie usw. ).
  - Es ist eine für das Unternehmen **passende Methodik (Vorgehensmodell) zu entwickeln**, dass die Besonderheiten des jeweiligen Kontextes berücksichtigt.
- **Balance für die „agilen“ Werte individuell justieren:**

„Agile“ Werte/Prinzipien	„Klassische“ Werte/Prinzipien
Selbstorganisation	Hierarchische Führung
Individuen und Interaktionen	Prozesse und Werkzeuge
Funktionierende Software (frühes Ausliefern)	Dokumentation (Konzepte, Spezifikationen)
Vertrauensvolle Zusammenarbeit	Verträge und Vereinbarungen
Reagieren auf Veränderungen	Pläne

# Organisationen nutzen weder klassische noch agile Methoden durchgängig



(Komus, 2017)



## The bottom line

If you achieve these you can ignore the rest of the checklist. Your process is fine.

- ☐ Delivering working, tested software every 4 weeks or less
- ☐ Delivering what the business needs most
- ☐ Process is continuously improving

- ☐ Clearly defined product owner (PO)
  - ☐ PO is empowered to prioritize
  - ☐ PO has knowledge to prioritize
  - ☐ PO has direct contact with team
  - ☐ PO has direct contact with stakeholders
  - ☐ PO speaks with one voice (in case PO is a team)
- ☐ Team has a sprint backlog
  - ☐ Highly visible
  - ☐ Updated daily
  - ☐ Owned exclusively by the team
- ☐ Daily Scrum happens
  - ☐ Whole team participates
  - ☐ Problems & impediments are surfaced
- ☐ Demo happens after every sprint
  - ☐ Shows working, tested software
  - ☐ Feedback received from stakeholders & PO
- ☐ Have Definition of Done (DoD)
  - ☐ DoD achievable within each iteration
  - ☐ Team respects DoD

## Core Scrum

These are central to Scrum. Without these you probably shouldn't call it Scrum.

- ☐ Retrospective happens after every sprint
  - ☐ Results in concrete improvement proposals
  - ☐ Some proposals actually get implemented
  - ☐ Whole team + PO participates
- ☐ PO has a product backlog (PBL)
  - ☐ Top items are prioritized by business value
  - ☐ Top items are estimated
  - ☐ Estimates written by the team
  - ☐ Top items in PBL small enough to fit in a sprint
  - ☐ PO understands purpose of all backlog items
- ☐ Have sprint planning meetings
  - ☐ PO participates
  - ☐ PO brings up-to-date PBL
  - ☐ Whole team participates
  - ☐ Results in a sprint plan
  - ☐ Whole team believes plan is achievable
  - ☐ PO satisfied with priorities
- ☐ Timeboxed iterations
  - ☐ Iteration length 4 weeks or less
  - ☐ Always end on time
  - ☐ Team not disrupted or controlled by outsiders
  - ☐ Team usually delivers what they committed to
- ☐ Team members sit together
  - ☐ Max 9 people per team

# the unofficial Scrum Checklist

crisp  
Henrik Kniberg

## Recommended but not always necessary

Most of these will usually be needed, but not always all of them. Experiment!

- ☐ Team has all skills needed to bring backlog items to Done
- ☐ Team members not locked into specific roles
- ☐ Iterations that are doomed to fail are terminated early
- ☐ PO has product vision that is in sync with PBL
- ☐ PBL and product vision is highly visible
- ☐ Everyone on the team participates in estimating
- ☐ PO available when team is estimating
- ☐ Estimate relative size (story points) rather than time
- ☐ Whole team knows top 1-3 impediments
  - ☐ SM has strategy for how to fix top impediment
  - ☐ SM focusing on removing impediments
  - ☐ Escalated to management when team can't solve
- ☐ Team has a Scrum Master (SM)
  - ☐ SM sits with the team
- ☐ PBL items are broken into tasks within a sprint
  - ☐ Sprint tasks are estimated
  - ☐ Estimates for ongoing tasks are updated daily
- ☐ Velocity is measured
  - ☐ All items in sprint plan have an estimate
  - ☐ PO uses velocity for release planning
  - ☐ Velocity only includes items that are Done
- ☐ Team has a sprint burndown chart
  - ☐ Highly visible
  - ☐ Updated daily
- ☐ Daily Scrum is every day, same time & place
  - ☐ PO participates at least a few times per week
  - ☐ Max 15 minutes
  - ☐ Each team member knows what the others are doing

## Scaling

These are pretty fundamental to any Scrum scaling effort.

- ☐ You have a Chief Product Owner (if many POs)
- ☐ Dependent teams do Scrum of Scrums
- ☐ Dependent teams integrate within each sprint

## Positive indicators

Leading indicators of a good Scrum implementation.

- ☐ Having fun! High energy level.
- ☐ Overtime work is rare and happens voluntarily
- ☐ Discussing, criticizing, and experimenting with the process

# Scrum vs. XP (Extreme Programming)

15 min. zum Lesen/ Verstehen:

<https://www.coscreen.co/blog/extreme-programming-vs-scrum-difference/#:~:text=Extreme%20programming%20is%20a%20software,with%20products%20other%20than%20softw are.>

Fragen?

(Komus, 2017)

Richtig oder falsch?

1. „... You can't say you're using scrum if you don't adopt the whole of it ...“
2. Pair programming, Test driven design und Collective code ownership sind ‚Best practices‘ von XP.
3. Wie XP kann auch Scrum nur für Softwareentwicklungsprojekte eingesetzt werden
4. Die DoD definiert, wann ein Feature als fertig angesehen werden kann.
5. Die 5 Scrum-Werte sind: Commitment, Fokus, Offenheit, Respekt und Mut.

(Komus, 2017)

# Team-Velocity

Was ist Velocity (20 min.)?

<https://www.agile-academy.com/de/scrum-master/was-bedeutet-velocity-fuer-ihr-team/>

Häufiger Missbrauch – bitte vermeiden bzw. abwehren:

<https://www.colenet.de/blog/scrum/3-fehler-die-scrum-teams-bei-der-verwendung-von-velocity-vermeiden-sollten-und-wie-sie-stattdessen-performance-prognosesicherheit-und-wert-messen-koennen/>

Fragen?

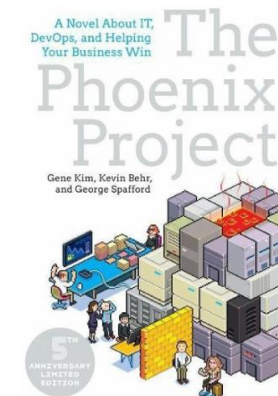
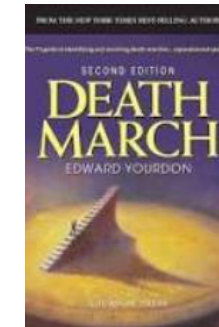
(Komus, 2017)

„*Agile Estimating and Planning*“, M.Cohn, Prentice Hall, 2011.  
„*Agile Testing: A Practical Guide for Testers and Agile Teams*“,  
L. Crispin, Addison-Wesley, 2008.

“*Wien wartet auf Dich! Produktive Projekte und Teams.*”  
T. de Marco, T. Lister, Hanser, 2014.

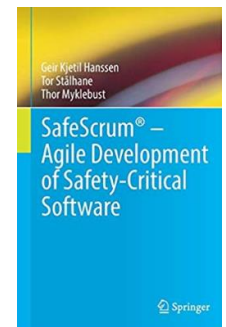
„*Death March. The Complete Software Developer’s Guide to Surviving „Mission Impossible“ Projects.*“  
E. Yourdon, Prentice Hall, 1999.

“*The Phoenix Project.*  
*A Novel about IT, DevOps, and Helping Your Business Win.*”  
G. Kim, K. Behr, G. Spafford, IT Revolution Press, 2014





- Cohn, M. 2010. *Succeeding with Agile. Software development using Scrum*. Boston Addison-Wesley.
- Cohn, M. 2004. *User stories applied for Agile Software Development*. Boston Addison-Wesley.
- Johnson, J. 2006. *My Life is Failure*. The Standish Group Int'l, Inc.
- Baumgartner, M., Klonk, M., Pichler, H., Seidl, R., Tanczos, S. 2013. *Agile Testing*. München: Carl Hanser.
- Gregory, Janet; Crispin, Lisa: *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional; 2009
- Gregory, Janet; Crispin, Lisa: *More Agile Testing*. Addison-Wesley Professional; 2014
- Hanssen, Geir Kjetil; Stalhane, Tor; Myklebust, Thor: *SafeScrum® – Agile Development of Safety-Critical Software*. Springer 2018.
- Eckstein, J. 2009. *Agile Softwareentwicklung mit verteilten Teams*. Heidelberg: dpunkt.verlag.
- Opelt, A., Gloger, B., Pfarl, W., Mittermayr, R. 2014. *Der agile Festpreis*. München Hanser.
- Brooks F. P. jun. 2003. *Vom Mythos des Mann-Monats*. Bonn: mitp-Verlag.



- <http://www.scrumguides.org/>
- <http://www.extremeprogramming.org/>
- <http://www.istqb.org/certification-path-root/agile-tester-extension.html>
- <http://agilemanifesto.org/>
- <http://safescrum.no>
- Beck, Kent: Extreme Programming Explained: Embrace Change, Addison-Wesley Longman, 1999, (aktuelle Ausgabe: 2. Aufl. November 2004)
- Martin, Robert C.: Agile Software Development. Principles, Patterns, Practices, Prentice Hall, 2003, (Neuaufgabe: Pearson, International ed., März 2011)
- Beck, Kent u.a.: Manifesto for Agile Software Development, 2001
- Sutherland, K. Schwaber, Der Scrum-Guide – Der gültige Leitfaden für Scrum: Die Spielregeln, Juli 2013.
- D. Mühlbauer: Das Q in Agile. Qualitätssicherung in agilen Projekten. OBJEKTspektrum, Ausgabe Agility, 2015.
- G. Utas: Robust Communications Software. Extreme Availability, Reliability and Scalability for Carrier-Grade Systems. Wiley 2005.



# Verwendete Quellen

- Bennett, N. und Lemoine, G.J. 2014. "What VUCA Really Means for You", *Harvard Business Review* (92:1/2), S. 27-27.
- Jenny, B. 2014. *Projektmanagement: Das Wissen für den Profi*, (3. Auflage). Zürich: vdf Hochschulverlag AG.
- Komus, A. 2016. "agiles Projektmanagement - agiles Management", Frankfurt, [https://www.komus.de/app/download/8835259686/2016-02-agiles\\_Management-PM.pdf?t=1505927227](https://www.komus.de/app/download/8835259686/2016-02-agiles_Management-PM.pdf?t=1505927227), 07.06.2019.
- Komus, A. 2017. "Abschlussbericht: Status Quo Agile 2016/2017 - 3. Studie über Erfolg und Anwendungsformen von agilen Methoden", Hochschule Koblenz, Koblenz, <https://www.process-and-project.net/studien/download/downloadbereich-status-quo-agile/>, 07.06.2019.
- Schlauderer, S., Overhage, S. und Fehrenbach, B. 2015. "Widely used but also highly valued? Acceptance factors and their perceptions in water-scrum-fall projects", in *Proceedings of the Thirty Sixth International Conference on Information Systems (ICIS)*, Fort Worth.
- Stacey, R.D. 1996. *Strategic Management & Organisational Dynamics*, (2. Auflage). London: Pitman.
- Tiemeyer, E., Beims, M., Bergmann, R. und Ebert, C. 2018. *Handbuch IT-Projektmanagement: Vorgehensmodelle, Managementinstrumente, Good Practices*, (3. Auflage). München: Carl Hanser.
- Wieczorrek, H.W. und Mertens, P. 2011. *Management von IT-Projekten: Von der Planung zur Realisierung*, (4. Auflage). Berlin, Heidelberg: Springer.