

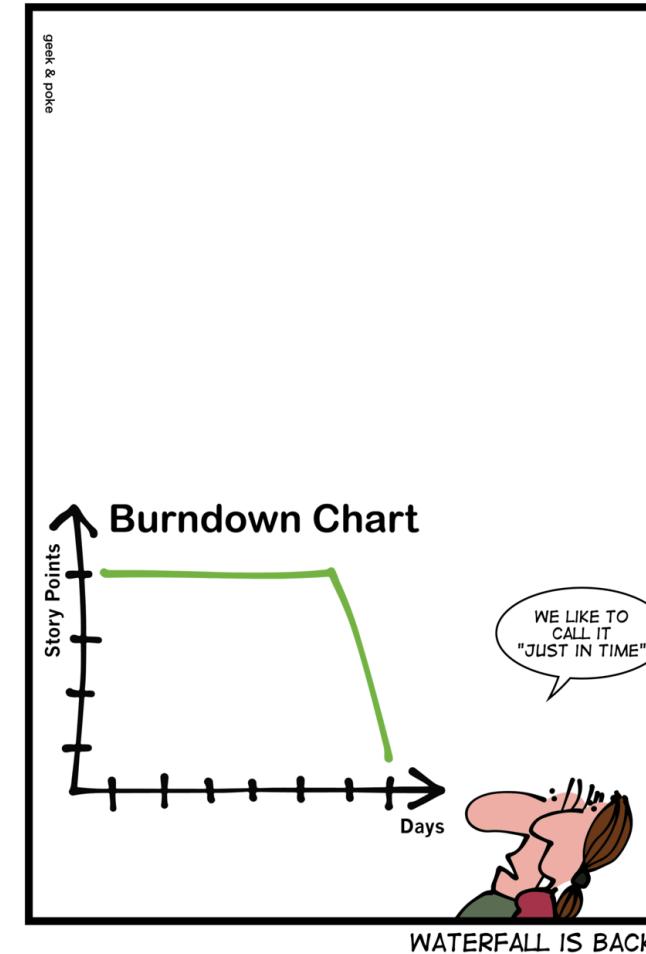


# Vorgehens- und Prozessmodelle

HAW Hamburg / Fachbereich Informatik

Tim Lüecke

([Tim.Lueecke@haw-hamburg.de](mailto:Tim.Lueecke@haw-hamburg.de))

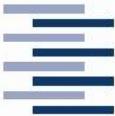




# Agenda

## ■ **Motivation und Überblick**

- Traditionelle Modelle
- Agile Modelle
- Auswahl des richtigen Modells
- Zusammenfassung



# Code & Fix

## ■ Vorteile

- Entspricht unserem Drang, schnell voranzukommen
- Liefert schnell ein lauffähiges Programm
- Tätigkeiten (codieren und spontanes Testen) sind relativ einfach

## ■ Nachteile

- Projekt ist nicht planbar; keine Entscheidung, was in welcher Qualität hergestellt werden soll
- Fehlerbehebung führt zu Umstrukturierung, wodurch Fehlerbehebung und Weiterentwicklung teurer werden
- Selbst entworfene Software wird von Benutzern nicht akzeptiert
- Fehler sind schwer zu finden, da Tests schlecht vorbereitet sind und unzureichend durchgeführt werden
- „Code & Fix“ sabotiert die Qualität

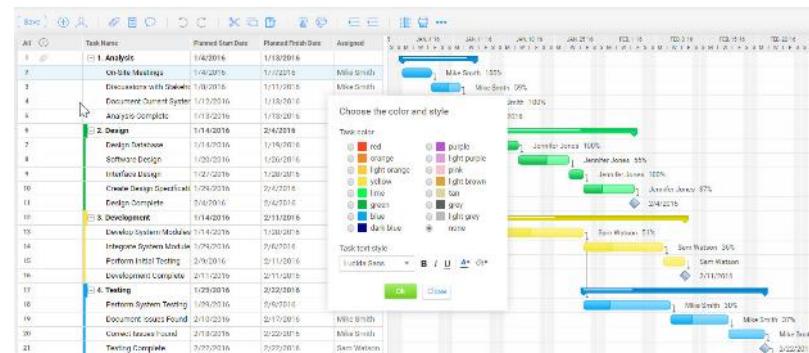




# Übersicht über Vorgehens- und Projektmodelle

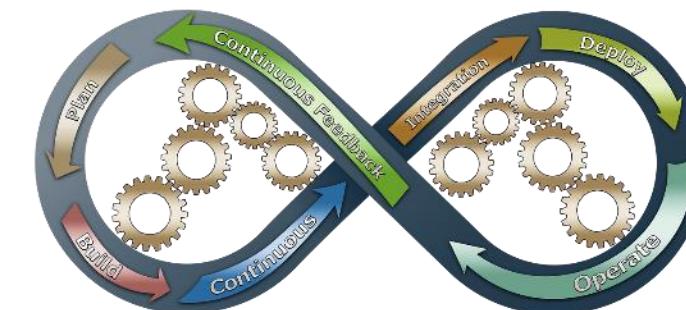
## Klassische / Traditionelle Modelle

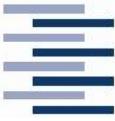
- Wasserfallmodell
- Iteratives Wasserfallmodell
- V-Modell
- Rational Unified Process



## Agile Modelle

- Lean
- Extreme Programming
- Kanban
- SCRUM





# Agenda

- Motivation und Überblick

- **Traditionelle Modelle**

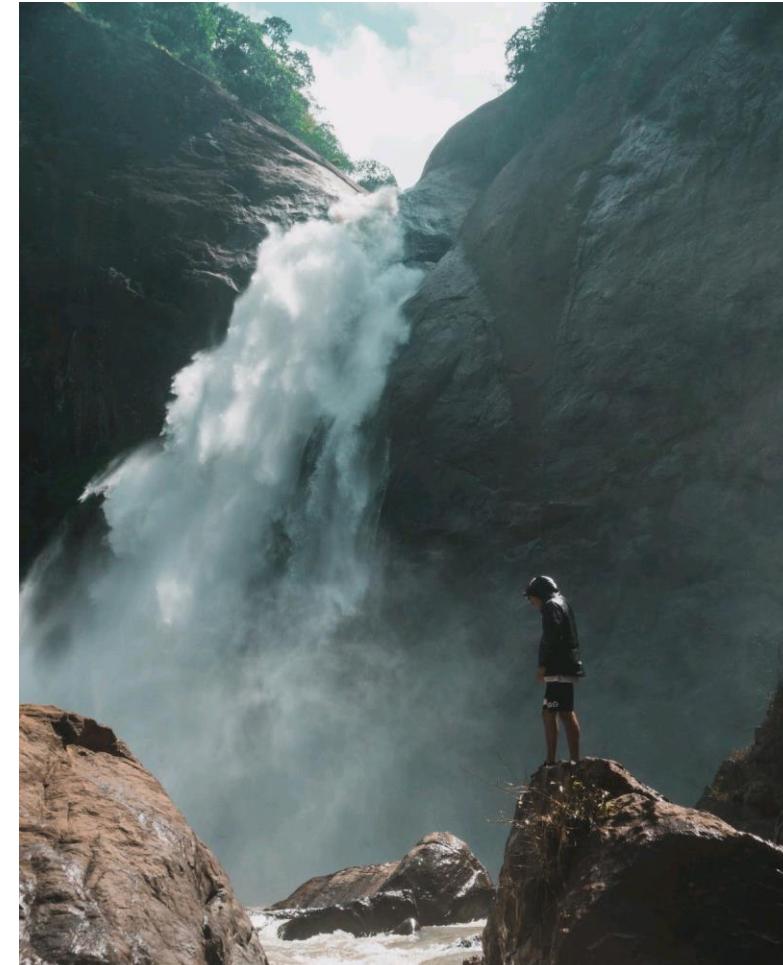
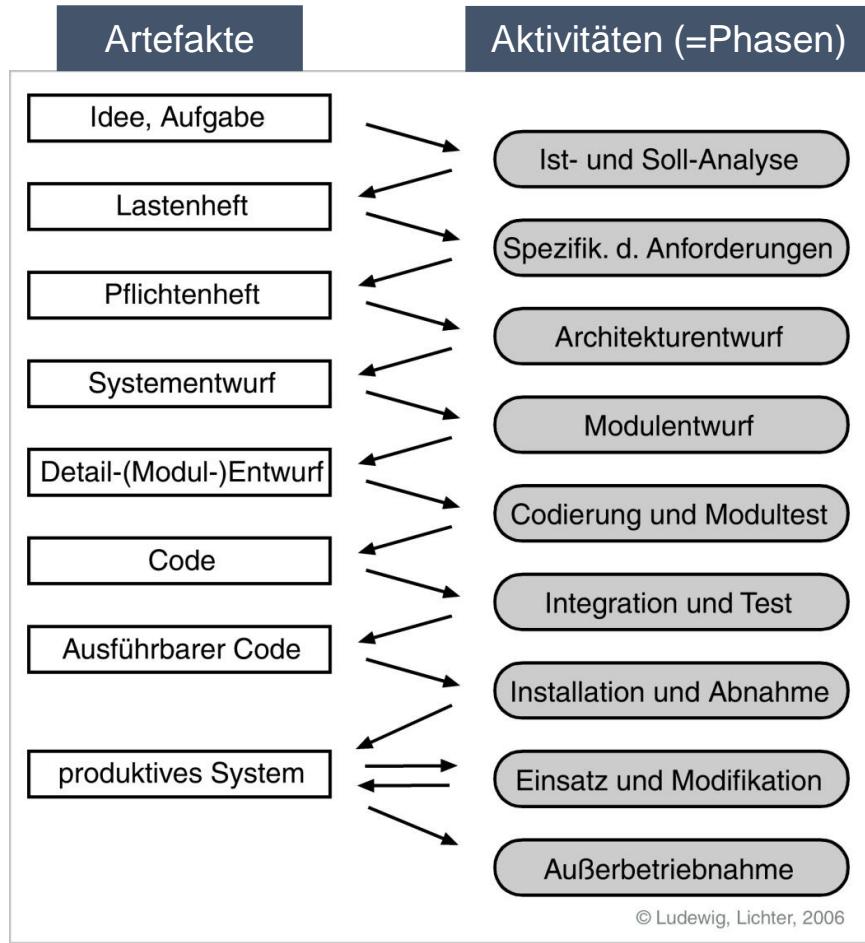
- Agile Modelle
- Auswahl des richtigen Modells
- Zusammenfassung

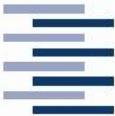


# Wasserfallmodell

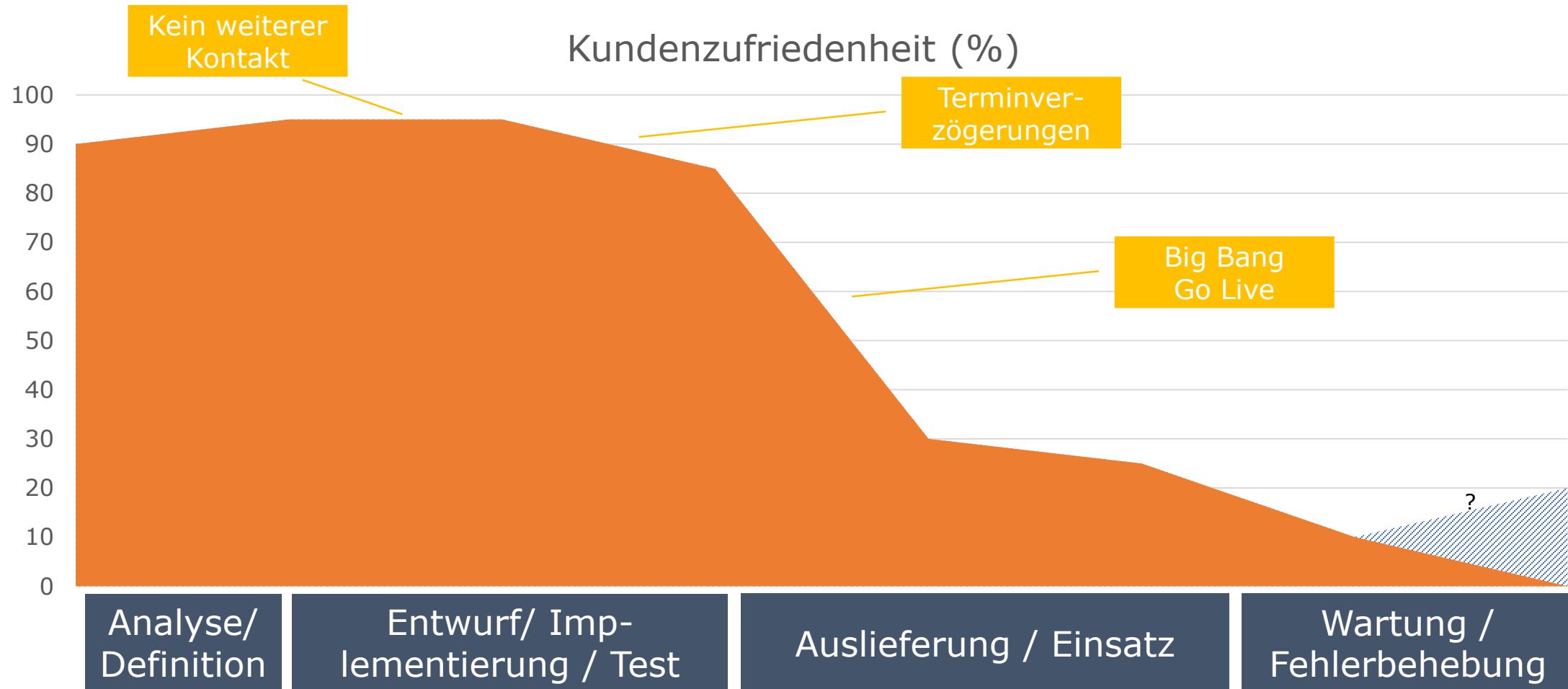


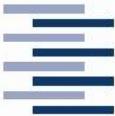
# Wasserfallmodell





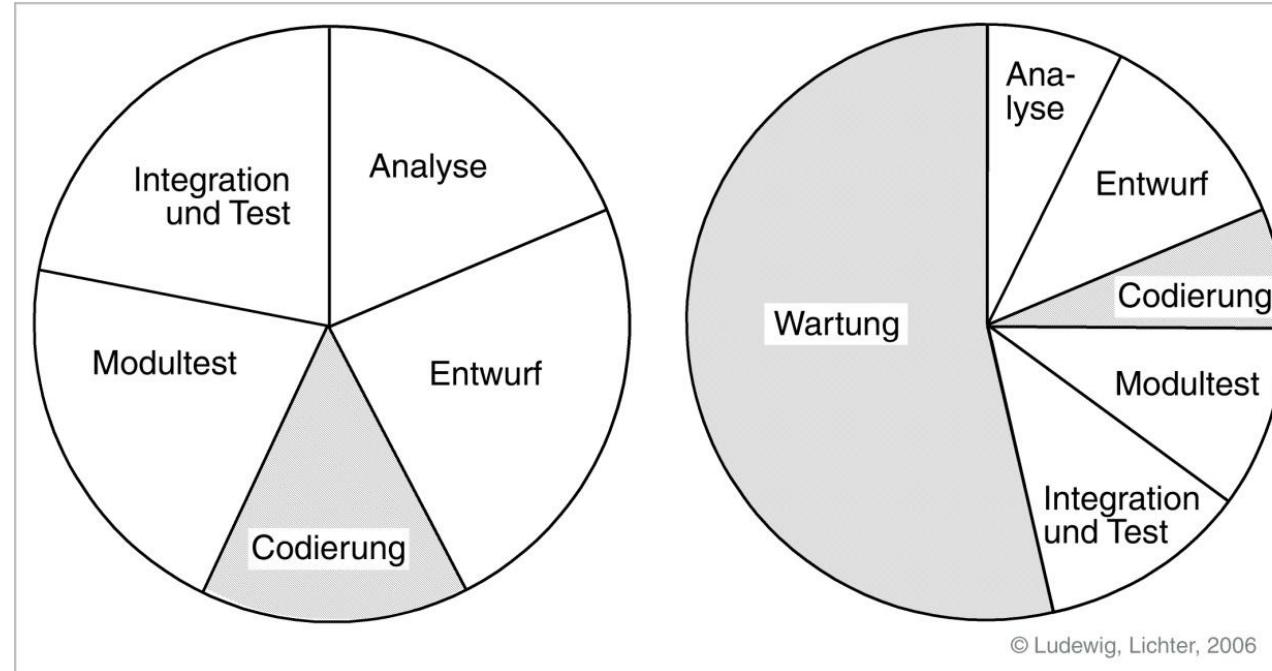
# Das “kundenorientierte” Grundproblem von Wasserfallmodellen





# Aufwandsverteilung und Kosten

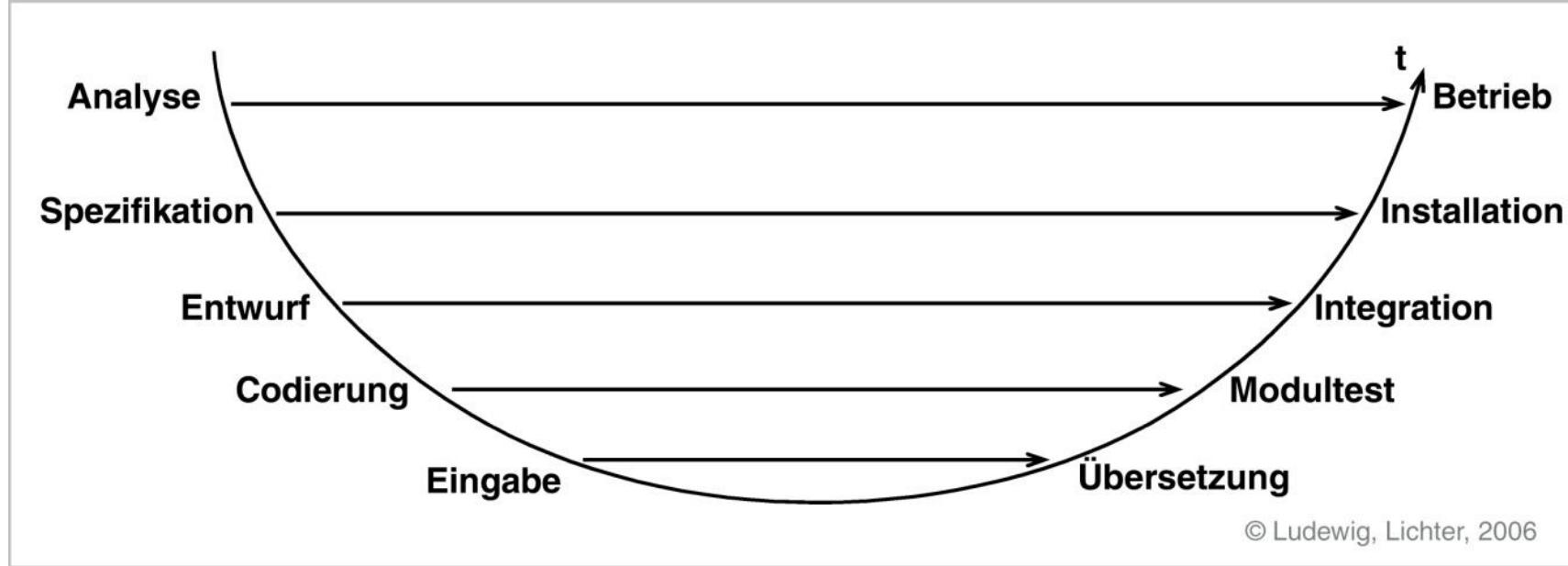
- Aufwand in den einzelnen Phasen des Software-Projekts (nach Boehm)



- 40/20/40 (%-Anteile auf Analyse+Entwurf, Implementierung, Test inkl. Korrektur); [Boehm, 1987]
- 60/15/25; [Baskette, 1987]
- Kostenanteil der Codierung in Wahrheit typischerweise nur **15-20%** der Entwicklungskosten



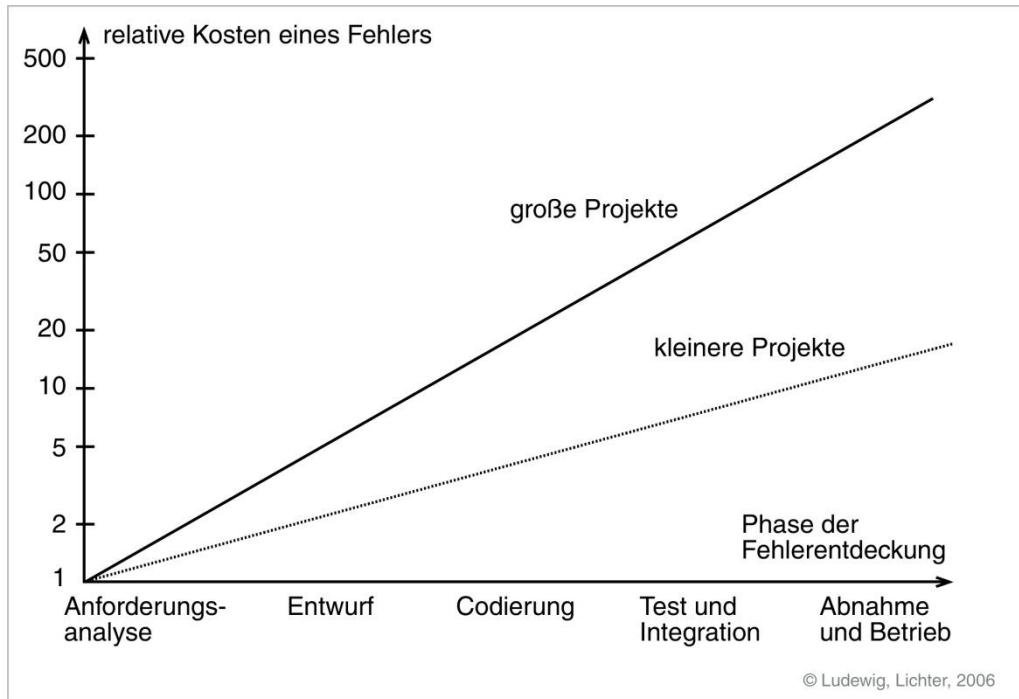
# Beziehung zwischen Fehlerentstehung und -entdeckung



typischerweise werden Fehler auf derselben Abstraktionsebene entdeckt, auf der sie begangen werden; Fehler in der Analyse werden meist erst im Betrieb entdeckt



# Relative Kosten eines Fehlers



- Fehlerkosten steigen mit der Latenzzeit **exponentiell** an
- daher: **frühe Fehler** sind aus SE-Sicht die schlimmsten; hier rentiert sich ein **hoher Aufwand**



# Inkrementelles Wasserfallmodell



# Iteratives Wasserfallmodell

Release 1      Release 2      Release 3



- Projektphasen überschneiden sich
- Kein Big Bang sondern Unterteilung in Sub-Releases mit separater Abnahme
- Überschneidung der Sub-Release Entwicklungszeiträume

- **Vorteile:**

- Kundenorientierter: Validierung schon nach jedem Sub-Release
- Separate Teams pro Phase ermöglichen bessere Resourcennutzung: Mitarbeiter können direkt im nächsten Sub-Release weiter arbeiten

- **Nachteile:**

- Schwieriger und fragiler zu planen (z.B. Schnitt der einzelnen Releases)
- Bei Verzögerungen können Leerläufe entstehen
- Silo-Bildung zwischen den Teams
- Change Requests erfordern Änderung des Gesamtplans



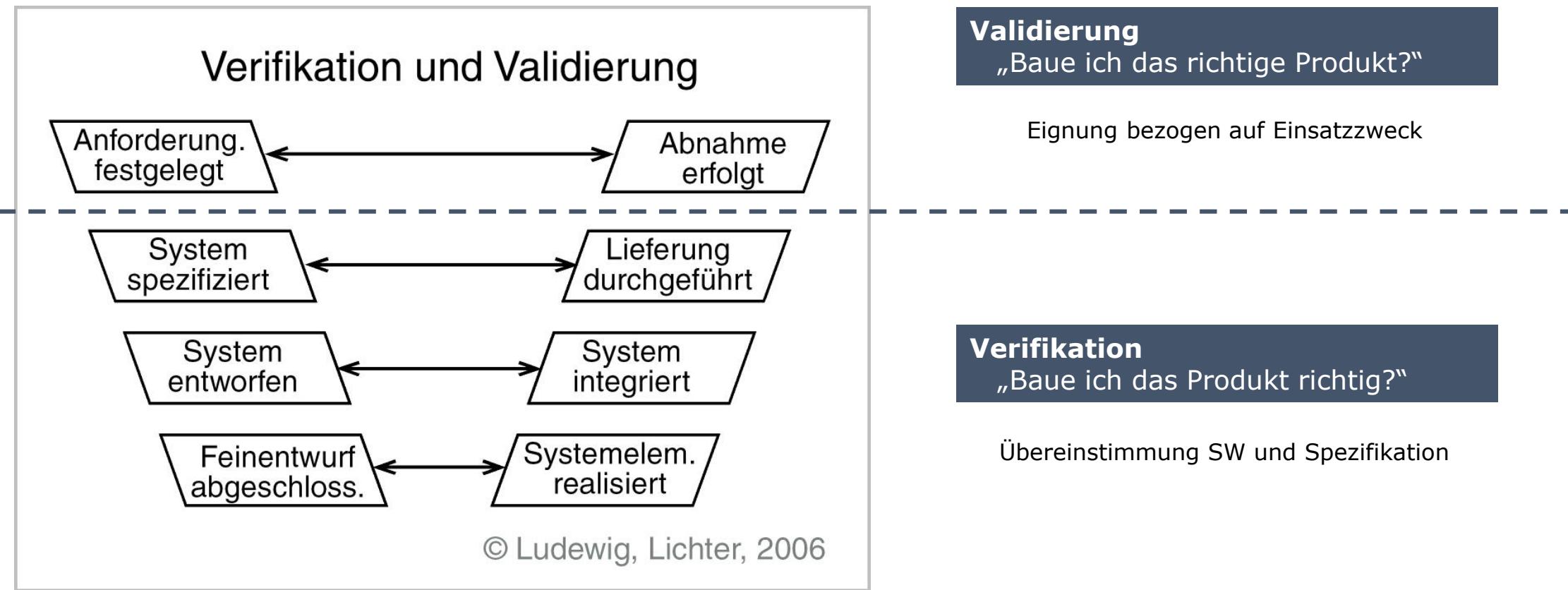
# V-Modell



# V-Modell nach Boehm

**Ziel:** Integration von Wasserfallmodell und Qualitätssicherung

**Prinzip:** Gegenüberstellung von Entwicklungs- und Verifikationstätigkeiten





# Begriffsklärung V-Modell

## **Vorsicht, Begriffsverwirrung**

- hier: **V-Modell nach Boehm**
- Weiterentwicklung des Boehm-Modells:  
**V-Modell ® des Bundes**
  - Standardisierung der Software-Bearbeitung im Bereich der Bundesverwaltung
  - [V-Modell ® XT \(„extreme Tailoring“\)](#):  
Mitwirkende: u.a. Airbus Defence & Space, Siemens AG, ...
  - Umfang Version 2.2: 455 Seiten  
(1.x noch 900 Seiten)
  - Schwerpunktiger Prozess mit vielen Vorgaben zu Organisation, Projektmanagement, Konfigurationsmanagement, Methoden





# Prototyping + Spiralmodell



# Prototyping

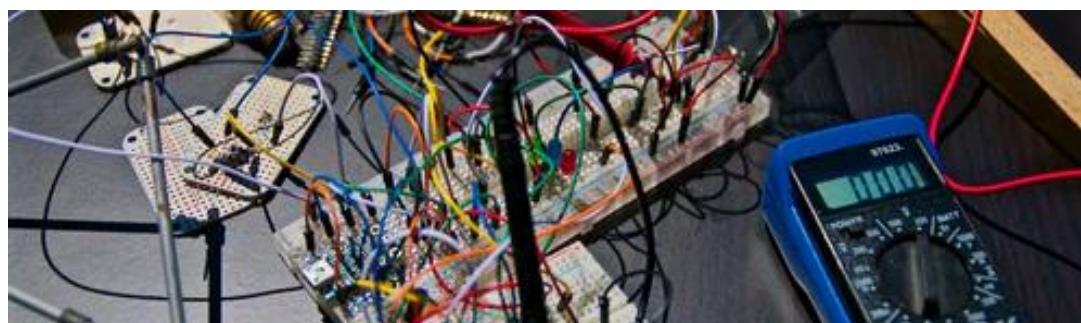
## Arten von Prototypen

- **Horizontaler Prototyp:**

Realisiert Funktion einer Systemebene vollständig

- **Vertikaler Prototyp:**

Realisiert ausgewählte Teile des Systems durch alle Ebenen (Ebenen: GUI, Anwendungskern/Geschäftslogik, Zugriff-/Persistenzschicht, usw.)



## Bewertung

- **Vorteile**

- Reduktion des Entwicklungsrisikos (Risikomanagement!)
- Bessere Einbindung von Benutzer und Auftraggeber
- Schnell erstellbar, da es nicht auf Qualität ankommt

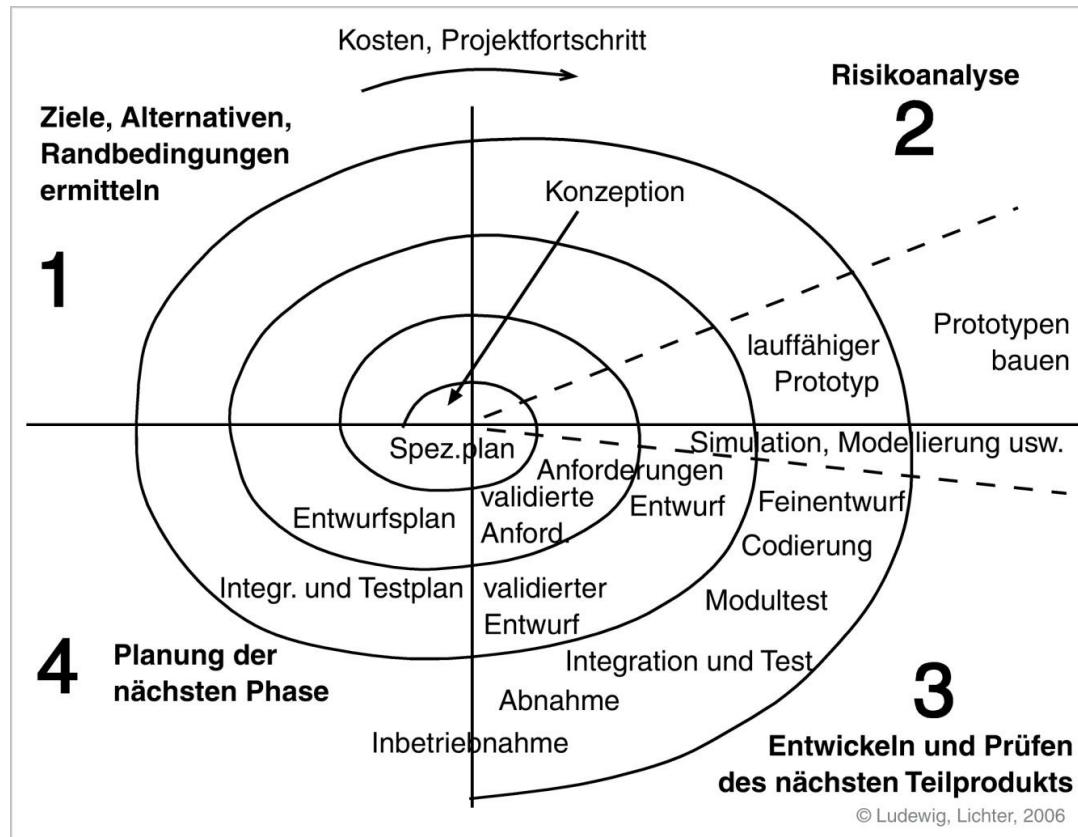
- **Nachteile**

- Zusätzlicher Aufwand
- Unterscheidung zwischen Wesentlichem und Details verwischt
- Gefahr, dass der qualitativ minderwertige Prototyp als Basis für das eigentliche System verwendet wird



# Spiralmodell (nach Boehm)

## Generisches Modell für PM (von 1986)



## Bewertung

### Vorteile

- Flexibler Prozessablauf
- Modell wird nicht für gesamte Entwicklung festgelegt
- Entwicklung kann flexibel angepasst werden (inkl. Abbruch)
- Fehler und ungeeignete Alternativen werden frühzeitig eliminiert, da Risikobetrachtung explizit integriert ist

### Nachteile

- Genaue Planung? Daher schwierig in externen vertrags-basierten Projekten
- Erfahrung für die Beurteilung der Risiken benötigt



# Rational Unified Process



# RUP: Rational Unified Process (fromer Unified Process (UP))

- Basiert auf Prinzipien und wesentlichen Prozess-Bestandteilen
- Gibt Rollen, Aufgaben und Artefakte für die Softwareentwicklung vor
- Ist zugeschnitten für Objektorientierte Software und anpassbar für jedes Projekt
- Der RUP ist eine konkrete, kommerzielle Ausprägung des Unified Process (UP)
- Schlägt Tools und Methoden vor, enthält viele Templates für Dokumente





# Grundlagen von RUP

## Prinzipien

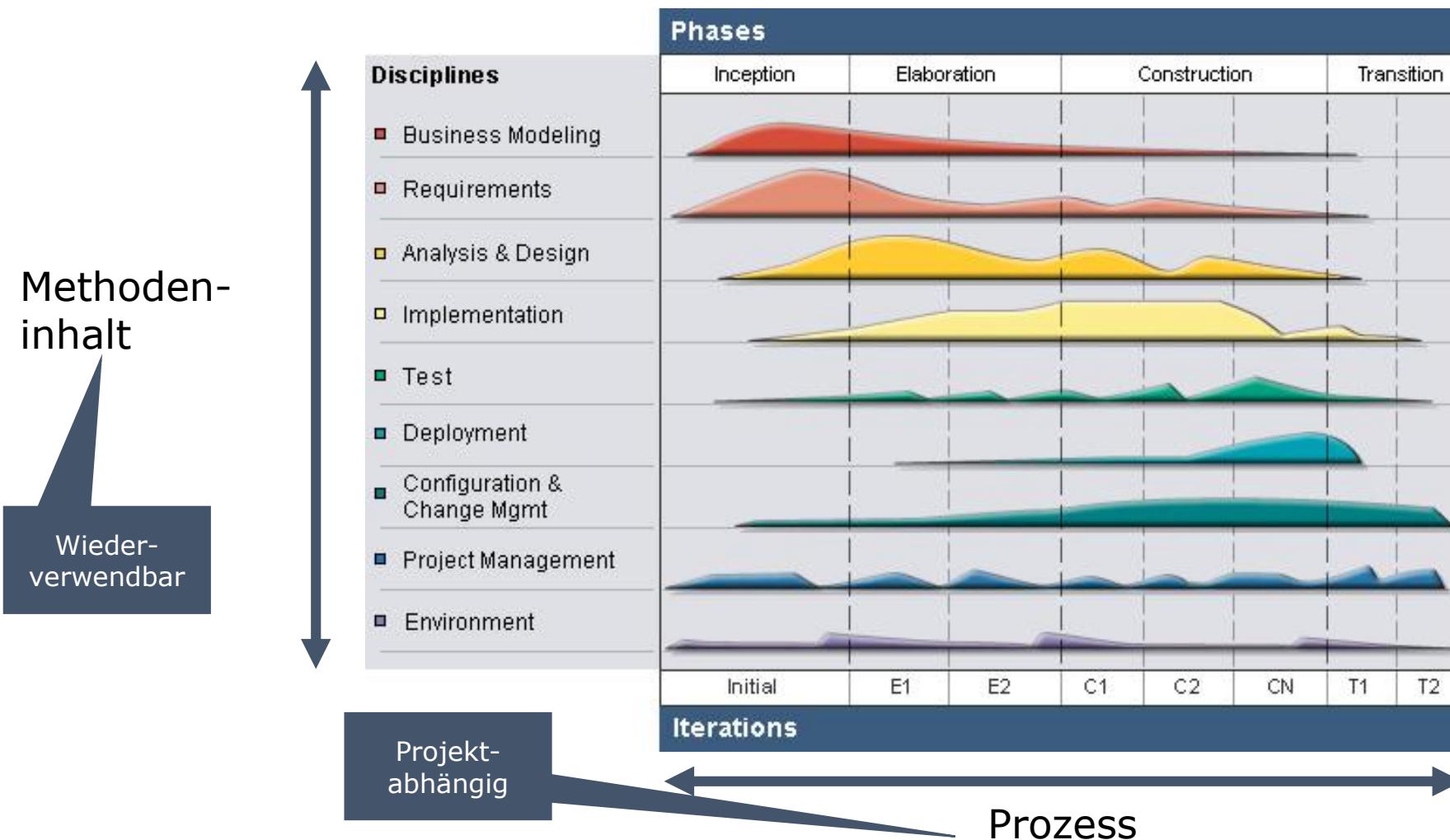
- Adapt the Process
- Balance Competing Stakeholder Priorities
- Collaborate Across Teams
- Demonstrate Value Iteratively
- Elevate Level of Abstraction
- Focus Continuously On Quality

## Wesentliche Prozesselemente

1. **Vision:** Develop a Vision
2. **Plan:** Manage to the Plan
3. **Risks:** Mitigate Risks and Track Related Issues
4. **Business Case:** Examine the Business Case
5. **Architecture:** Design a Component Architecture
6. **Prototype:** Incrementally Build and Test the Product
7. **Evaluation:** Regularly Assess Results
8. **Change Requests:** Manage + Control Changes
9. **User Support:** Deploy a Usable Product
10. **Process:** Adopt a Process that Fits Your Project



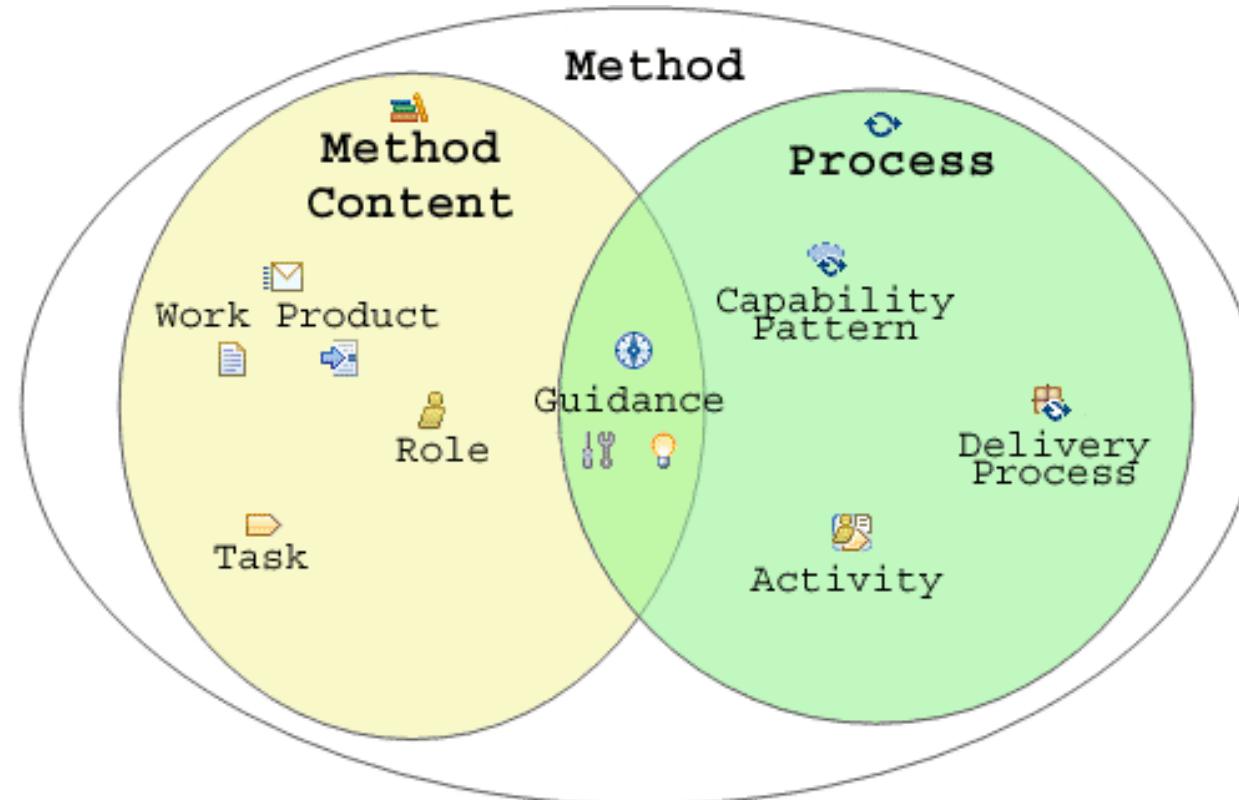
# Unified Method Architecture (UMA)



Source: <https://www.ibm.com/developerworks/rational/library/dec05/haumer/haumer-pdf.pdf>



# UMA: Methodeninhalt und Prozesse



Source: <https://www.ibm.com/developerworks/rational/library/dec05/haumer/haumer-pdf.pdf>



# Phasen im RUP: pro Projekt unterschiedlich ausgeprägt

Ziele  
Artefakte  
Meilenstein

## Inception

- Projektokus finden
- Anforderungen soweit verstehen, dass Aussagen zur Wirtschaftlichkeit gemacht werden können
- Bewerten der Risiken
- „Vision Document“: Kernanforderungen, Beschränkungen
- Risikobetrachtungen
- Zentrale Anwendungsfälle
- erste Prototypen für risikoreiche Bereiche

„Lifecycle Objective“: Soll das Projekt durchgeführt werden?

## Elaboration

- fehlende Anforderungen identifizieren
- Grundlegende Architektur-entscheidungen
- erster Prototyp mit Kern
- Risikomanagement
- Anwendungsfälle
- Nichtfunktionale Anforderungen
- Architektur

“Lifecycle Architecture”

## Construction

- Implementierung des Systems auf Basis der Architektur
- Integration, Test
- Benutzerdokumentation
- Software
- Dokumentation
- Releasebeschreibung

„Initial Operational Capability“: Ist das Produkt stabil genug?  
(Beta-Version der Software)

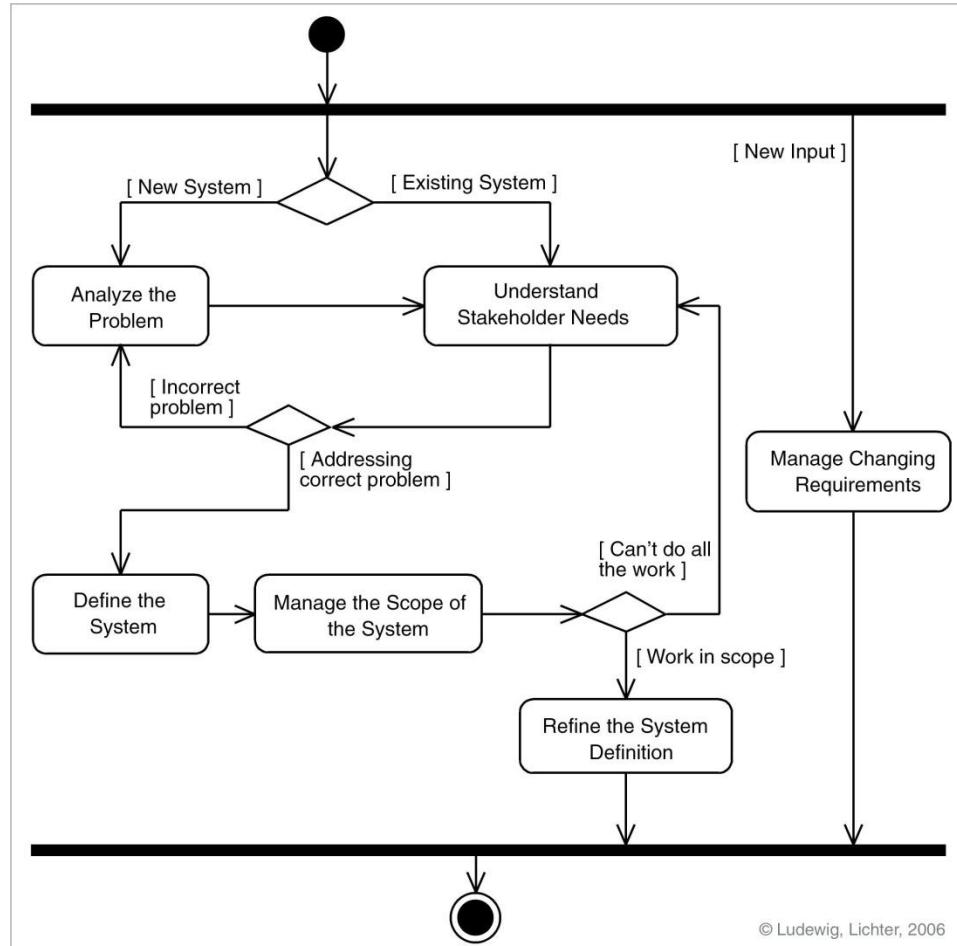
## Transition

- Live System wird durch Feedback verbessert
- Vervollständigung System und Dokumente
- Migration (Datenbanken)
- Parallelbetrieb
- Einsetzbarkeit
- Software
- Dokumentation

„Product Release“: Ist der Benutzer zufrieden?



# Disziplinen im RUP

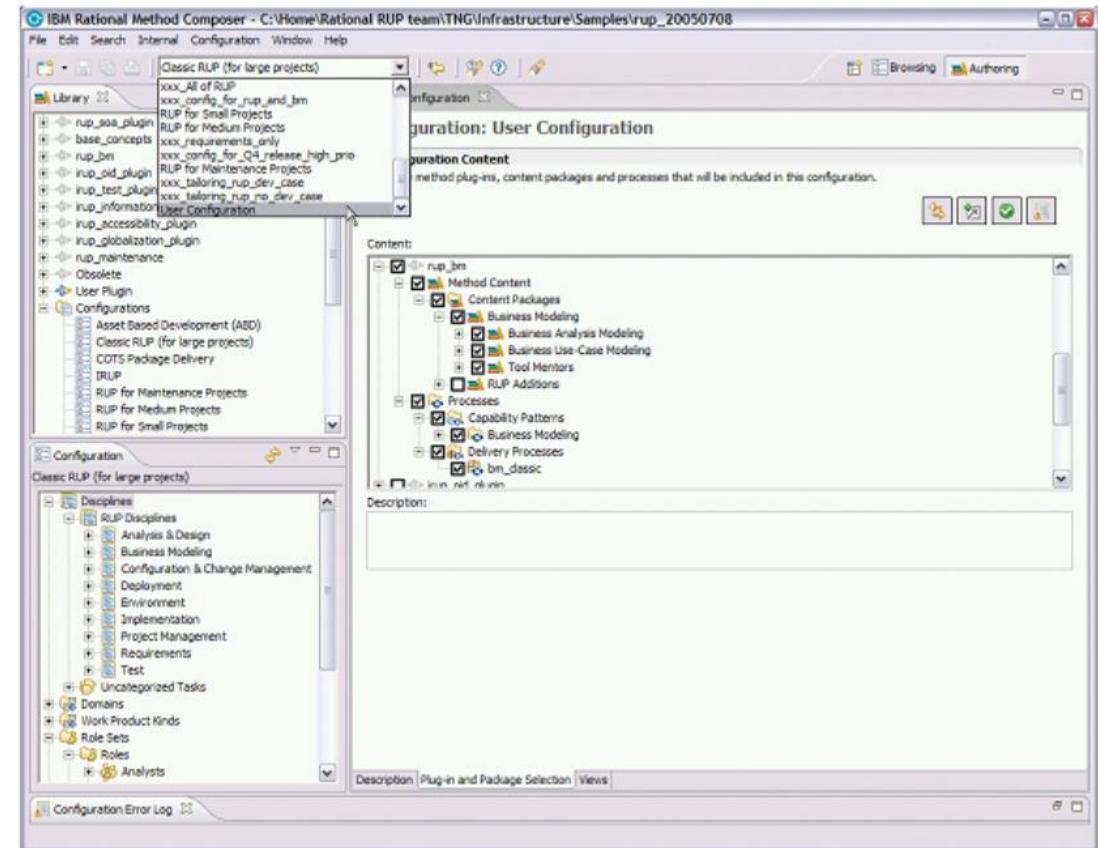


- *Beispiel:* Requirements Disziplin im RUP
- Im RUP sind alle Disziplinen durch Aktivitätsdiagramme und Aufgabenbeschreibungen definiert



# Anpassungen an das konkrete Projekt: Rational Method Composer (RMC)

- **Plattform** zur Konfiguration einer konsistenten Prozessbeschreibung für das konkrete Projekt
- Methodik wird über ein eigenes **Tool konfiguriert** und angepasst
- Über einen **Marktplatz** können existierende Erweiterungen und Anpassungen integriert werden
- **Ergebnis** ist eine navigierbare Webseite als Beschreibung der Methodik





# Vorraussetzungen, Vor- und Nachteile von RUP

## Vorraussetzungen

- Gute Projektmanagement-Fertigkeiten (Planung der Anzahl und Dauer der einzelnen Iterationen)
- Kenntnisse objektorientierter Konzepte

## Vorteile

- Gute Darstellung als HTML-basierte Dokumentation
- Hohe Detaillierung
- Hohe Anpassbarkeit

## Nachteile

- Anpassung an konkrete Gegebenheiten durchaus komplex
- Prozessdefinition ist nicht stabil
- Vortäuschen einer algorithmischen Vorgehensweise durch Disciplines/Workflows
- es zählt jedoch das jeweilige Ergebnis eines Arbeitsschrittes!



Wer kann sich vorstellen mal in einem klassischen Vorgehensmodell zu arbeiten?

**pingo.upb.de → 157137**





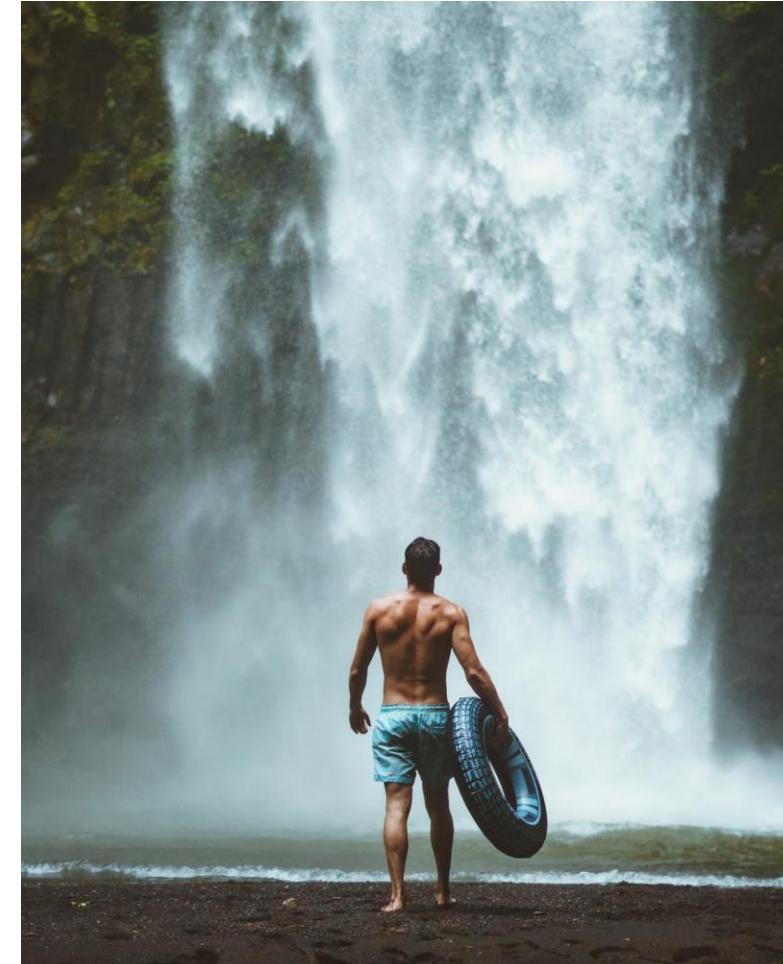
# Agenda

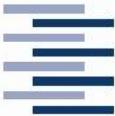
- Motivation und Überblick
- Traditionelle Modelle
- **Agile Modelle**
- Auswahl des richtigen Modells
- Zusammenfassung



# Probleme des Wasserfalls?

- **Unklare** Zielvorstellungen und Anforderungen des Kunden
- → **Häufige Änderungen** während des Projekts (Change Requests)
- Arbeitspakete, die nicht im kurzfristigen Planungshorizont liegen, bergen zu viele **Unsicherheiten**
- **Silobildung**: Zu stark abgegrenzte Aktivitäten/Teams („Teilprodukte über die Mauer werfen“)
- Test des Produkts am Ende ist viel **zu spät**
- ...





# Kritik an traditionellen Ansätzen

- Software-Bürokratie
- Kunde zu spät eingebunden
- Vertragsorientierung statt Kundenorientierung
- Kein Spaß
- Träge und teuer
- ...
- Typischerweise begründet:
  - durch Beobachtung,
  - "gesunden Menschenverstand"
  - nicht auf wissenschaftlicher Basis





# Der Agile Ansatz

- Geisteshaltung durch agiles Manifest populär geworden
- Umgesetzt durch diverse "Agile Methoden"
- Ein wesentlicher Bestandteil: konstantes, schnelles Feedback!

Agile is not  
a thing you buy.

Agile is  
a thing you are.

## Agile Manifest

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Quelle: <http://agilemanifesto.org/>



# Agiles Manifest – Prinzipien 1/2

Our highest priority is to **satisfy the customer** through **early** and continuous delivery of valuable software.

**Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers **must work together daily** throughout the project.

Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of **conveying information** to and within a development team is **face-to-face conversation**.



## Agiles Manifest – Prinzipien 2/2

**Working software** is the primary **measure of progress**.

Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to **technical excellence and good design** enhances agility.

**Simplicity** - the art of maximizing the amount of work not done--is essential.

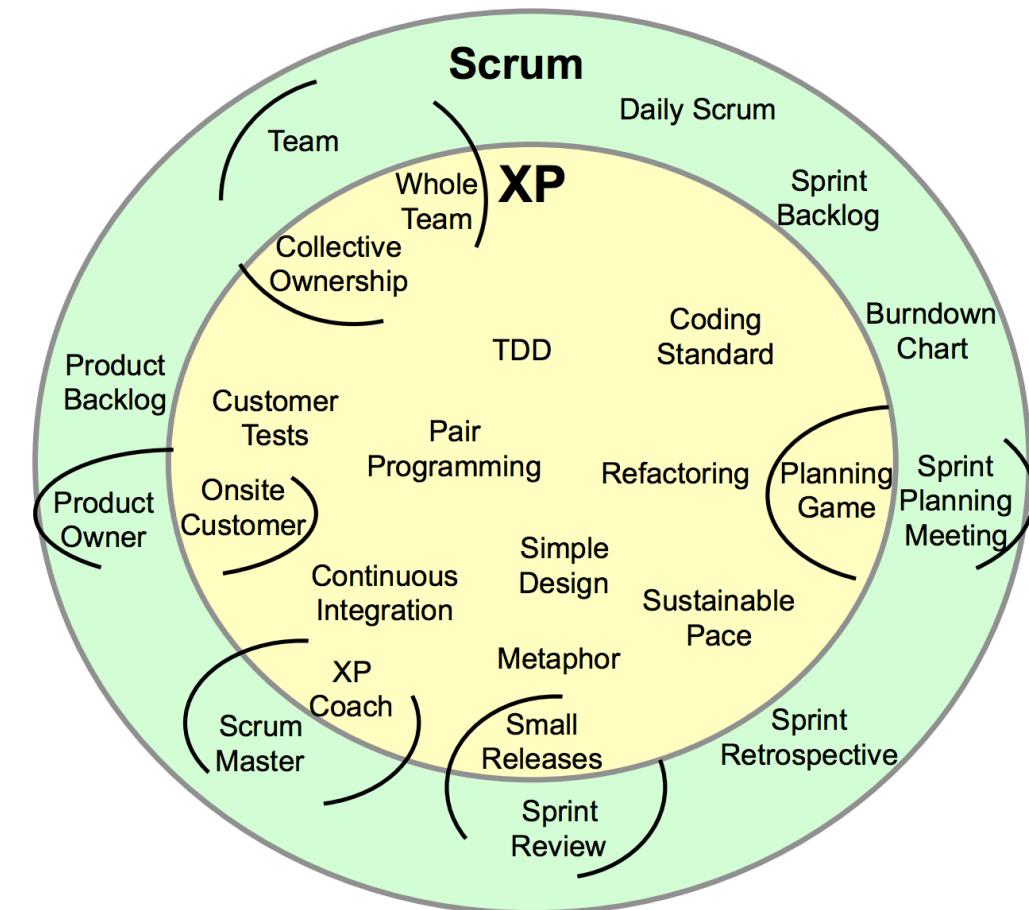
The best **architectures**, requirements, and designs **emerge** from self-organizing teams.

At regular intervals, **the team reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.

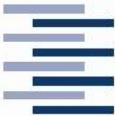


# Agile Methoden

- Die meisten Methoden in den 80ern/90ern entwickelt
- „Agile“ ist ein Oberbegriff für Methoden, die dessen Werte und Prinzipien folgen, z.B.
  - **XP** (Extreme Programming): Fokus auf teaminterne Engineering-Praktiken („Von Programmieren für Programmierer“)
  - **SCRUM**: „Framework“ mit Fokus auf Struktur und Kommunikation („Management“)
- Viele Techniken und Prinzipien sind nicht „neu“!
  - Iterationen → Boehms' Spiralmodell
  - Story Points → Function Points
  - Continuous Integration → „Daily Build“ bei Microsoft (1995)
  - Retrospektive → aus CMMI, KAIZEN (Kontinuierliche Verbesserung)
  - ...

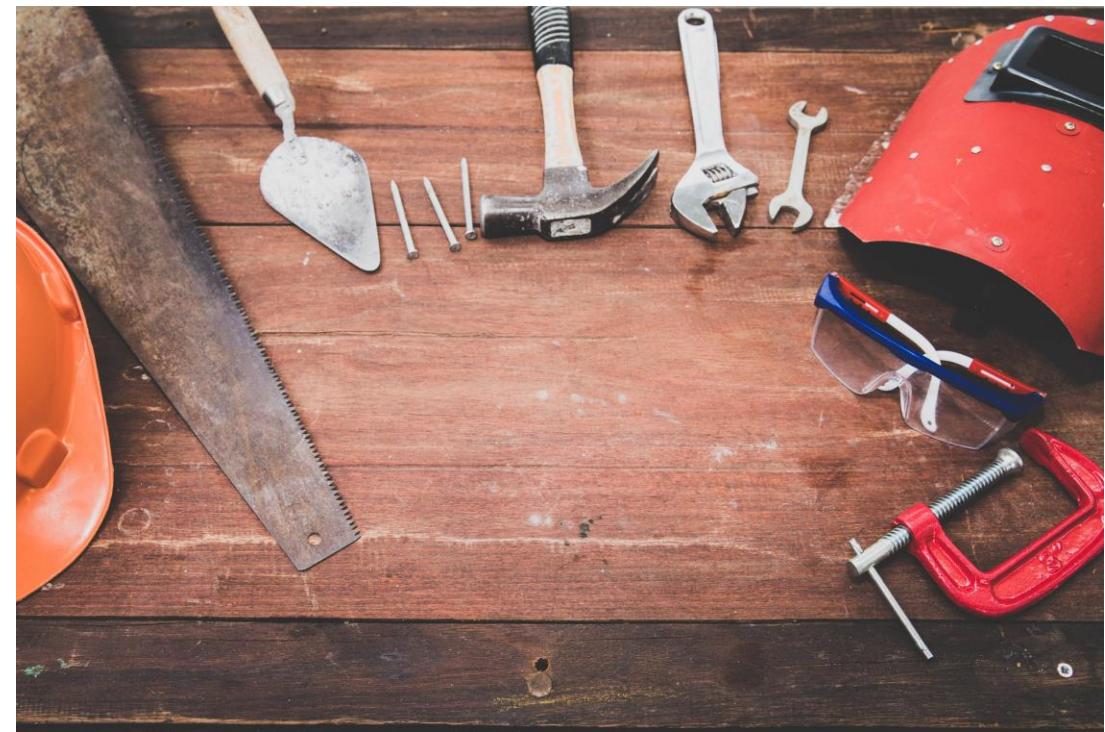


Quelle: Lean from the Trenches, Henrik Kniberg, 2012



# Voraussetzungen für „Agilen Ansatz“

- Verantwortlich arbeiten können nur **hoch qualifizierte Teams**
- **Kleine Teams** mit bis zu 10 Personen (sonst zu viel Kommunikation)
- **Gemeinsamer Raum** für alle Teammitglieder (für agile Techniken unabwendbar!)
- **Teamwork:** Star-Architekten, Gurus und Alleinarbeiter nicht brauchbar
- **Management-Akzeptanz** bei Auftragnehmer und Auftraggeber



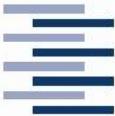


# Agile Softwareentwicklung: Gemeinsame Kernmerkmale

- **Kurze Zyklen bis zur (mehrfachen) Auslieferung**
  - Nach ca. einem Monat ausliefern
  - Dazu Release von Kunden abnehmen und bewerten lassen
- **Evolutionäre Entwicklung**
  - Keine große Architektur, sondern mit kleinster (aber angemessener!) Architektur beginnen und dann Refactoring
- **Testgetriebene Entwicklung**
  - Automatisierte Test vor dem Code schreiben
  - Testen alle Features (User-Stories statt traditioneller Anforderungen)
- **Teamarbeit**
  - Pair Programming, Collective Code Ownership
- **Dokumentation verliert an Wichtigkeit**
  - Nicht "keine Dokumentation"!
  - Kommunikation (mit Kunden und im Team) ersetzt die Dokumentation
- **Kunde als Teil des Teams**
  - Befragbarer, kompetenter und immer verfügbarer Kunde im Team kann Entscheidungen treffen und das Team so Zeit sparen
  - Kunde bestimmt Umfang des Wertes nach seinen Prioritäten
- **Vertrauen**
  - Kunde hat Vertrauen in das Team und das Vorgehen
  - Statt Festpreis: Variabler Preis oder variabler Umfang
- **Änderungen willkommen**
  - Kein Change-Request-Verfahren, sondern Änderungen für neues Release immer möglich



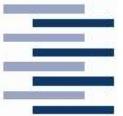
# Kanban



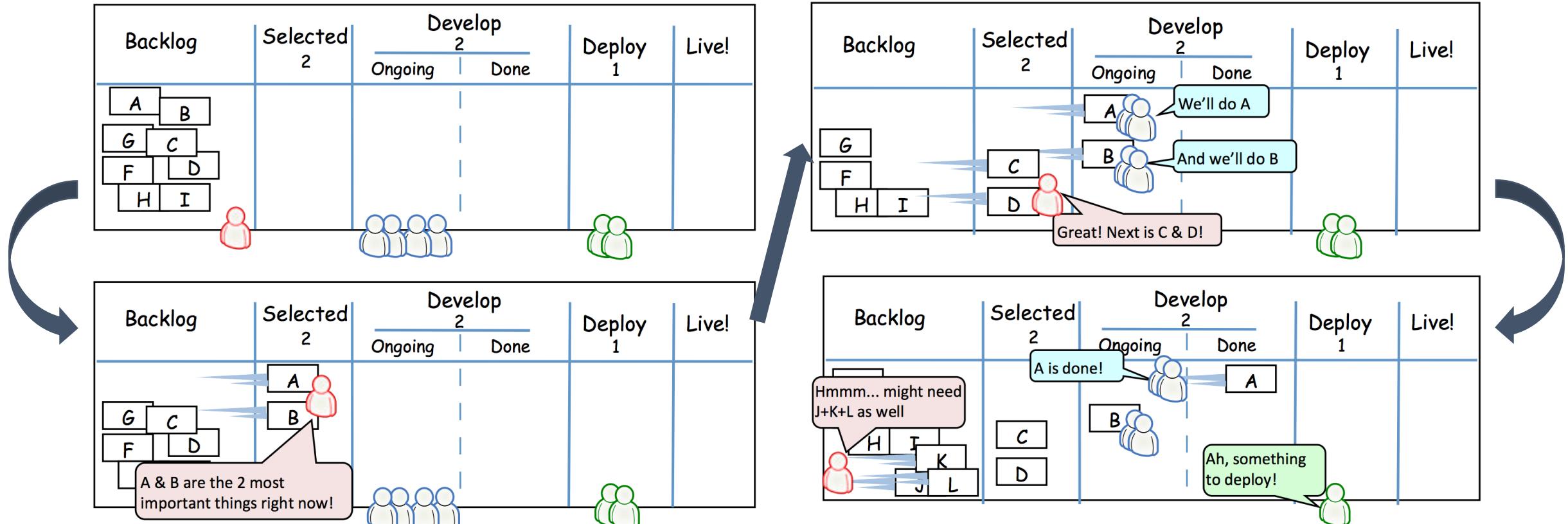
# Kanban in a (mini) Nutshell

- Kanban ist eine Methode zur **Produktionsprozessteuerung**
- Japanisch für „**Signalkarte**“ – reflektiert Einsatz im Produktionsprozess
- Kanban ist alt (1947), Kanban in der IT „neu“
- Kanban möchte den **Fluss optimieren** und Flaschenhälse im Prozess zu erkennen bzw. verringern, der Fokus liegt auf Fluss, nicht auf die Auslastung von Maschinen

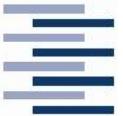




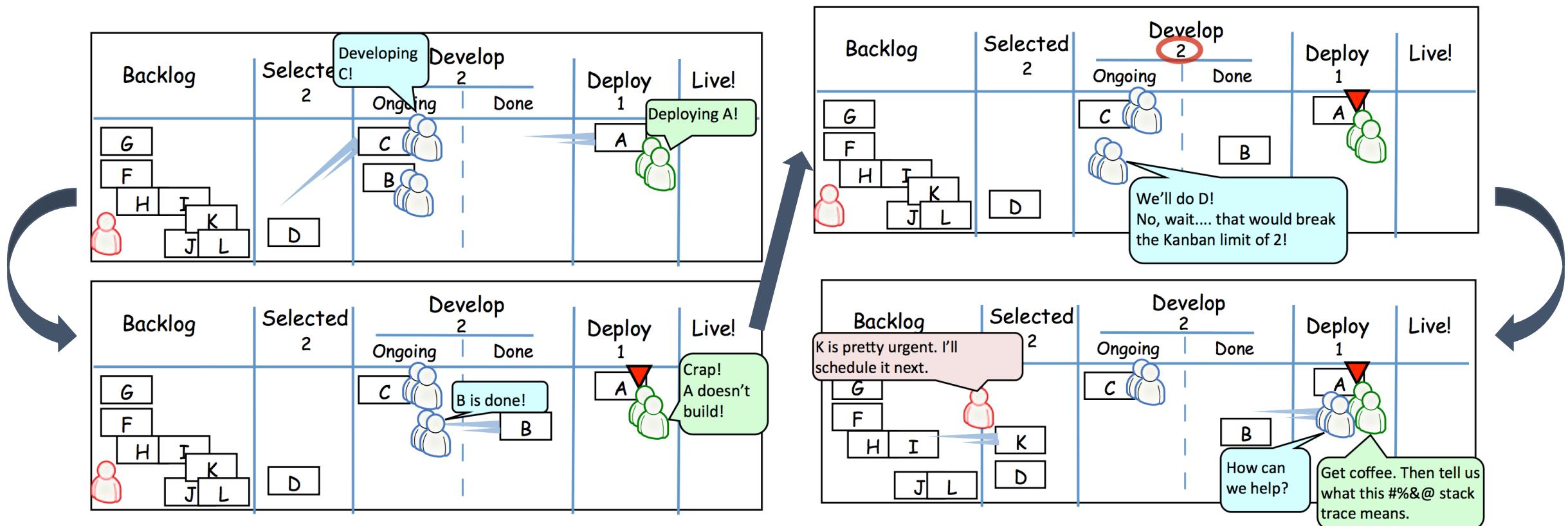
# Ein Tag im Kanban-Land / 1



Quelle: Lean from the Trenches, Henrik Kniberg, 2012



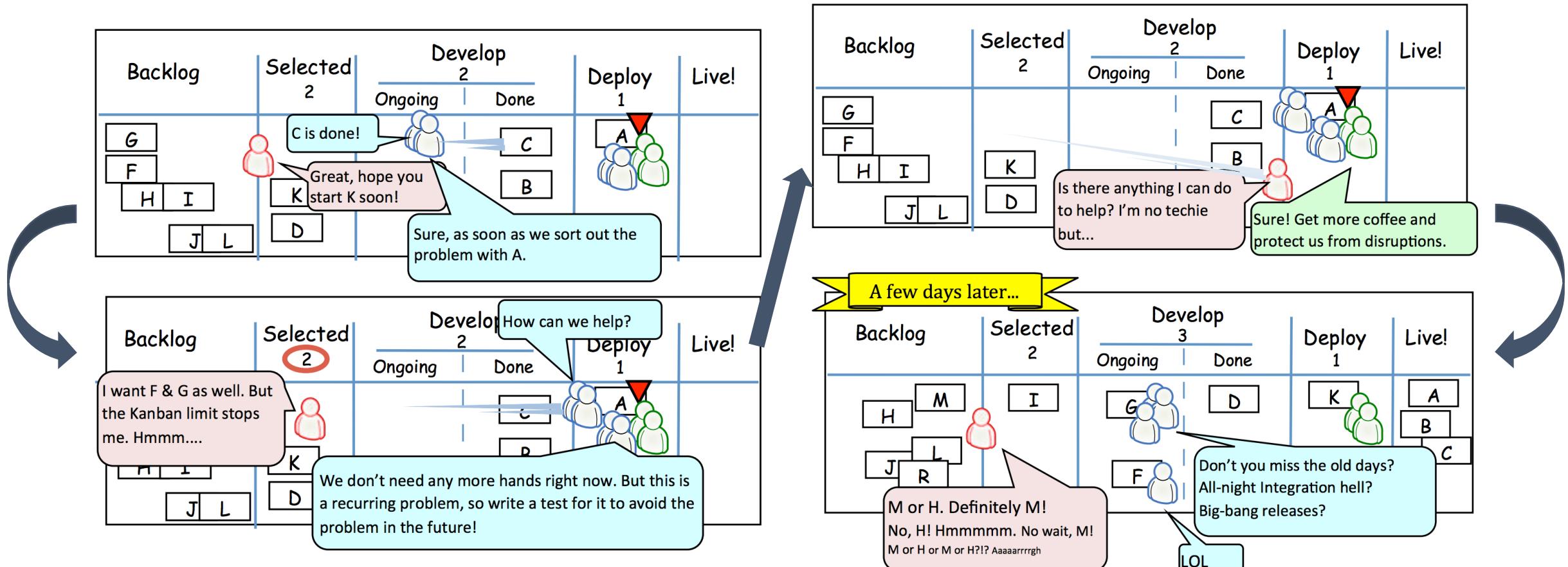
# Ein Tag im Kanban-Land / 2



Quelle: Lean from the Trenches, Henrik Kniberg, 2012



# Ein Tag im Kanban-Land / 3

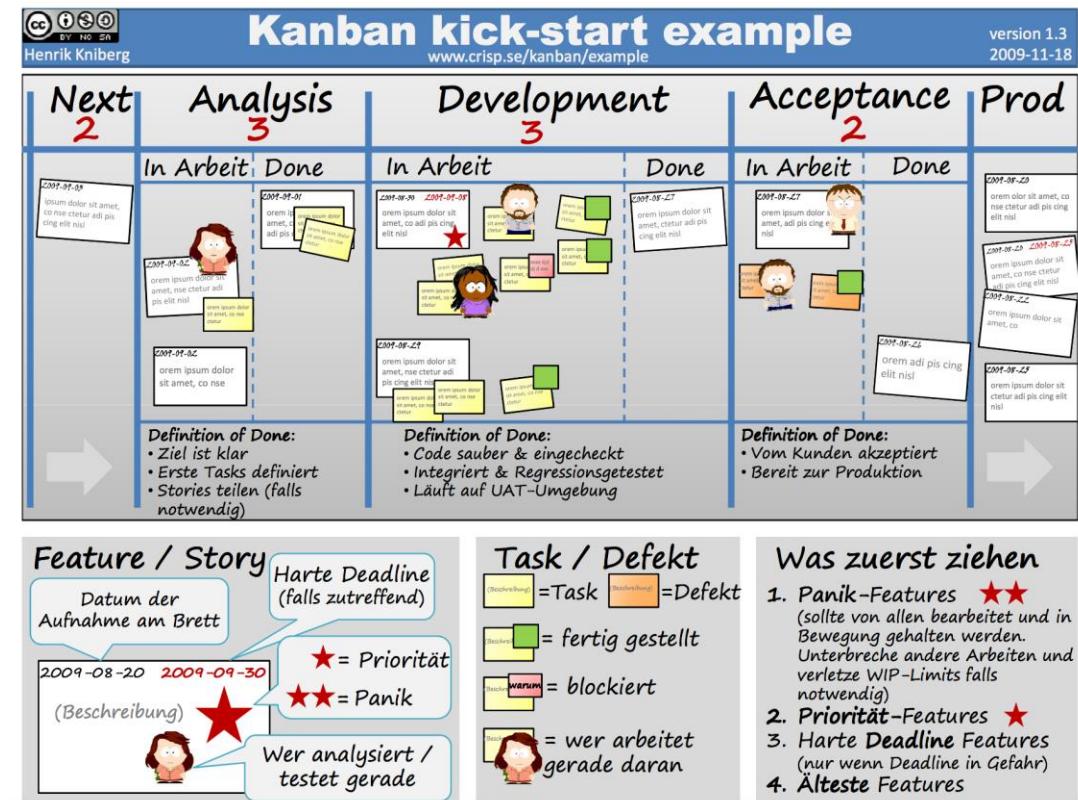


Quelle: Lean from the Trenches, Henrik Kniberg, 2012



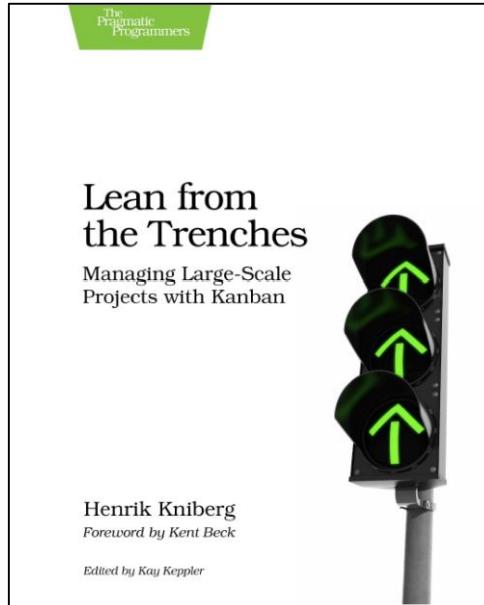
# Kanban - Prinzipien

1. **Workflow visualisieren**  
(wird auch in Scrum gemacht)
2. **Pull-Prinzip:** Aufgaben werden durch Bearbeiter „geholt“, nicht in die nächste Aktivität „geschoben“
3. **„Work in Progress“ („WIP“)** beschränken (hilft, zu fokussieren und Flaschenhälse zu erkennen!)
4. **„Cycle time“** (Durchlaufzeit einer Karte) messen und maximieren – Prozess optimieren, so dass die Cycle time niedrig und vorhersagbar ist
  - = unnötige Lagerzeit vermeiden
  - funktioniert gut bei ähnlich dimensionierten Aufgaben und eingespieltem Team





# Literatur / Referenzen



## Kanban

<http://blog.crisp.se/2014/03/27/henrikkniberg/spotify-engineering-culture-part-1>  
<https://www.crisp.se/file-uploads/kanban-kick-start.pdf>  
<http://limitedwipsociety.ning.com/>

Kanban & Scrum, Einführung, Vergleich

<http://www.infoq.com/minibooks/kanban-scrum-minibook>

Fallstudie: Kanban bei Parship

<http://www.limitedwip.org/case-studies/parship/>



# Lean



# Lean

- Westlicher Begriff für das TPS („Toyota Production System“)
- „Lean“ stammt aus der Produktion, „Agile“ aus der Softwareentwicklung – mit ähnlichen Zielen
- Agile konzentriert sich auf das Produkt – Lean auf den Prozess
- Prinzipien
  - Eliminate waste
  - Keep getting better
  - Decide as late as possible
  - Deliver as fast as possible
  - Empower the team
  - Build quality in
  - Optimize the whole

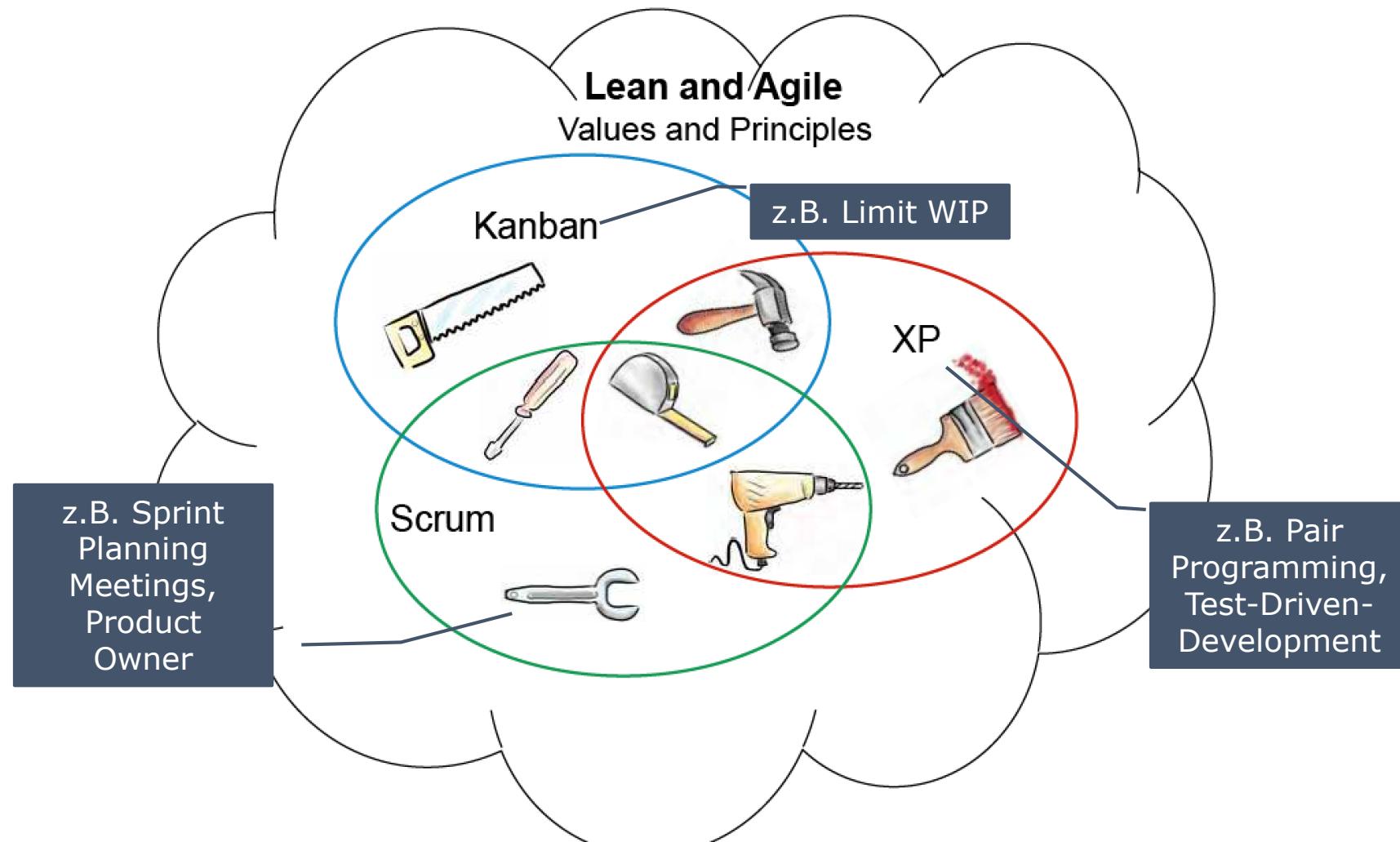
„Während agile Prozesse eine hohe Priorität darauf legen, auf **veränderte Anforderungen reagieren** zu können, was in einer sehr flexiblen Vorgehensweise resultiert, ist das Ziel einer Lean Production, die **Prozesse** möglichst zu **standardisieren** und so weit zu **vereinfachen**, bis nur noch die Elemente übrig bleiben, die wesentlich zur Wertschöpfung beitragen.“

„Tatsächlich ist die **Vermeidung von Verschwendungen eines der zentralen Prinzipien**. Sei es in Form von Überproduktion, hohen Lagerbeständen oder – dem bei Softwareentwicklung wohl schlimmsten Faktor – häufigem Task-Switching.“

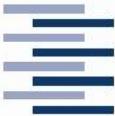
Quelle: <http://t3n.de/news/lean-vs-agil-beiden-ansaetze-667334/>



# Lean, Agile, Scrum, XP, Kanban, ...

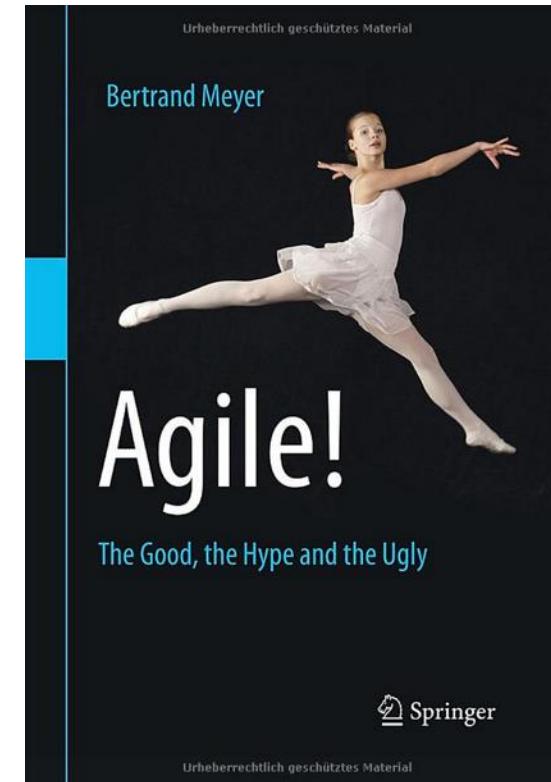


Quelle: Lean from the Trenches, Henrik Kniberg, 2012



# Agile Softwareentwicklung: Bewertung

- Bertrand Meyer bewertet agile Entwicklungsmethodik aus entspannter und wissenschaftlicher Sichtweise
- Definiert Agile Methoden
- „The Good, the Hype and the Ugly“
- 





# Agile Softwareentwicklung: "The Brilliant"

- Kurze Iterationen
- "Closed-window" Regel  
*Beispiel:* Keine Änderung des Sprint-Backlogs während eines Sprints bei SCRUM
- Refactoring  
(jedoch nicht als Ersatz für einen guten Entwurf)
- Für jede Funktionalität gibt es einen Test
- Continuous Integration





# Agile Softwareentwicklung: "The Good"

- Kurze tägliche Treffen
- Fokus auf funktionsfähigem Code
- Tests als ein expliziter Auftrag und Wert im Projekt
- Änderungen werden nicht als Störeffekt betrachtet
- Regressionstest durchführen und analysieren
- übermäßiges Branching vermeiden
- Ein "Product burndown chart"

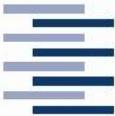




# Agile Softwareentwicklung: "The Indifferent"

- Pair programming
  - Oft falsch verstanden; gemeint ist: Zwei gleich kompetente Programmierer (keine Ausbildungssituation) arbeiten gleichzeitig
  - Manchmal sinnvoll? Ja! Immer sinnvoll? Nein!
  - Jeden dazu zwingen? Nein!
- Ein "Open-space"-Büro
- Selbstorganisierende Teams
  - Nicht jedes Orchester kann ohne Dirigenten arbeiten (fast keines...)
- Verträgliche Arbeitszeiten einhalten
- Erstellung von nur minimaler Funktionalität
- Planning game, planning poker (Aufwände schätzen) – kann nicht jeder
- Jeder im Team kann alles machen

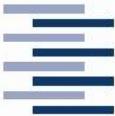




# Agile Softwareentwicklung: "The Ugly" / 1

- Ablehnung von Projekt-"Anfangsaufgaben", insbesondere Anforderungserhebung und Architekturarbeit
- Wissen um gutes Requirements Engineering ignorieren
- Tests statt einer Spezifikation
- Feature-basierende Entwicklung und ignorieren von Abhängigkeiten
  - manche brauchen eine gemeinsame Basis (Architektur)
- "Agil-Experte" (z.B. Scrum Master) als separate Rolle, nicht nur labern, sondern machen!
- „Testgetriebene“ Entwicklung
  - im Sinne von Hauptaufgabe: Test schreiben -> Code schreiben/fixen -> Refactoring -> Repeat
  - Test-First ist dagegen in der Liste der guten Sachen





# Agile Softwareentwicklung: "The Ugly" / 2

- Ablehnung traditioneller Projektleitungsaufgaben
  - Geht für die meisten Teams nicht
  - Viele Entwickler wollen weder "Hilfs-Projektleiter" sein noch einem solchen ausgeliefert sein
- Ablehnung von Hilfs-/Zwischenprodukten und nichtauslieferbaren Produktteilen
- Ablehnung von vorab erstellten Softwarearchitekturen
- Ablehnung von Wiederverwendbarkeit als Entwicklungsziel
- Ablehnung von Erweiterbarkeit als Entwicklungsziel
- Kunde im Team
  - In der Praxis ist Kunde schwierig (Product-Owner ist gut)

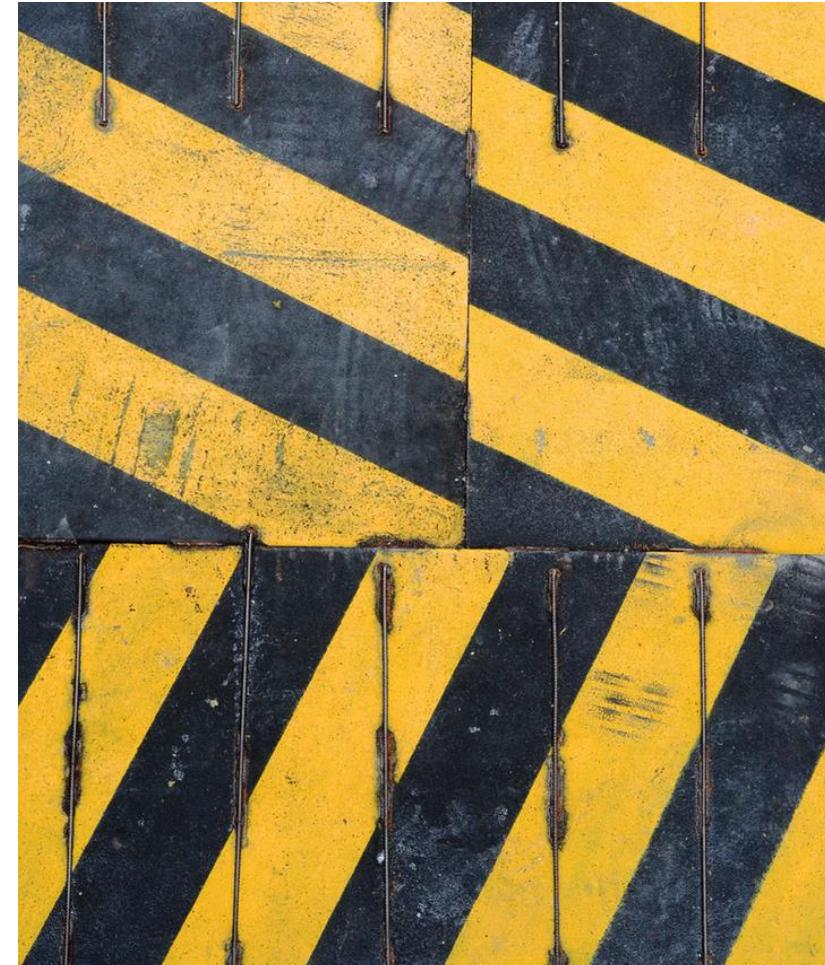




# Ein paar Worte der Vorsicht...

- Seien Sie nicht dogmatisch
  - „Kunde permanent vor Ort“
  - „Daily Scrum“, „Daily Standup“
  - „Feste Zeiträume für Iterationen“
  - „Keine Anforderungsanalyse / Design im Voraus“
  - „In jeder Iteration Software bzw. Software mit ‚direkter‘ Kundenfunktion“
  - ...

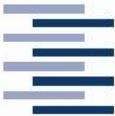
**Passen Sie projektspezifisch an  
und finden Sie „Ihren“ Weg!**





# Agenda

- Motivation und Überblick
- Traditionelle Modelle
- Agile Modelle
- **Auswahl des richtigen Modells**
- Zusammenfassung



# Beispiel-Projekt: Ablösung des Inventur-Systems

- Ein bestehendes **Inventur-System** auf Host-Basis soll durch ein auf **moderner Technologie** basierendes System abgelöst werden
- Die Ablösung erfolgt **1:1**, Kapazitäten für **Endanwender**, die testen und unterstützen sind knapp
- **Hauptgrund** ist das Entwicklerteam, welches in **drei Jahren** in Rente geht und das Wissen zumeist in den Köpfen hat
- Das System hat **30 MJ** Entwicklung benötigt und wird jetzt von einem 3-köpfigen Team betreut. **Größere Änderungen** werden nur noch bei Gesetzesänderungen nötig
- Der Kunde selbst vollzieht gerade eine **agile Transformation**

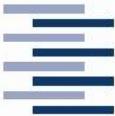




# Übungsaufgabe

- ① Diskutieren Sie welches der dargestellten Vorgehensmodelle Sie dem Kunden vorschlagen würden? Was sind pro und kontra?
- ② 2-er Gruppen
- ③ 10 min.





# Wie findet man das richtige Vorgehen?

- Besinnung auf die wesentlichen Anforderungen
- Agil häufig durch folgende Faktoren motiviert:
  - Schnellere time-to-market
  - Bessere Fähigkeit auf Änderungen zu reagieren (Prioritäten, Scope, Anforderungen)
  - Mehr Kundenbeteiligung
  - Frühzeitige Reduktion von Risiken
  - Bessere Transparenz
  - Agile Organisationskultur
- Diese können aber auf vielen Wegen erreicht werden

Durch Scope-Reduzierung oder höhere Produktivität?

Was ist mit Funktionalitäten, die stark voneinander abhängen?

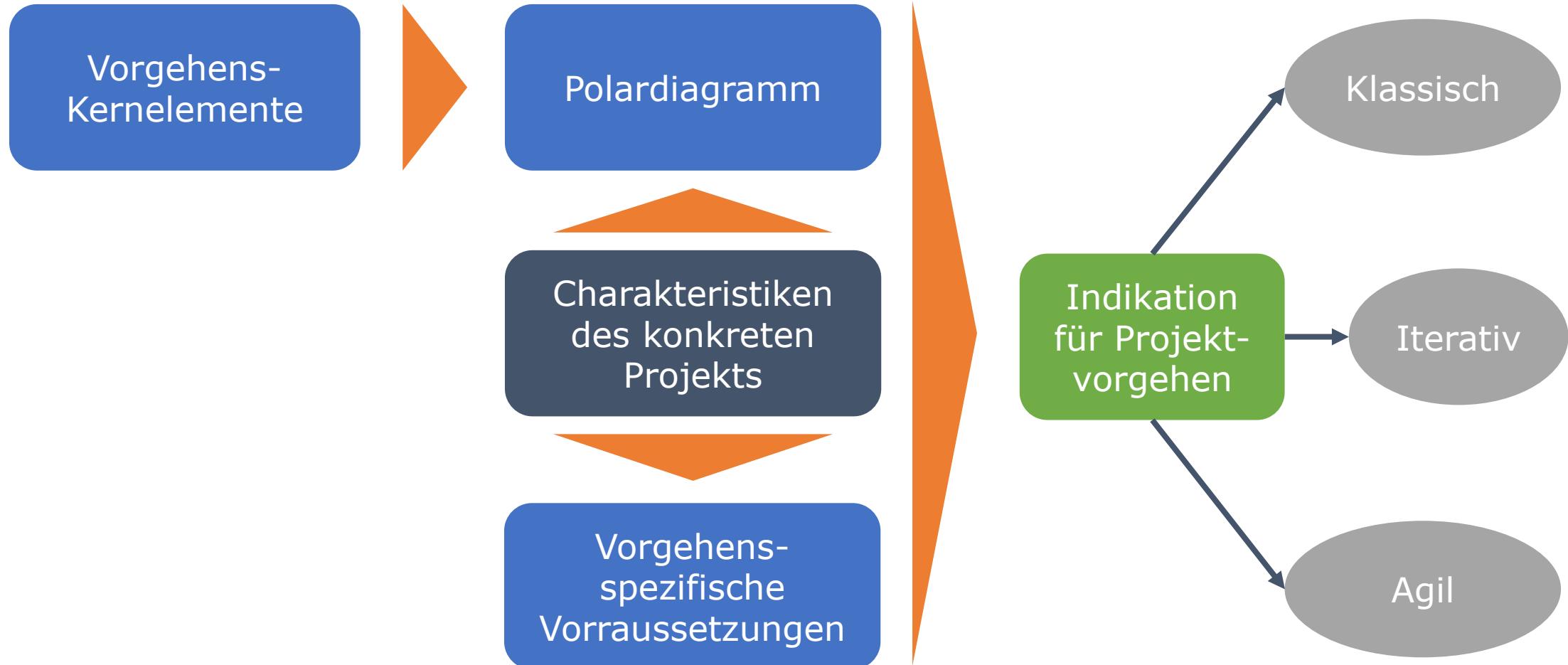
Was, wenn der Kunde keine Zeit hat, oder keine Qualifikation?

Für alle Risiken überhaupt möglich? Architektur?

Was ist wenn die Kultur nicht stabil ist?

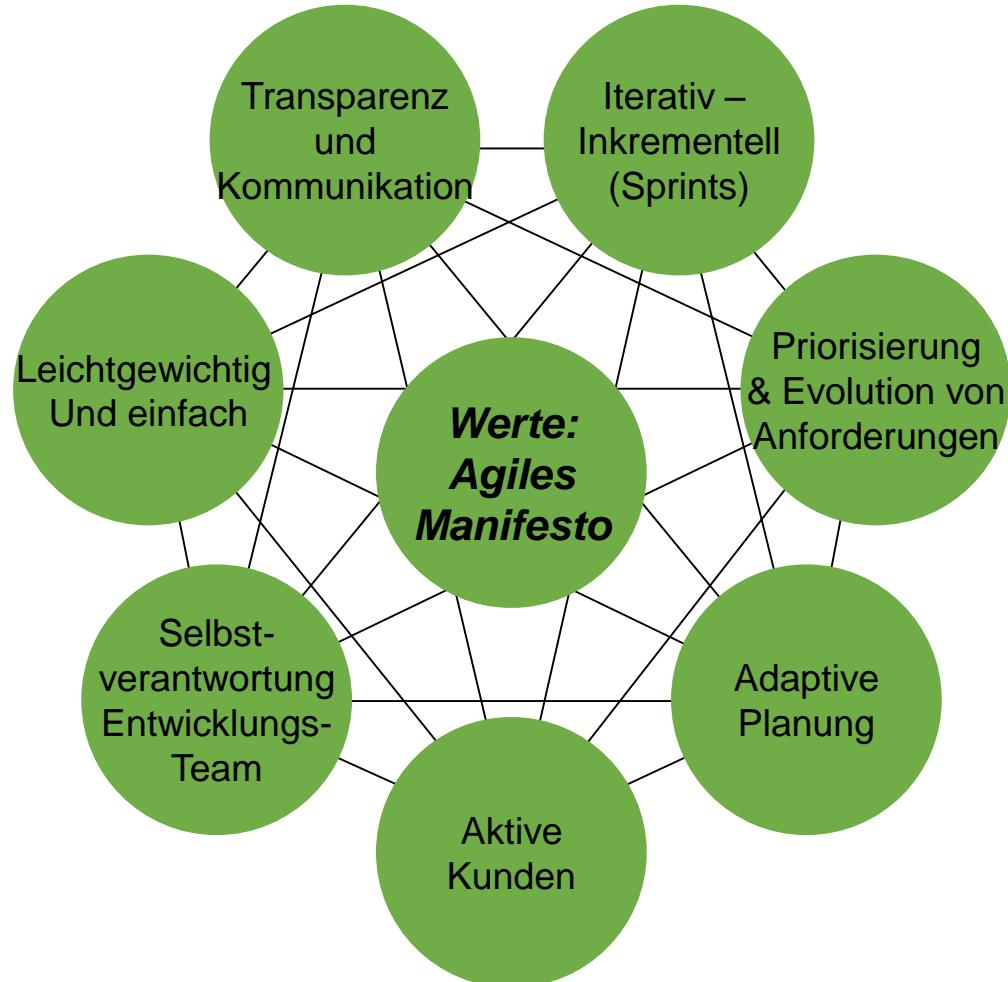


# Beispiel: Capgemini Lifecycle Selection Method





# Kern-Elemente des agilen Vorgehens aus dem Manifesto



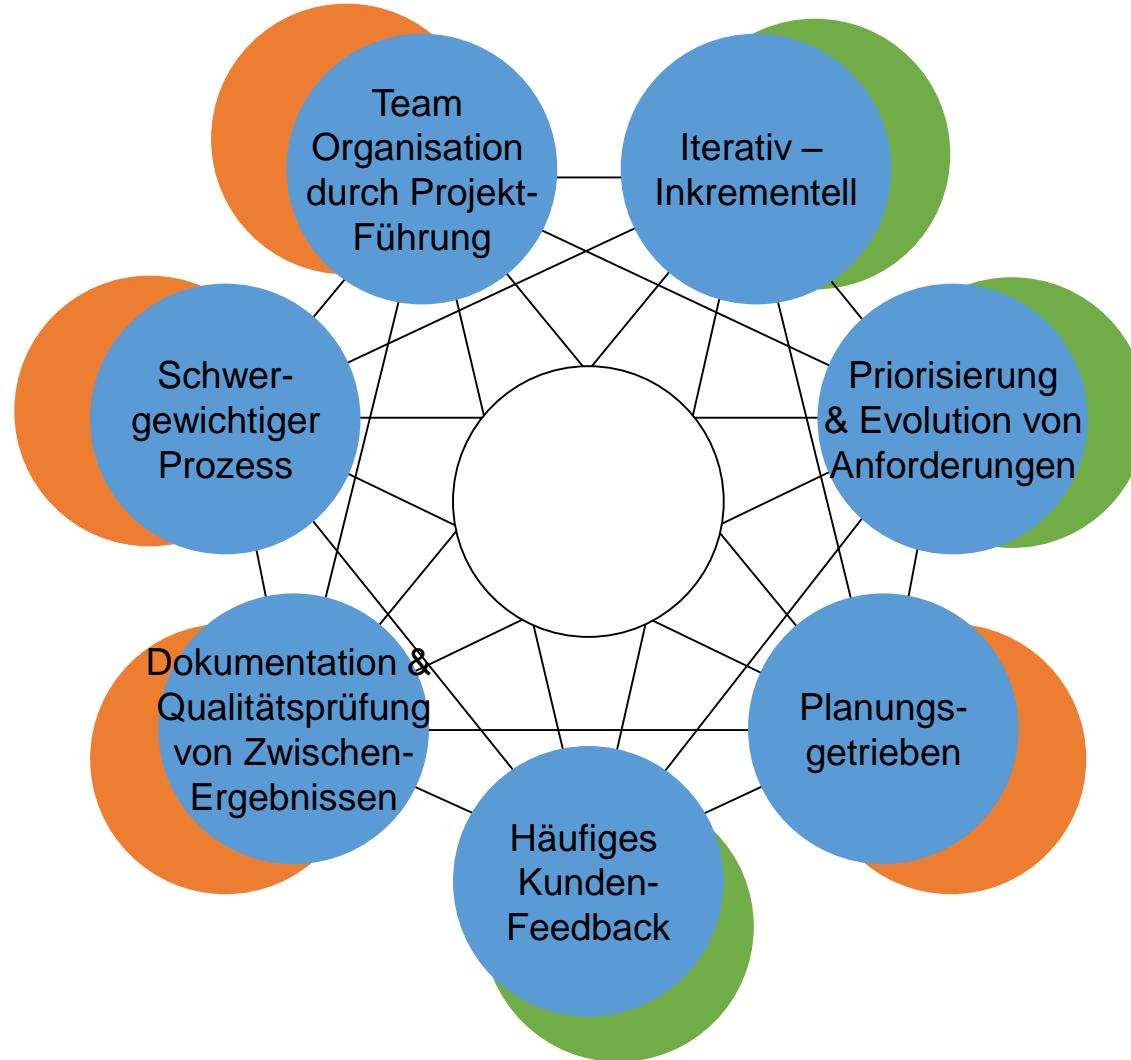


# Kern-Elemente aus dem traditionellen Vorgehen



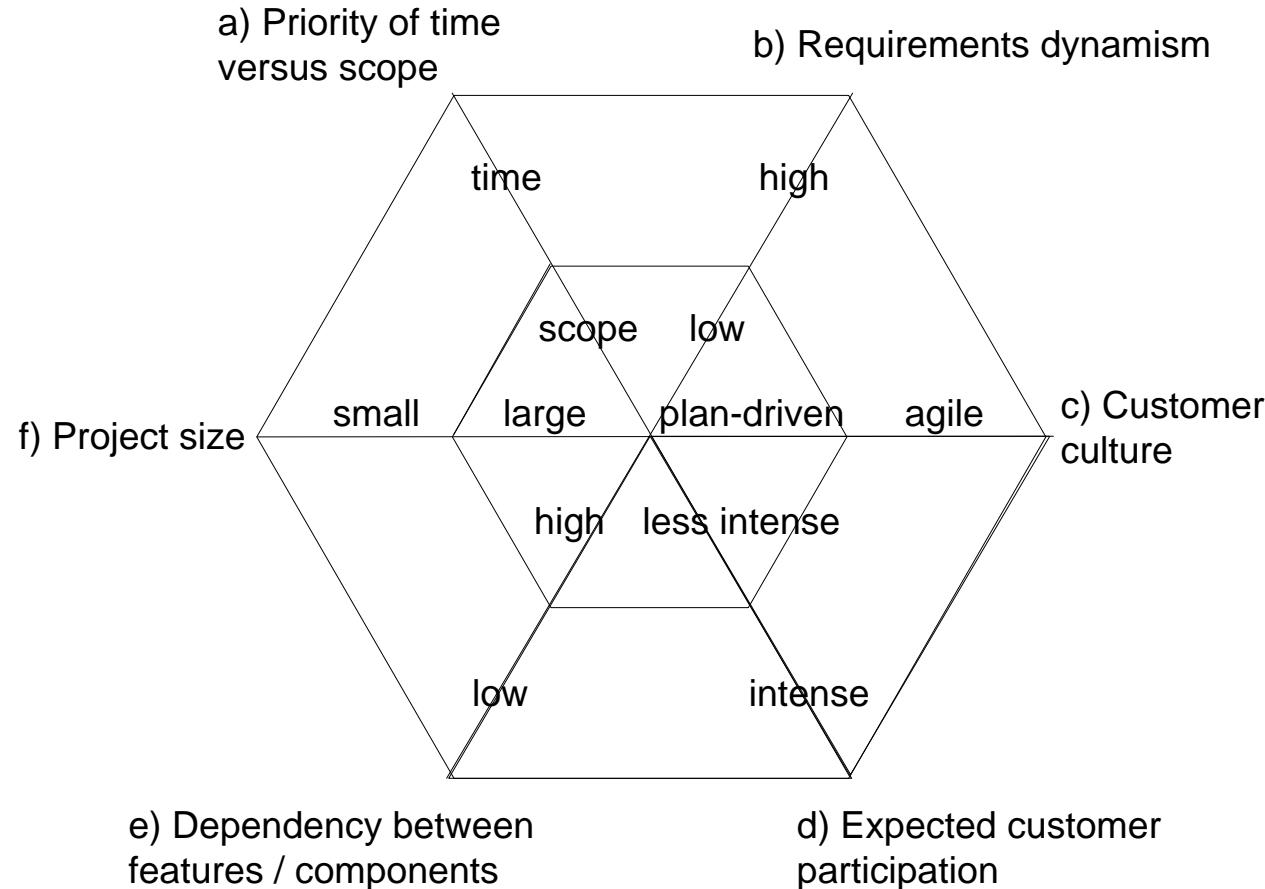


# Kern-Elemente des iterativen Ansatzes liegen zwischen agilem und klassischen Vorgehen





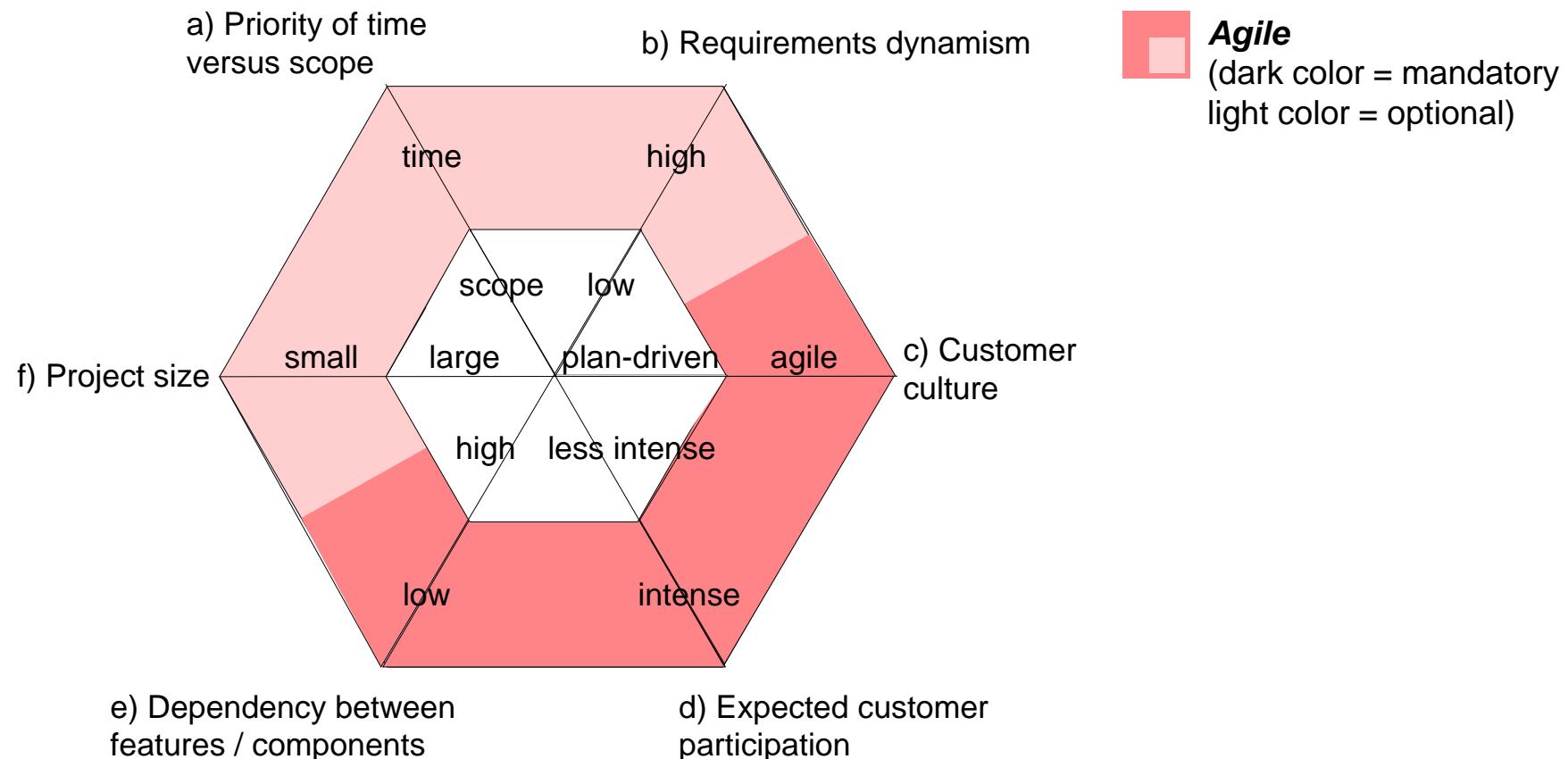
In einem Polardiagramm werden alle gemeinsamen Dimensionen der Modelle abgebildet



Ableitung von  
idealen  
Projekttypen

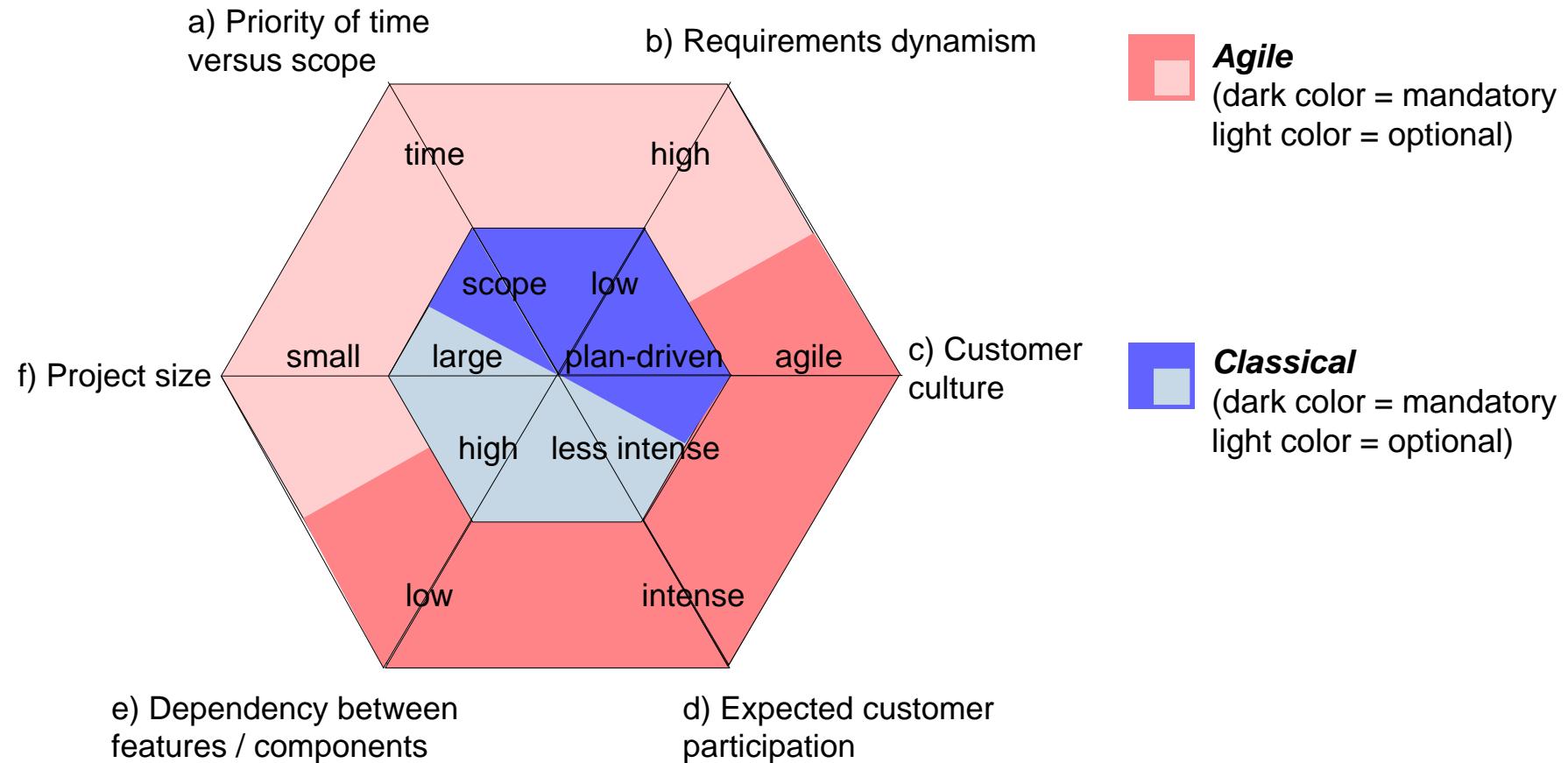


# Der ideale agile Projekttyp liegt im äußeren Kreis



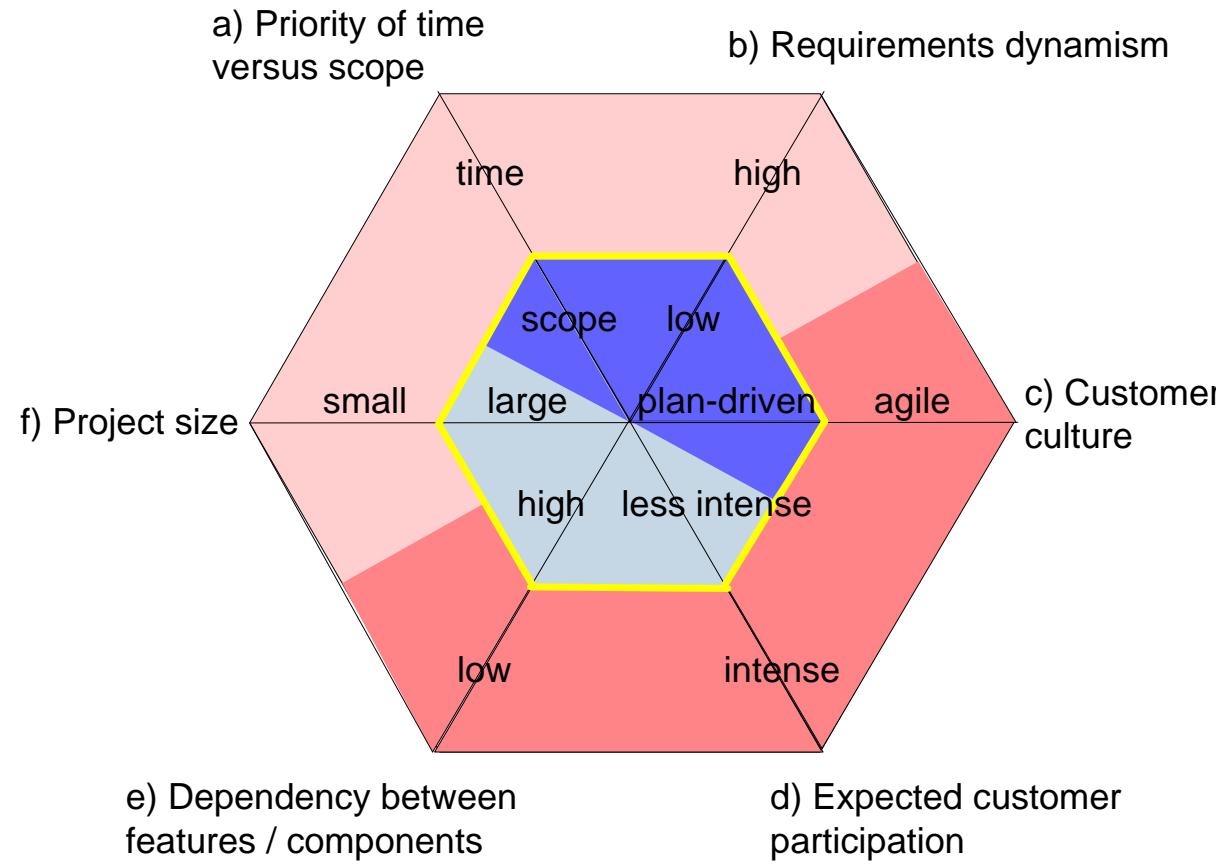


# Der ideale klassische Projekttyp liegt im inneren Kreis





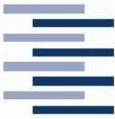
# Der ideale iterative Projekttyp liegt dazwischen



**Agile**  
(dark color = mandatory  
light color = optional)

**Classical**  
(dark color = mandatory  
light color = optional)

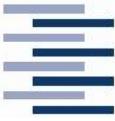
**Medium border**  
(medium ideal iterative  
project type and  
borderline ideal classical  
project type )



# Übungsaufgabe

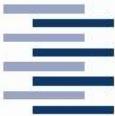
- ① Ordnen Sie ihr SEP2-Projekt in dem Polardiagramm ein und prüfen Sie, ob wir das richtige Vorgehen gewählt haben!
- ② SEP2-Gruppen
- ③ 10min





# Agenda

- Motivation und Überblick
- Traditionelle Modelle
- Agile Modelle
- Auswahl des richtigen Modells
- **Zusammenfassung**



# Zusammenfassung

- Entwickeln sie Software systematisch und **verbessern sie stetig ihr Vorgehen**
- Achten sie darauf, projektspezifisch gut **angepasste Methoden** einzusetzen (manchmal Agil, manchmal nicht)
- **Kommunizieren** sie und betrachten sie Softwareentwicklung nicht nur als technische Aufgabe
- Informatik und SE entwickelt sich weiter: **Am Ball bleiben**
- **Vorsicht** vor „XL“=„Extreme Laberei“, „Certified Geldverdiener-Master“ und Religion statt Wissenschaft

