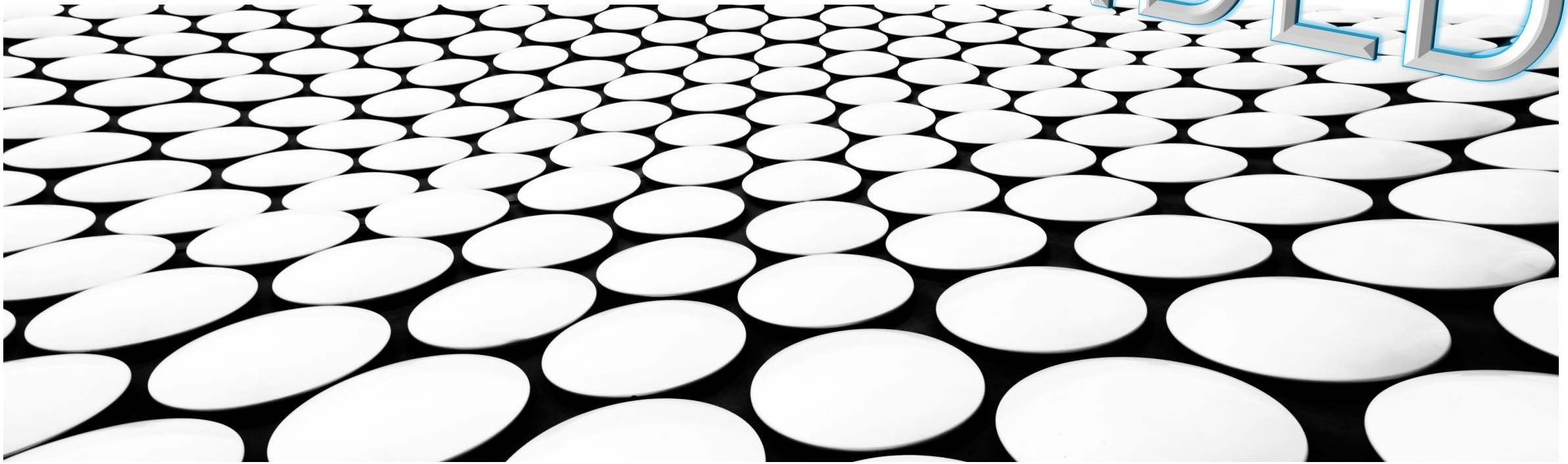
SOFTWARE ENGINEERING EXTENDED

WAS IST SOFTWARE ENGINEERING?



WAS IST SOFTWARE ENGINEERING

- Software Engineering (deutsch: Softwaretechnik) ist die praktische Anwendung wissenschaftlicher Erkenntnisse, um Software wirtschaftlich zu entwickeln, einzusetzen und zu warten.

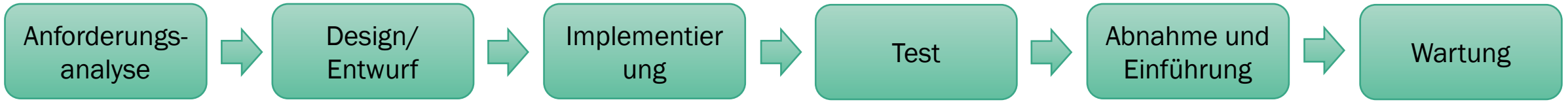
ZIELE VON SOFTWARE ENGINEERING

- Erstellung von Qualitätssoftware
- Fehler vermeiden
- Aufwände minimieren
- Kundennutzen maximieren

SOFTWARE LEBENSZYKLUS

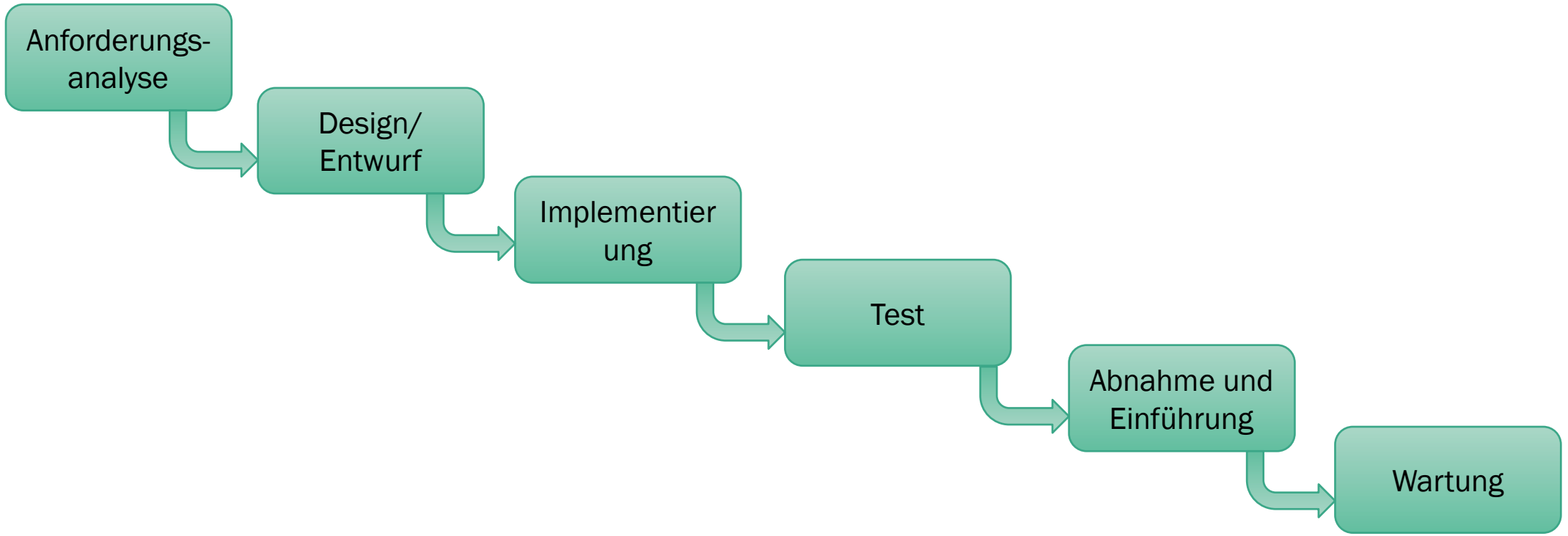
WAS IST DER SOFTWARE LEBENSZYKLUS?

- Jedes Softwaresystem durchläuft einen Software Lebenszyklus
  - Muss bei Softwareentwicklung beachtet werden.



VORGEHENSMODELLE

WAS IST DAS WASSERFALLMODELL



WAS IST DAS WASSERFALLMODELL

- Lineares Modell
- Phasen folgen zeitlich aufeinander
- Können nicht gleichzeitig ablaufen
- Zu Beginn werden Meilensteine definiert, die den Endpunkt einer Phase darstellen
- Endet eine Phase, beginnt die nächste

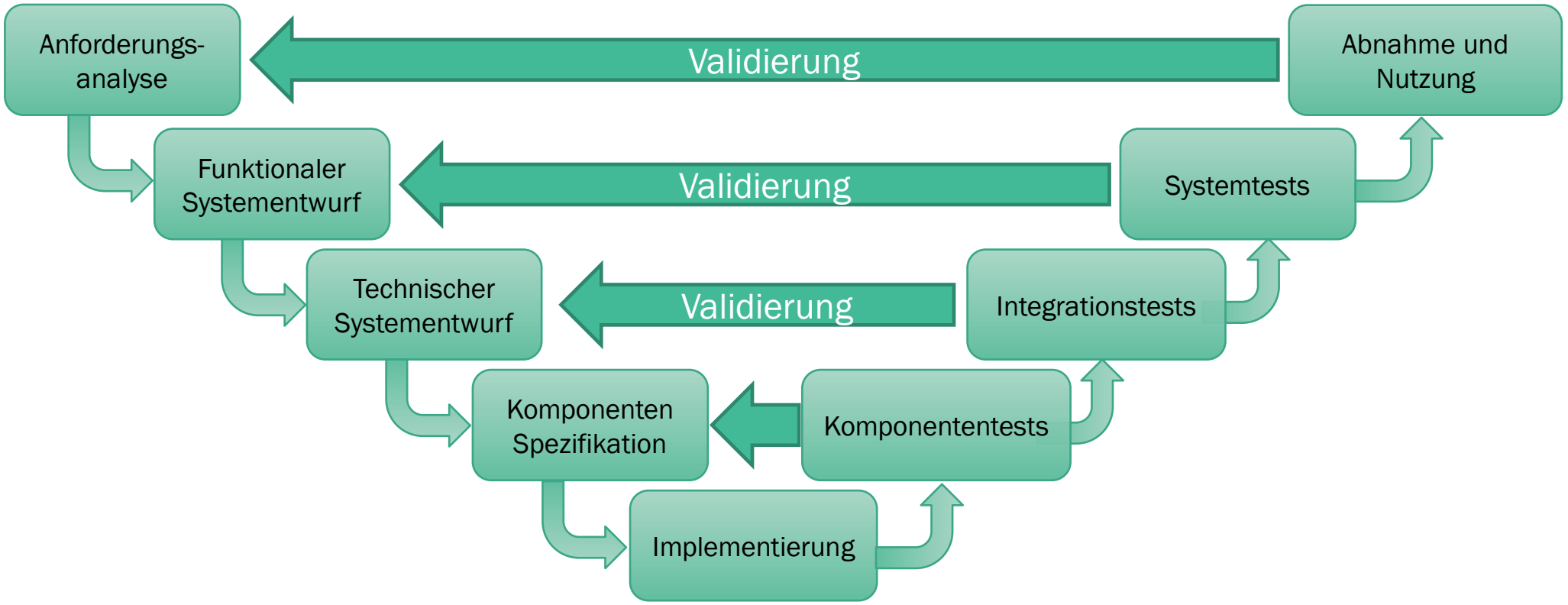
WAS IST DAS WASSERFALLMODELL

Nachteile des Wasserfallmodells

- Klare Abgrenzungen der Phasen in der Realität kaum möglich.
- Keine Flexibilität
- Anforderungen sind fast nie stabil
- In der Realität kaum umsetzbar

DAS V-MODELL

WAS IST DAS V-MODELL



WAS IST DAS V-MODELL

Vorteile des V-Modells

- Hohe Testabdeckung
- In der Konstruktionsphase werden bereits Tests geplant
- Dadurch wird gleich auf Realisierbarkeit geachtet

PROBLEM AN KLASSISCHEN MODELLEN

PROBLEM AN KLASSISCHEN VORGEHENSMODELLEN

- Zwar systematisches Vorgehen, aber in der Praxis meist nicht umsetzbar
  - Große und Umfangreiche Projekte sind schwer vorher komplett durchzuplanen.
  - Modulare Entwicklung kaum möglich (z.B. Entwickeln der Kernfunktionen, um zu testen, ob das Produkt einen Markt hat)

ITERATIVE ENTWICKLUNG

ITERATIVE ENTWICKLUNG

- Planung von mehreren Iterationsschritten
- In jedem Iterationsschritt werden alle Phasen durchlaufen
  - Analyse, Entwurf, Implementierung und Test
- Im folgenden Iterationsschritt werden Mängel aus vorherigem Schritt behoben

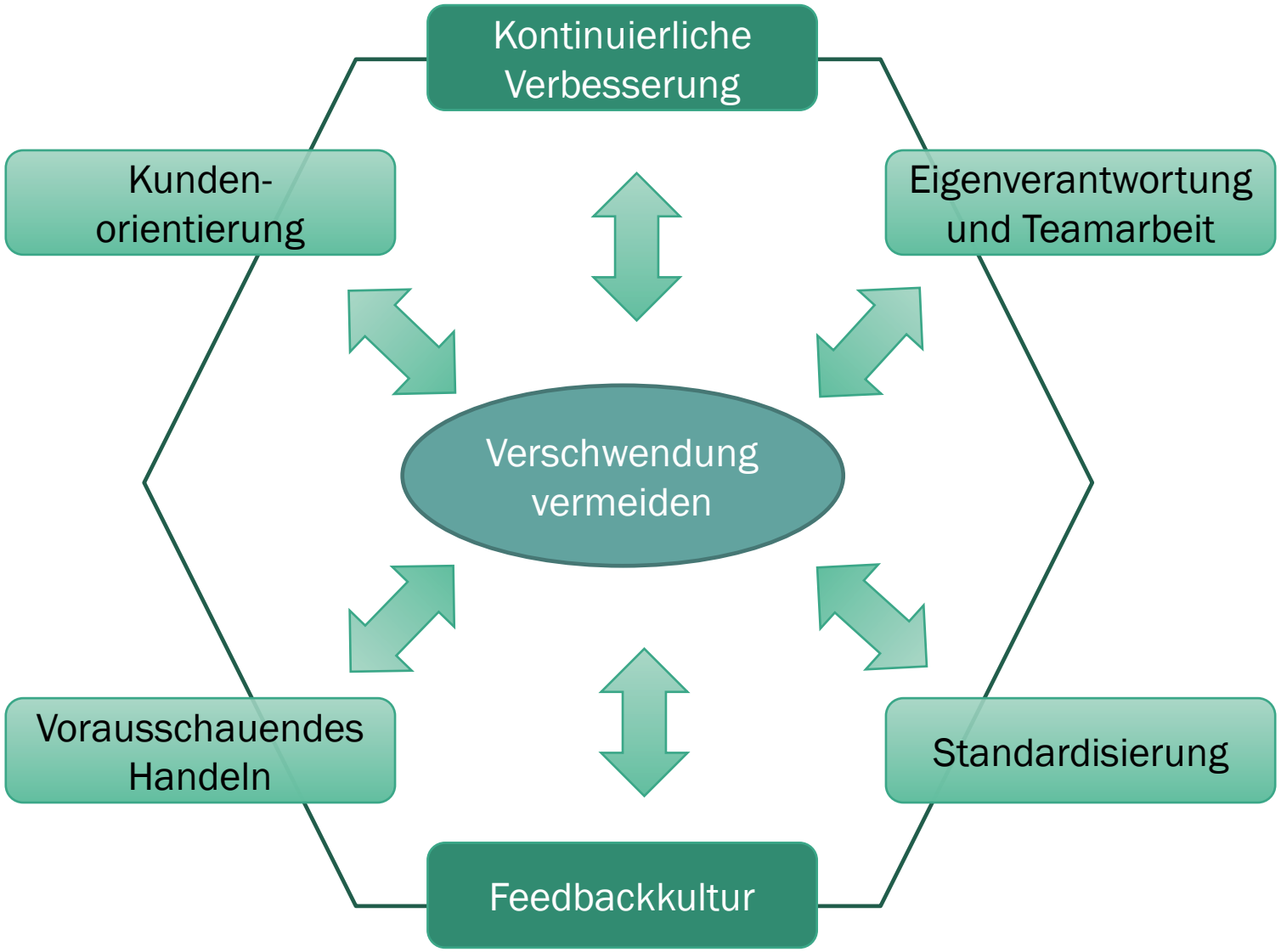
INKREMENTELLE ENTWICKLUNG

- Gesamtsystem bleibt im Gesamtumfang offen
- Wird Schritt für Schritt in Ausbaustufen realisiert(Inkremente)

LEAN MANAGEMENT

LEAN MANAGEMENT

Lean Management umfasst die Methoden und Verfahrensweisen zur Gestaltung von Wertschöpfungsketten durch Vermeidung von Verschwendung. Der Lean-Ansatz in der Managementphilosophie ist unabhängig von der Art des Produktes. Ebenso ist es auf alle Produktionsabläufe und Unternehmensbereiche anwendbar.



LEAN MANAGEMENT

- Der Begriff Lean Management (dt. schlankes Management) beschreibt eine bestimmte Zielsetzung im unternehmerischen Umfeld. “Effiziente Gestaltung der gesamten Wertschöpfungskette durch Vermeidung von Verschwendung, unabhängig von der Art des Produktes (Güter oder Dienstleistungen), den Produktionsabläufen oder den Unternehmensbereichen”. Diese Philosophie wird auf das gesamte Unternehmen angewandt. Nur durch eine umfassende Ausrichtung anhand dieser Maßstäbe kann das Lean Management erfolgreich umgesetzt werden.

LEAN MANAGEMENT

- Der Ursprung der Lean-Ansätze (z.B. auch Lean Administration oder Lean Production) geht auf das Toyota Produktionssystem zurück. Während einer Studie konnte in den Produktionsstätten des japanischen Autobauers eine höhere Qualität und Effizienz als bei den amerikanischen oder europäischen Mitbewerben festgestellt werden. Daraus ergab sich ein enormes Potenzial zur Optimierung innerhalb der gesamten Industrie.
- Im Zuge dieser Erkenntnisse wurden entscheidende Ansätze dieses Systems adaptiert. Die Lean-Philosophie findet bis heute Anklang. Heutzutage wird sie branchen-übergreifend auf der ganzen Welt verwendet. Die Grundidee lässt sich anhand von 7 Prinzipien zusammenfassen.

DIE PRINZIPIEN DES LEAN MANAGEMENT

- **Kundenorientierung**: Prozesse und Abläufe jeglicher Art sind auf die Erfüllung der Anforderungen des Kunden ausgerichtet. Das Ziel soll “aus Sicht des Kunden” gesetzt werden.
- **Kontinuierliche Verbesserung (continuous improvement)**: Die Erarbeitung beständiger, schrittweiser und konkreter Verbesserungsvorschläge durch Einbeziehung aller Mitarbeiter. Unterschiedliches Know How sorgt für einen vielfältigen Input.
- **Eigenverantwortung und Teamarbeit**: Aufgaben werden im Team erledigt und die Ausführung der Tätigkeiten erfolgt in Eigenverantwortung. Die Mitarbeiter werden durch die vom Unternehmen eingeführten Standards für die jeweiligen Bereiche unterstützt.
- **Standardisierung**: Es erfolgt eine Standardisierung von sich wiederholenden Tätigkeiten. Jeder Mitarbeiter hat sich an diese zu halten. Vorschläge zur Weiterentwicklung der Standards sind expliziter Teil der Lean-Kultur im Unternehmen.

DIE PRINZIPIEN DES LEAN MANAGEMENT

- **Feedback**: Unterstützung und Wertschätzung der Mitarbeiter durch direktes Feedback. Die Rückmeldung erfolgt mit Hilfe von objektiven Zahlen, Daten und Fakten. Zusätzlich werden subjektive Meinungen und Reaktionen in den Prozess mit einbezogen.
- **Vorrausschauendes Handeln**: Ausrichtung von Entscheidungen am langfristigen Erfolg des Unternehmens. (nicht “just in time” an der aktuellen Situation)
- **Verschwendung eliminieren**: Optimierung der Prozesse im gesamten Unternehmen durch Einsatz von Methoden, Denkweisen und Werkzeugen. Vermeidung sämtlicher Arten der Verschwendung.

VORTEILE VON LEAN MANAGEMENT

Die wesentliche Methode des Lean Management ist, eine neue Unternehmenskultur zu verinnerlichen. In dieser Kultur sollen Fehler nicht verurteilen oder gar bestrafen werden. Man betrachtet Fehler und Probleme als wertvolle Möglichkeit zur Weiterentwicklung und zur Verbesserung des eigenen Unternehmens.

Gleichzeitig werden durch Lean Management und den damit verbundenen Methoden die Prozesse entlang der gesamten Wertschöpfungskette (engl. Supply Chain) verbessert. Dies geschieht im Wesentlichen durch das Identifizieren und eliminieren der angesprochenen Verschwendung.

FLEXIBILITÄT ALS ENTSCHEIDENDER FAKTOR

Die Mitarbeiter werden beim Lean-Konzept als Menschen fokussiert betrachtet. Das bedeutet, dass sie ihre Einstellung selbst reflektieren müssen, um so Platz für eine neue Einstellung gegenüber ihrer eigenen Arbeit einzunehmen.

Der Lean-Gedanke kann nur dann dauerhafte Erfolge bringen, wenn es dem Management gelingt, das Verhalten der Mitarbeiter positiv zu beeinflussen. Neue, flexible Verhaltensweisen müssen im Unternehmen verankernd werden, um einen langfristigen Erfolg der Umstellung zu gewährleisten. Lean Production (schlanke Produktion) kann dabei als Umsetzungswerkzeug dienen.

Diese neue Philosophie, die im Wesentlichen auf dem Lösen von Problemen und der kontinuierlichen Verbesserung basiert, kann aus der Anwendung von Lean-Methoden eine wichtige Stütze der Kundenorientierung entstehen lassen. Durch die Flexibilisierung und die dadurch resultierende Steigerung des Outputs, kann sie somit letztendlich zum andauernden Erfolg eines Unternehmens beitragen.

FLEXIBILITÄT ALS ENTSCHEIDENDER FAKTOR

Im Vordergrund steht die Optimierung aller wertschöpfenden Prozesse durch den Einsatz von Lean-Methoden, Denkweisen und Werkzeugen. Kundenorientierung und Kostensenkung sind die zwei wichtigsten Aspekte im Lean Konzept. Diese sollen durch den Einsatz einer prozessorientierten Unternehmensführung und Gestaltung eindeutig definierter Prozesse erreicht werden.

Es ist wichtig hierbei zu beachten, dass das Aufsetzen und Implementieren einer völlig neuen Unternehmensphilosophie ein großer Schritt ist. Dies bringt allerlei Aufwand und Disziplin mit sich. Auf lange Sicht gesehen, bringt dieser Schritt jedoch eine Reihe an Vorteilen mit sich. Diese können entscheidend für Erfolg und Misserfolg eines Unternehmens sein.

Lean Management bietet somit durch seine Philosophie und Methodik die Möglichkeit, sich auf volatile Marktanforderungen bestens einzustellen. Die oben genannten Prinzipien bilden eine praktische Guideline, um die eigenen Prozesse zu optimieren. Bei konsequenter Umsetzung kann sich so eine Verbesserung in sämtlichen Arbeitsbereichen erzielen lassen.

DAS AGILE MANIFEST

DAS AGILE MANIFEST

- Agile Vorgehensmodelle beruhen auf Agilen Manifest
- Im Jahr 2001 von einer Gruppe von IT-Spezialisten verfasst
- 4 Leitsätze
- 12 Prinzipien

DIE 4 LEITSÄTZE DES AGILEN MANIFEST

- Quelle: https://agilemanifesto.org/iso/de/manifesto.html

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

DIE 12 PRINZIPIEN DES AGILEN MANIFEST

- Unsere höchste Priorität ist es,
  den Kunden durch frühe und kontinuierliche Auslieferung
  wertvoller Software zufrieden zu stellen.
- Heiße Anforderungsänderungen selbst spät
  in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen
  zum Wettbewerbsvorteil des Kunden.
- Liefere funktionierende Software
  regelmäßig innerhalb weniger Wochen oder Monate und
  bevorzuge dabei die kürzere Zeitspanne.

DIE 12 PRINZIPIEN DES AGILEN MANIFEST

- Fachexperten und Entwickler
  müssen während des Projektes
  täglich zusammenarbeiten.
- Errichte Projekte rund um motivierte Individuen.
  Gib ihnen das Umfeld und die Unterstützung, die sie benötigen
  und vertraue darauf, dass sie die Aufgabe erledigen.
- Die effizienteste und effektivste Methode, Informationen
  an und innerhalb eines Entwicklungsteams zu übermitteln,
  ist im Gespräch von Angesicht zu Angesicht.

DIE 12 PRINZIPIEN DES AGILEN MANIFEST

- Funktionierende Software ist das wichtigste Fortschrittsmaß.
- Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
- Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.

DIE 12 PRINZIPIEN DES AGILEN MANIFEST

- Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.
- Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
- In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

AGILE VS. LEAN

DIE UNTERSCHIEDE

AGILE VS. LEAN

Agile konzentriert sich auf das Produkt – Lean auf den Prozess

Alleine hier fallen schon einige Parallelen zu agilen Methoden ins Auge. Auch agile Teams arbeiten selbstorganisierend, das heißt sie entscheiden selbst wie sie ihre Aufgaben erfüllen. Weil Software häufig, in möglichst kurzen Intervallen, ausgeliefert werden soll, bietet sich für agile Teams auch regelmäßig die Gelegenheit zu Feedback – sowohl zum Produkt als auch zum Prozess.

Andererseits finden sich hier schon die ersten Unterschiede.

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.

AGILE VS. LEAN

Zwar ist das Ziel beider Ansätze die Zufriedenstellung des Kunden. Sie unterscheiden sich aber wesentlich im Ansatz: Konzentriert sich Agile auf das Produkt, so konzentriert sich Lean eher auf den Prozess.

Diese Tatsache lässt sich auch kulturell erklären. Der Ursprung der Lean-Prinzipien liegt im Produktionssystem von Toyota, einem japanischen Unternehmen. Gerade die japanische Kultur ist, wie einige andere asiatische Kulturen auch, von einer großen Hochachtung gegenüber der Perfektion von Prozessen geprägt. Der Sushi-Meister, der seinen Lehrling erst den Fisch schneiden lässt, nachdem es jener perfekt versteht den Reis zu rühren mag ein Klischee sein, es illustriert aber treffend, wie die Fokussierung auf das Meistern bestimmter Handgriffe zu einer besseren Befriedigung der Kundenwünsche führt.

Im Kontrast dazu stehen westliche Kulturen, die sich wesentlich auf das fertige Produkt konzentrieren und den Kunden dadurch besser bedienen, dass sie ständig versuchen das Produkt zu optimieren.

VORGEHENSMODELL SCRUM

WAS IST SCRUM

- Agile Methode der Softwareentwicklung
- Konzept von Scrum wird auch außerhalb des Kontextes der Softwareentwicklung angewendet(z.B. Projektmanagement)
- Übersetzt: Gedränge
  - Ähnlich wie in einem Rugby-Team treffen sich alle Scrum-Mitglieder täglich, um sich gegenseitig über den Stand der Dinge zu informieren und abzustimmen

WAS IST SCRUM

- Ansatz: empirisch, inkrementell, iterativ
- Ziel: Kostengünstige Entwicklung hochwertiger Produkte
- Von Ken Schwaber und Jeff Sutherland entwickelt

WAS IST SCRUM

- **Idee:** In kurzen Zyklen releasefähige Software auszuliefern
- Man arbeitet innerhalb sogenannter Sprints
  - Zeitspanne: Zwei bis vier Wochen
- **Ergebnis:** Fertiges Inkrement der Software

BESTANDTEILE VON SCRUM

- **Rollen:**
  - Product Owner
  - Scrum Master
  - Entwicklungsteam
- **Ereignisse:**
  - Sprint-Planning
  - Daily-Scrum
  - Sprint-Review
  - Retrospektive
- **Artefakte:**
  - Product Backlog
  - Sprint Backlog
  - Produkt-Inkrement

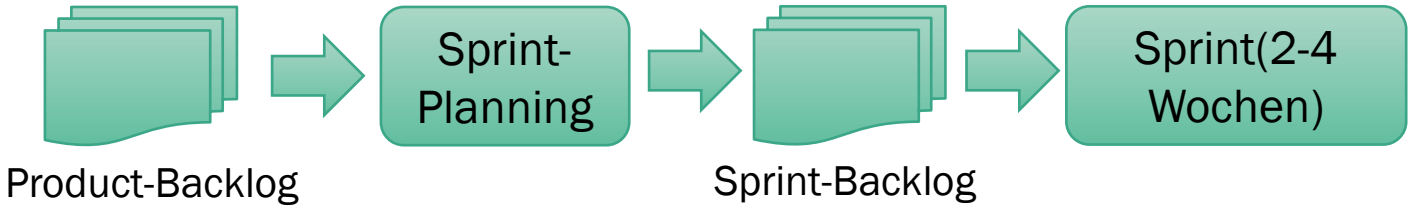
Daneben gibt es noch weitere Rollen wie Anwender, Stakeholder und Management, die in aller Regel aber nicht fester Bestandteil des Scrum-Teams sind.

SCRUM – DER PROZESS

- **Product-Backlog:** Dokument aller Anforderungen an die Software
- Sprint-Planning: Das ganze Scrum Team plant die Arbeit für den kommenden Sprint
- Ergebnis: Sprint-Backlog mit Tasks

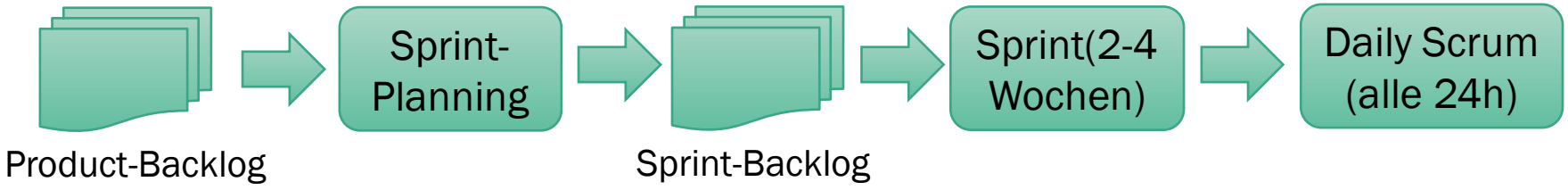
SCRUM – DER PROZESS

- **Sprint:** Innerhalb des Sprints werden alle Tasks ausgeführt
- **Wichtig:** Sprints sind zeitlich begrenzt
- Keine Änderungen oder neue Anforderungen während eines Sprints, sondern erst wieder im nächsten Sprint-Planning



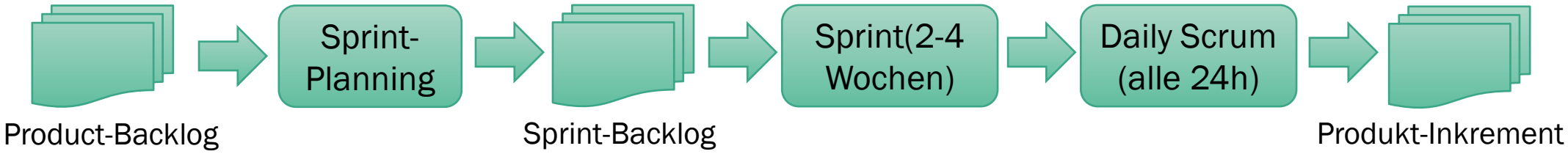
SCRUM – DER PROZESS

- **Daily-Scrum:**
- Jedes Teammitglied beantwortet drei Fragen:
  - Was habe ich gestern alles geschafft?
  - Was werde ich heute tun?
  - Was behindert mich bei meiner täglichen Arbeit?



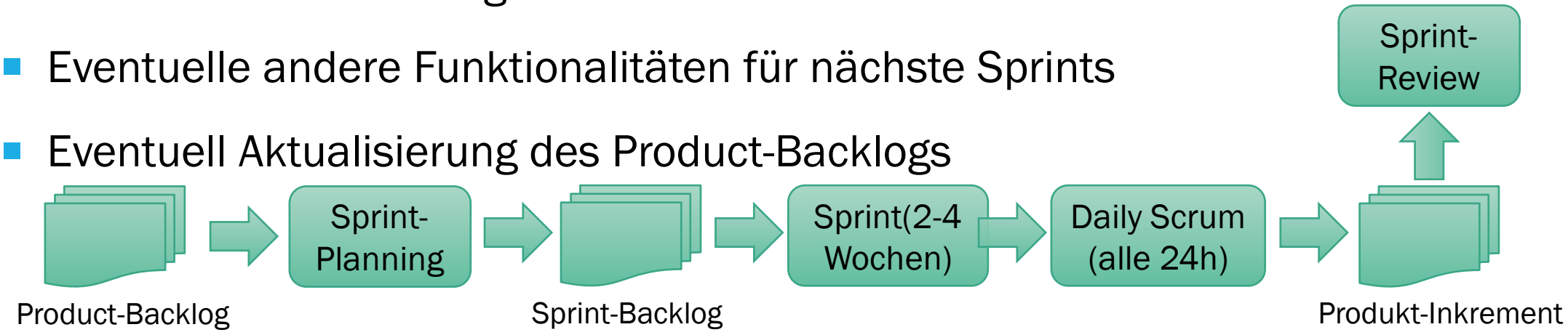
SCRUM – DER PROZESS

- **Produkt-Inkrement:**
- Fertig entwickeltes Produkt-Inkrement, dass sofort ausgeliefert werden könnte



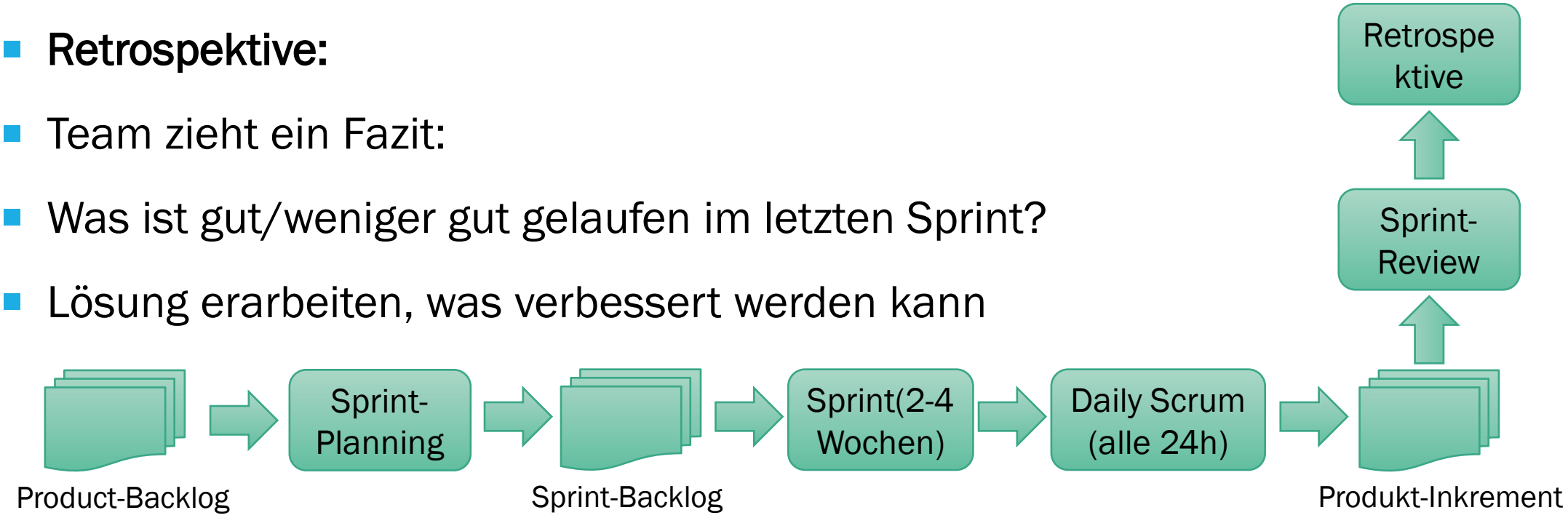
SCRUM – DER PROZESS

- **Sprint-Review:**
- Überprüfung des Ergebnisses
- Erfüllt es die Erwartungen?
- Eventuelle andere Funktionalitäten für nächste Sprints
- Eventuell Aktualisierung des Product-Backlogs



SCRUM – DER PROZESS

- **Retrospektive:**
- Team zieht ein Fazit:
- Was ist gut/weniger gut gelaufen im letzten Sprint?
- Lösung erarbeiten, was verbessert werden kann



SCRUM – DIE ROLLEN

- **Product-Owner:**
- Verantwortlich für den wirtschaftlichen Erfolg des Produkts
- Vertritt die Interessen der Stakeholder
- Hält Anforderungen im Product-Backlog fest
  - Entscheidet, in welcher Reihenfolge die Product-Backlog Einträge umgesetzt werden
- Product-Backlog ist nicht statisch
- Produkt-Inkrement überprüfen
  - „Definition of Done“

Wichtig: Der Product-Owner darf während eines Sprints nicht eingreifen

SCRUM – DIE ROLLEN

- **Scrum-Master:**
- Der beste Freund des Teams
- Aufgaben:
  - Organisatorischer Ablauf von Scrum
  - Anlaufstelle für nicht-fachliche Fragen
  - Vermittler bei Meinungsverschiedenheiten
  - Beschaffung benötigter Ressourcen
  - Beseitigt Hindernisse

SCRUM – DIE ROLLEN

- **Entwicklungsteam:**
- Besteht in der Regel aus 3-9 Mitgliedern und ist interdisziplinär
  - Team verfügt über das komplette erforderliche Know-How
- Neben fachlichen Fähigkeiten sind auch die sozialen Fähigkeiten wichtig
  - Team arbeitet komplett selbstorganisiert
  - Gleichberechtigung zwischen allen Teammitgliedern
- Aufgaben:
  - Realisierbaren Umfang von Sprint-Backlog einschätzen
  - Aufgaben und Tasks herunterbrechen
  - Aufgaben verteilen und abarbeiten

SCRUM – DIE ARTEFAKTE

- **Artefakte:**
- Dienen in der Softwareentwicklung dazu, den Überblick zu behalten

SCRUM – DIE ARTEFAKTE

- **Product-Backlog:**
- Existiert in jedem Projekt genau einmal
- Verantwortlich: Product-Owner
- Enthält Liste von Anforderungen an das zu erstellende Produkt
- Ist ein „lebendiges Artefakt“
  - Entwickelt sich über die Zeit weiter
- Product-Backlog Items sind geordnet

SCRUM – DIE ARTEFAKTE

- **Sprint-Backlog:**
- Liste von Product-Backlog Items, die im kommenden Sprint umgesetzt werden sollen
  - Enthält zudem zugehörige Tasks
- Verantwortlich: Entwicklungsteam
- Gibt auf einen Blick aktuellen Bearbeitungsstand wieder

SCRUM – DIE ARTEFAKTE

- **Produkt-Inkrement:**
- Ergebnis des Sprints
  - Muss potenziell auslieferbar sein
  - Muss wie geplant umgesetzt worden sein(„Definition of Done“)

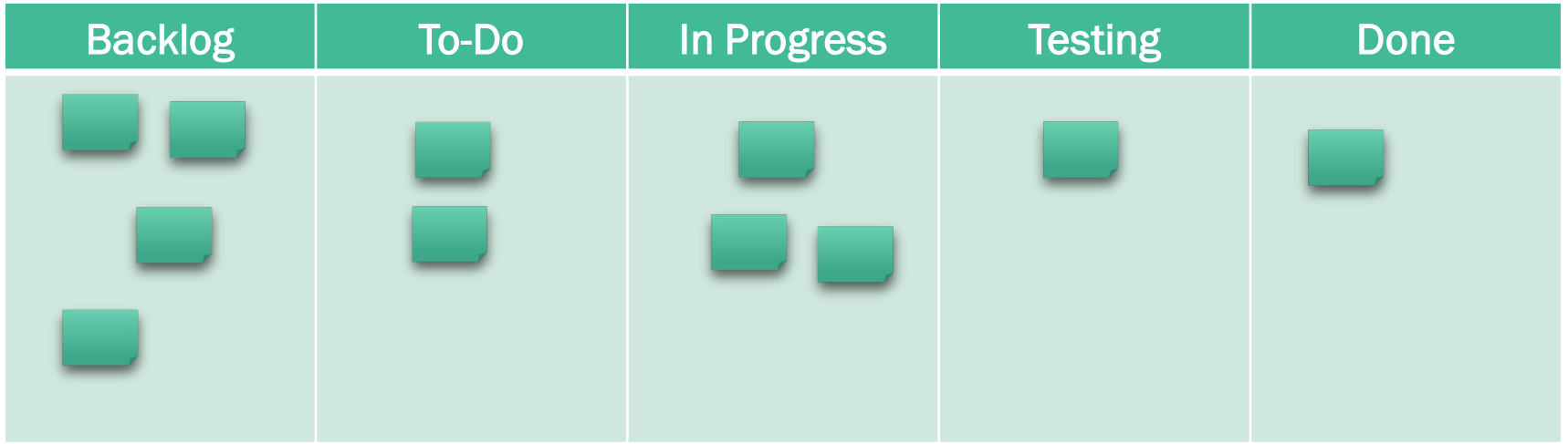
VORGEHENSMODELL KANBAN

ALLGEMEINES ZU KANBAN

- Japanisch für Signalkarte
- Ursprung: Toyota-Produktionssystem
  - Optimaler Fluss durch Fertigung
- Kanban Prinzipien können auf Softwareentwicklung übertragen werden.
  - Begründer: David Anderson(2007)

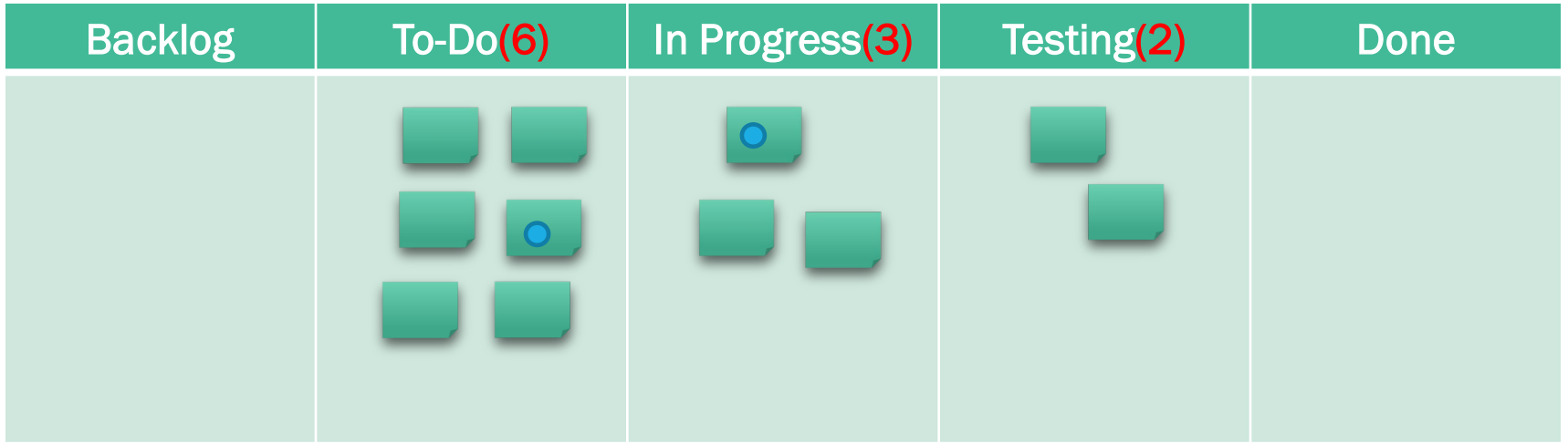
KANBAN PRINZIPIEN

- **Erstes Prinzip:** Visualisiere den Fluss von Arbeit
- Kanban-Board:
- Vorteil: Stets Überblick über das gesamte Projekt



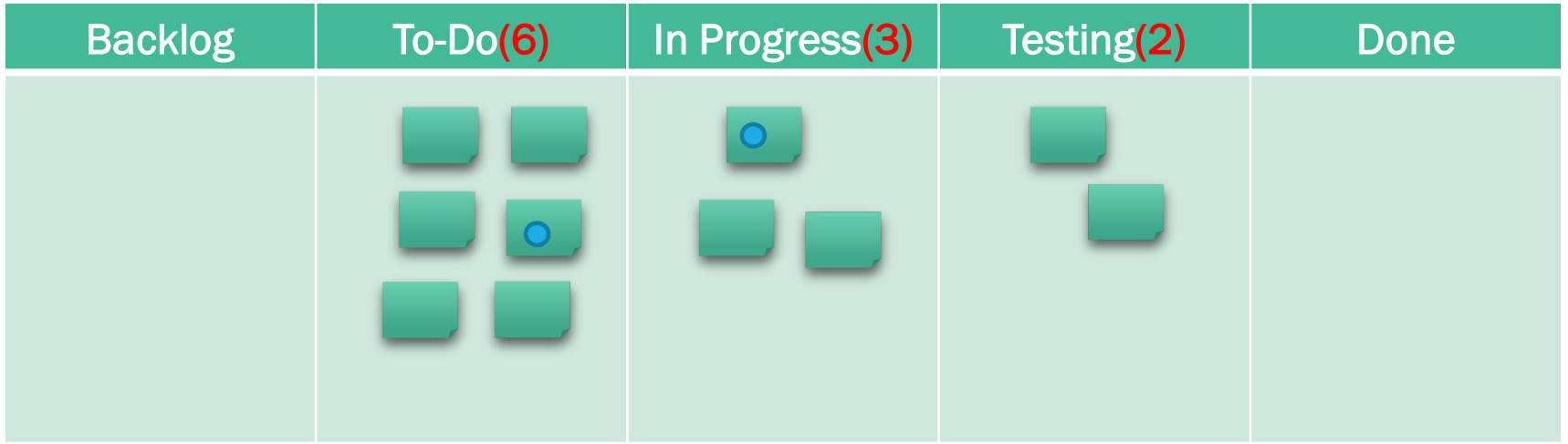
KANBAN PRINZIPIEN

- **Zweites Prinzip:** Anzahl der Tickets (pro Prozessschritt) begrenzen
- Pull Prinzip: Tickets werden in den nächsten Prozessschritt gezogen und dafür markiert



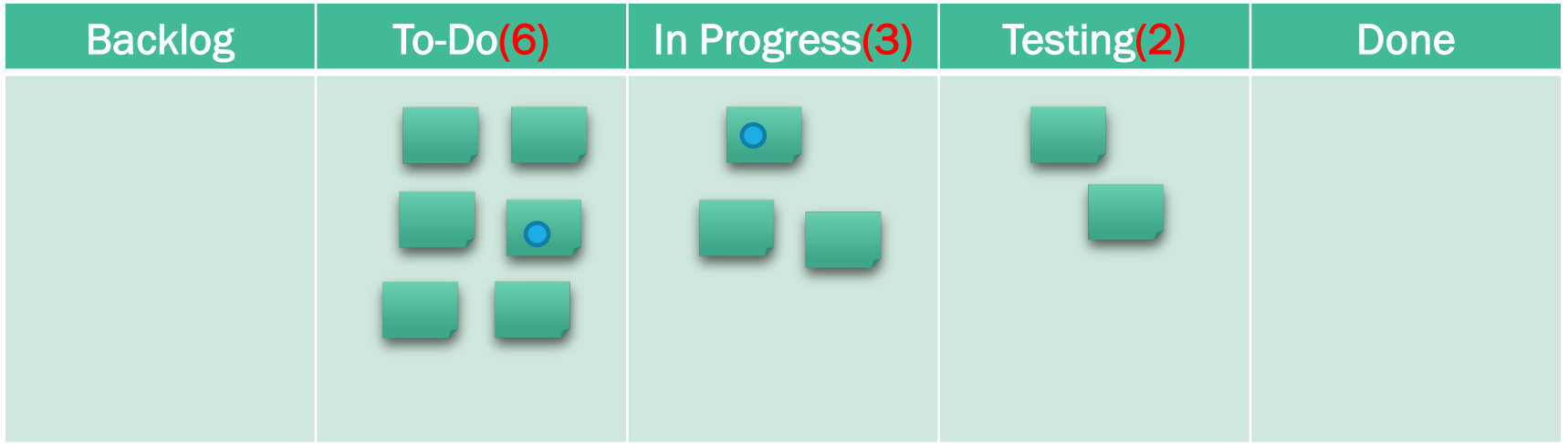
KANBAN PRINZIPIEN

- **Drittes Prinzip:** Den Fluss messen und steuern
- Durch die Übersichtliche visuelle Darstellung kann der Projektverlauf gemessen werden



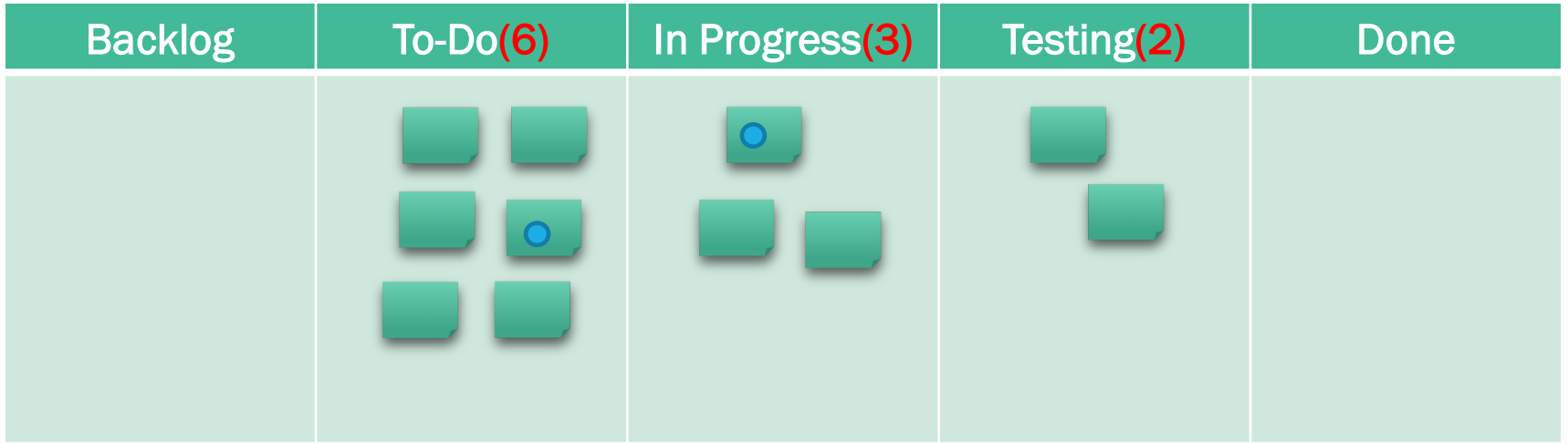
KANBAN PRINZIPIEN

- **Viertes Prinzip:** Regeln für die Prozesse explizit definieren
- Schafft eine gemeinsame Objektive Basis



KANBAN PRINZIPIEN

- **Fünftes Prinzip:** Den Prozess kontinuierlich verbessern
- z.B. durch tägliche Standup-Meetings oder Retrospektiven

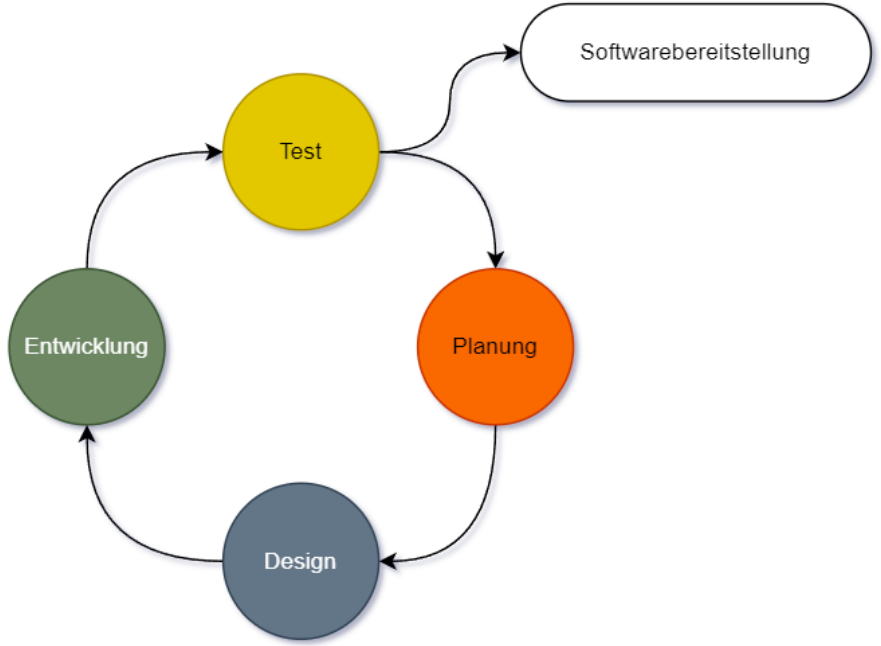


VORGEHENSMODELL SCRUMBAN

SCRUMBAN

Wie Sie vielleicht schon erraten haben, ist Scrumban eine Methode, die sowohl von Scrum als auch von Kanban geprägt ist. Manche sehen darin einen hybriden Ansatz, der das Beste von beiden Systemen in sich vereint.

Scrumban verwendet einen ähnlichen Sprint-Zyklus wie Scrum, erlaubt es jedoch wie Kanban, einzelne Aufgaben in den Plan aufzunehmen. So können die wichtigsten Arbeitsschritte ausgeführt werden und die Projektpläne bleiben einfach. Zudem setzt Scrumban Scrum-Meetings ein, um die Zusammenarbeit zu verbessern und die Ziele im Auge zu behalten.

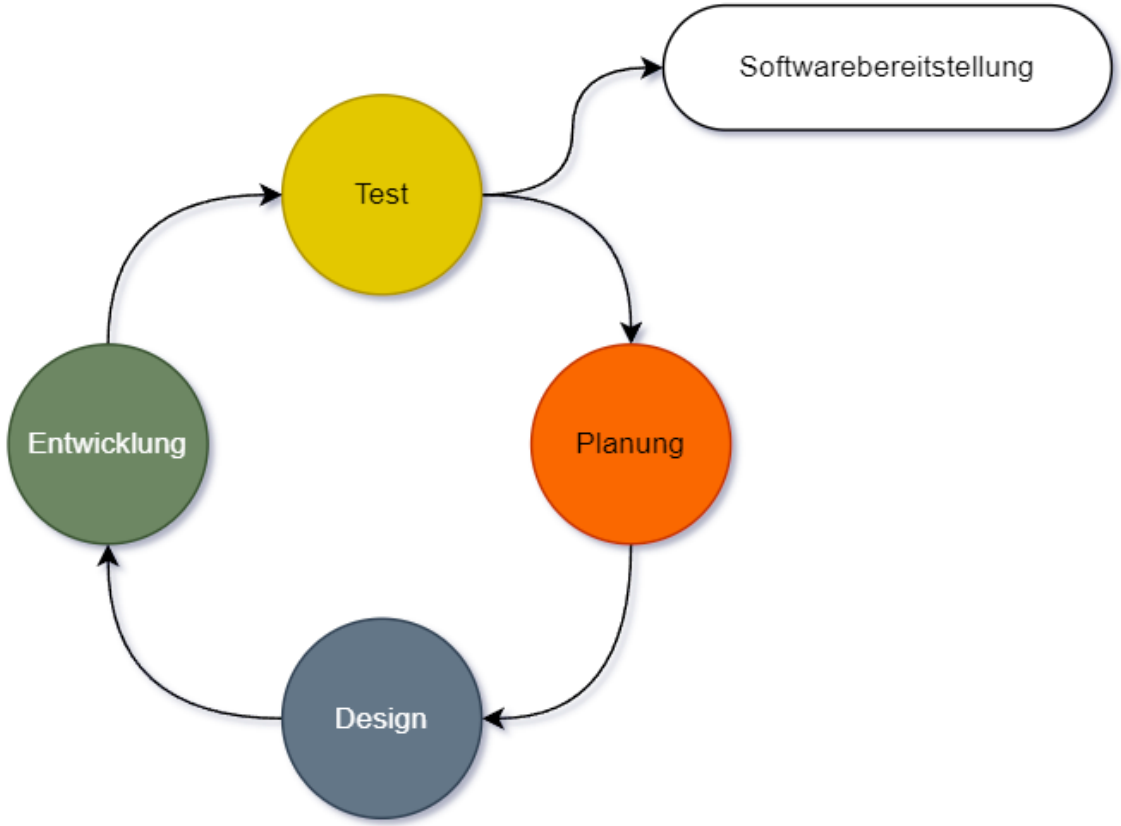


VORGEHENSMODELL EXTREME PROGRAMMING

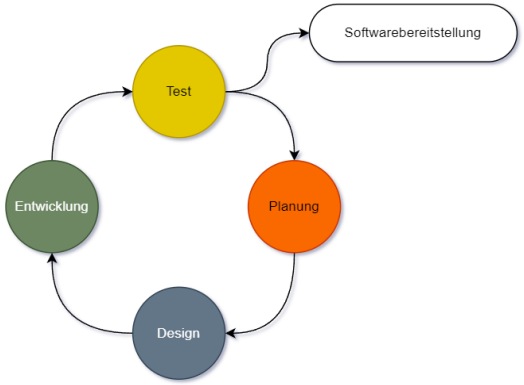
WAS IST EXTREME PROGRAMMING?

Der Begriff „Extreme Programming (XP)“ beschreibt im Wesentlichen die Art und Weise, wie Software programmiert wird. Agile Prozesse, kurze Entwicklungszyklen und schnelle Reaktionszeiten auf neue oder sich ändernde Anforderungen stehen dabei im Vordergrund.

Als Vater des Extreme Programming gilt der amerikanische Software-Ingenieur Kent Beck. Gemäß Insidern aus der Software-Programmierer-Szene beschreibt XP den extremsten Fall hinsichtlich agiler Vorgehensmethodik bei der Software-Programmierung. Die „Feuertaufe“ bestand XP in den Jahren 1997-1998 bei der programmtechnischen Umsetzung des Daimler-Chrysler „Project C“ – Chrysler Consolidated Compensation.



WAS IST EXTREME PROGRAMMING?

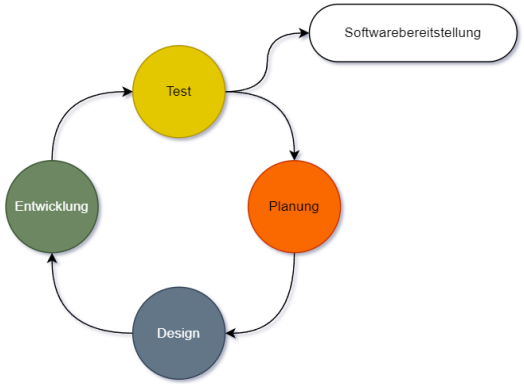


Das interne Softwareprogramm für eine neue Mitarbeiter-Gehaltsabrechnung war in eine kritische Sackgasse geraten. Eine vom Autobauer engagierte Software-Entwicklungsgruppe um Kent Beck setzte das Projekt, basierend auf der Methode „Extreme Programming“ neu auf. Der neue Weg war von Erfolg gekrönt.

Beck fasste die in diesem Projekt gesammelten Erfahrungen in seinem Buch „Extreme Programming Explained“ zusammen. Als Gerüst für die Rahmenbedingungen der Extremprogrammierung bezeichnet der Autor darin die drei Hauptbestandteile

- Werte (Values),
- Prinzipien (Principles) und
- Techniken (Practices).

WAS IST EXTREME PROGRAMMING?

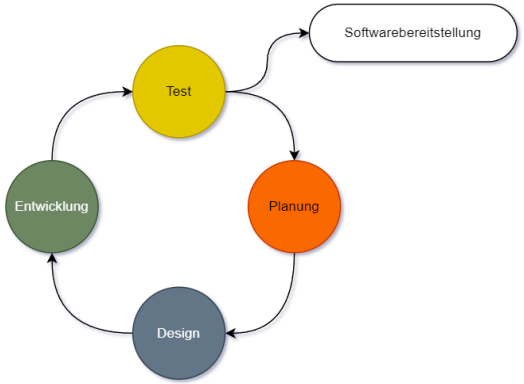


Charakteristisch: zyklische Vorgehensweise

Charakteristisch beim Extreme Programming ist die zyklische Verfahrensweise auf den Entwicklungsprojekt-Ebenen. Diese Struktur betrifft die eigentliche Programmierung. Sie wirkt aber auch prägend auf die obligatorische Abstimmung im Entwicklungsprojekt-Team.

Diesem Zyklus unterliegt letztlich auch das gemeinsam mit dem Kunden definierte Anforderungsmanagement. Spätestens an dieser Stelle wird deutlich, dass XP auf einen vom Kunden strikt vorgegebenen Projekt-Anforderungskatalog bewusst verzichtet.

WAS IST EXTREME PROGRAMMING?

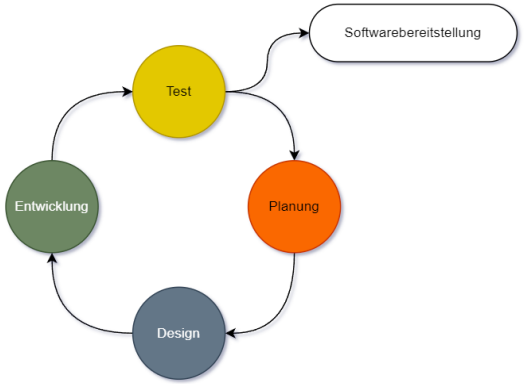


An seine Stelle tritt das sogenannte „Agile Modeling“. Es ermöglicht die Berücksichtigung von Kundenwünschen im Laufe der voran schreitenden Software-Entwicklung. So entstehen Entwicklungszyklen in Einheiten von einem Tag bis zu einer Woche. Sämtliche Themeninhalte einer komplexen Programmentwicklung, zum Beispiel

- Analyse der Anforderungen
- Produkt Design
- Implementierung
- Testphasen und Testtiefe

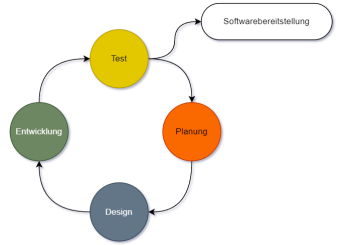
werden in diesen kurzen Zyklen aktualisiert und in den weiteren Entwicklungsprozess implementiert.

WAS IST EXTREME PROGRAMMING?



Diese zeitnahe und problemorientierte Verfahrensweise trägt der Erkenntnis Rechnung, dass Kunden und Auftraggeber die tatsächlichen Anforderungen und Leistungsprofile bei Projektbeginn oft nicht im Detail kennen. Die Erfahrung bei der klassischen Projektvorgabe zeigt auf, wie Pflichtenhefte häufig mit entbehrlichen Features belastet werden, während essentielle Funktionen vergessen werden.

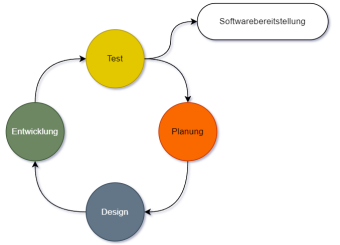
VORTEILE VON XP BEI KOMPLEXEN ENTWICKLUNGSPROJEKTEN



Die Vorteile von Extreme Programming entfalten insbesondere bei komplexen Software-Entwicklungsprojekten ihre problemorientierte Wirkung. An die Stelle einer komplexen, starren Projektvorgabe tritt die in kürzere Iterationszyklen unterteilte Vorgehensweise.

Auf diesem Wege wird die Software-Entwicklung vereinfacht, was die aktive Einbindung der Kunden und Auftraggeber in das Projekt ermöglicht. Insbesondere diese Möglichkeit der Kommunikation mit dem Kunden gibt den nächsten Programm-Entwicklungsschritten eine konkrete Richtung. Kundenwünsche lassen sich zeitnah korrigieren, revidieren oder ergänzend konkretisieren.

VORTEILE VON XP BEI KOMPLEXEN ENTWICKLUNGSPROJEKTEN



Die Software-Entwicklung lässt sich dank dieser zyklischen Vorgehensweise signifikant beschleunigen. Auch der mit dem reduzierten Zeitaufwand gekoppelte Kostenaspekt birgt Einsparpotenziale. Im günstigsten Falle bewirkt das Zusammenspiel erfahrener Programmierer in Verbindung mit aktiv eingebundenen Kunden Projektvorteile, die sich mit Hilfe des Extreme Programming in Form eines qualitativ hochwertigen, passgenauen Software-Programms darstellen.

Zu guter Letzt sollte noch einmal darauf hingewiesen werden, dass das in Verbindung mit der Extremprogrammierung und ihrer Organisationsform anzutreffende Kürzel „XP“ keinerlei Bezug zum Betriebssystem Windows XP hat, bei dem es für Experience stand.

PRINCE2

PRINCE2

PRINCE2, auch bekannt als Projects IN Controlled Environments (zu dt. „Projekte in kontrollierten Umgebungen“), verwendet die übergreifende Wasserfallmethode, um Phasen innerhalb eines Projekts zu definieren. Sie wurde ursprünglich von der britischen Regierung für IT-Projekte entwickelt und eignet sich immer noch in erster Linie für große IT-Vorhaben und nicht für herkömmliche produkt- oder marktorientierte Projekte.

PRINCE2

Zu den sieben wichtigsten Grundbereichen von PRINCE2 zählen:

1. Starten eines Projekts
2. Lenken eines Projekts
3. Initiieren eines Projekts
4. Steuerung eines Projekts
5. Managen der Produktbereitstellung
6. Managen eines Phasenübergangs
7. Abschließen eines Projekts

PRINCE2

Diese sieben Bereiche schaffen einen durchgängigen Projektprozess und bilden insgesamt eine effektive Projektmethodik für Unternehmen. Ziel ist es, Funktionen zu definieren und das Management zu unterstützen. Darüber hinaus kann PRINCE2 zur Rationalisierung einer Vielzahl von einzelnen Projektmanagementaufgaben eingesetzt werden, wie etwa zur Steuerung einer Phase, zum Management der Produktbereitstellung sowie zur Initiierung und zum Abschluss eines Projekts.

SIX SIGMA

6σ

SIX SIGMA

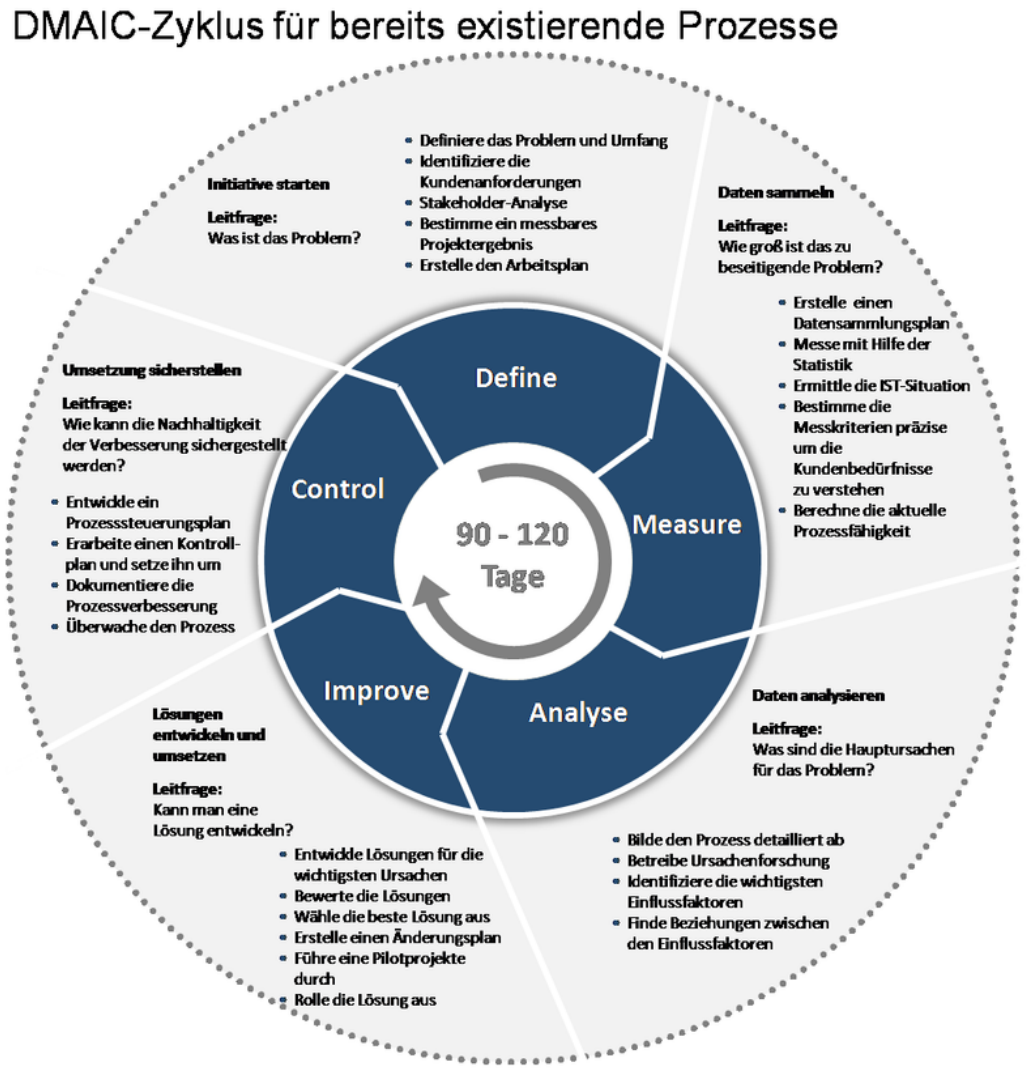
Im Gegensatz zu den anderen PM-Methoden wird Six Sigma im Qualitätsmanagement eingesetzt. Es wird häufig eher als eine Philosophie und nicht als eine herkömmliche Methodik beschrieben. Oft wird Six Sigma mit einer Lean-Methode oder einem Agile-Modell kombiniert, auch bekannt als Lean Six Sigma und Agile Six Sigma.

Hauptziel von Six Sigma ist die kontinuierliche Verbesserung von Prozessen und die Beseitigung von Fehlern. Dies geschieht durch kontinuierliche Verbesserungen durch Experten vor Ort, die Prozesse unterstützen, definieren und kontrollieren.

SIX SIGMA

Um bei dieser Methode noch einen Schritt weiterzugehen, können Sie einen Six Sigma DMAIC-Prozess einsetzen, bei dem eine phasenweise Vorgehensweise festgelegt wird. Diese Phasen umfassen:

- Definieren: Erstellung des Projektumfangs, Business Case und Durchführung eines ersten Stand-up-Meetings.
- Messen: Erfassung von Daten, die Aufschluss über Verbesserungsbedarf geben.
- Analysieren: Identifizierung der grundlegenden Ursachen eines Problems.
- Verbessern: Beseitigung der erkannten Ursachen.
- Kontrolle: Fortlaufender Einsatz der Lösungen für künftige Projekte.



Quelle: Six-Sigma-Deutschland, Prof. Dr. Matthias Schmieder

CRITICAL-PATH-METHODE

CRITICAL-PATH-METHODE

- Die Critical-Path-Methode dient dazu, die entscheidenden Aufgaben innerhalb eines Projekts zu ermitteln und zu planen. Dazu gehören die Erstellung von Aufgabenabhängigkeiten, die Nachverfolgung von Projektzielen und -fortschritten, die Priorisierung von Ergebnissen und die Verwaltung von Fälligkeitsdaten, die alle mit einem Projektstrukturplan vergleichbar sind.

- Ziel dieser Methodik ist es, erfolgreiche Projekte in großem Maßstab richtig zu managen, so dass Meilensteine und Ergebnisse korrekt abgebildet werden.

TEST DRIVEN DEVELOPMENT

TDD

TEST DRIVEN DEVELOPMENT

- Test Driven Development (deutsch: testgeleitete Entwicklung; kurz: TDD) ist ein Entwicklungs- und Designparadigma für Software, bei dem das Testen von Programmkomponenten dazu verwendet wird, den gesamten Prozess der Softwareentwicklung zu leiten. Test Driven Development ist eine Designstrategie, die das Testen vor dem Erstellen des Quellcodes ansiedelt und mit Bezug auf die Abläufe vorrangig behandelt. Das Ziel liegt darin, die Qualität der Software maßgeblich zu erhöhen und den Wartungsaufwand im Nachhinein zu verringern. TDD wird meist im Rahmen agiler Methoden und insbesondere beim Extreme Programming verwendet. Weitere Bezeichnungen für TDD sind testgetriebene Softwareentwicklung, testgeleitete Programmierung oder Test First Design.

TEST DRIVEN DEVELOPMENT - ALLGEMEINES

Developer Testing war bereits in den Anfängen der Softwareentwicklung präsent. Während unabhängige Tester Softwarelösungen prüfen sollten, waren die Entwickler damit befasst, den eigentlichen Code zu schreiben. Die Entwicklung und das Testen waren üblicherweise entkoppelt. Erst als Kent Beck den Aspekt des Testens in seinen Ausführungen zum Extreme Programming mehr und mehr betonte, wurde der Test-First-Ansatz einem breitem Publikum bekannt. Beck und seine Mitstreiter sagten zum Beispiel, dass sie gewöhnlich den Test zuerst schreiben und der Quellcode nach den Testfällen geschrieben und implementiert wird. Durch die fortschreitende Entwicklung hin zu agilen Methoden, fand der Test-First-Ansatz auch Einzug in andere Programmierparadigmen und wird heutzutage teilweise als eigene Designstrategie betrachtet.

TEST DRIVEN DEVELOPMENT - ALLGEMEINES

Im Gegensatz zum nachgeschalteten Testing, das bei herkömmlichen Paradigmen wie dem Wasserfallmodell eingesetzt wird, sind die Testfälle im TDD der Ausgangspunkt für das iterative, sich wiederholende Vorgehen: Zunächst werden Testfälle bestimmt und realisiert. Diese Tests schlagen häufig fehl. Anschließend wird genauso so viel Code verfasst, wie es für das Bestehen des Tests notwendig ist. Diese Codebestandteile werden dann refaktorisiert; das heißt, dass der Quellcode unter Beibehaltung von Funktionen sukzessiv erweitert oder neu strukturiert wird. Dies erfolgt zyklisch, bis alle Anforderungen an die Software erfüllt werden und der Code in das Produktivsystem übertragen werden kann.

TEST DRIVEN DEVELOPMENT - FUNKTIONSWEISE

Test Driven Development läuft inkrementell ab: Die Software wird Schritt für Schritt erweitert, nachdem die ersten Testfälle geschrieben wurden. Bei jedem Schritt wird die Software mit teilweise minimalen Funktionen angereichert und wieder getestet. Jeder fehlerhafte Test zieht das Verfassen von Quellcode nach sich; jeder bestandene Test erweitert den Funktionsumfang oder stellt die Funktionalität der Software sicher. Einzelne Testfälle, auch Unittests genannt, nehmen in der Regel nur wenig Zeit in Anspruch, sodass der Fortschritt bei der Softwareentwicklung unmittelbar sichtbar wird.

TEST DRIVEN DEVELOPMENT - FUNKTIONSWEISE

Einige Entwickler schreiben eine Zeile Code für das Testing und danach eine Zeile Code für den Releasekandidaten. Dieses Herunterbrechen auf kleine Bestandteile der Software führt zu einem Zyklus, der die täglichen Abläufe strukturiert. Exemplarisch wird dieses Vorgehen in den drei Gesetzen des Test Driven Development:

- Schreibe einen Test, bevor du Code für das Produktivsystem verfasst.
- Schreibe nur so viel Code für den Test, wie du für das Nichtbestehen des Tests benötigst (oder für das fehlgeschlagene Kompilieren)
- Schreibe nur so viel Code, wie es für den aktuellen Testfall und dessen Bestehen hinreichend ist.

TEST DRIVEN DEVELOPMENT - FUNKTIONSWEISE

Entwickler, die diese Regeln befolgen, dürften Zyklen verwenden, die nicht mehr als 30 Sekunden lang sind. Die nächste Stufe bildet ein Zyklus, der auf Minuten bezogen ist und auch als Red-Green-Refactor bezeichnet wird. Test schreiben: Test schlägt fehl und wird rot markiert.

Produktivcode schreiben und in das Produktivsystem integrieren: Test wird bestanden und wird grün markiert.

Refactoring: Der Code wird Schritt für Schritt erweitert, ergänzt und neu strukturiert. Entsprechende Testfälle und die zugehörigen Codebestandteile werden geschrieben. Sie durchlaufen den Zyklus erneut, bis die Software aus Entwicklersicht einfach, elegant und verständlich ist. Jede nicht getestete Codezeile kann beim Refactoring zu Problemen führen, da es hier in erster Linie um die Struktur des Codes geht und nicht um die Funktionalität der Software, die durch den Test-First-Ansatz gewährleistet wird.

TEST DRIVEN DEVELOPMENT - FUNKTIONSWEISE

Daneben gibt es weitere Zyklen, die sich auf die Evolution des Entwicklungsprozesses beziehen. Der sogenannte Specific-Generic-Cycle und der Primary Cycle sind zu erwähnen. Der erste Zyklus soll sicherstellen, dass einzelne Module in ihrer Gesamtheit das Ziel der Software erreichen. Der zweite Zyklus macht Grenzen deutlich, die durch die Architektur und die vorher bestimmten Regeln gegeben sind. Eine klare Systemarchitektur und das Ineinandergreifen aller Module und Komponenten sind hier das Ziel.

TEST DRIVEN DEVELOPMENT – VORTEILE/NACHTEILE

Der Ansatz des Test Driven Development hat gegenüber herkömmlichen Softwareentwicklungs-Methoden einige Vorteile.

- Das Resultat des TDD ist Software auf qualitativ hochwertigem Niveau.
- TDD sorgt für Software, die weniger wartungsintensiv und fehleranfällig ist.
- Die Fehleranalyse und eventuelle Wartungsarbeiten sind einfacher und schneller.
- Sowohl der Code als auch die Systemarchitektur sind sauber strukturiert und klar verständlich.
- Redundanzen und nicht benötigte Codebestandteile werden effektiv vermieden.

TEST DRIVEN DEVELOPMENT – BEDEUTUNG FÜR DIE PROGRAMMIERUNG

Vor dem Kontext agiler Methoden in der Softwareentwicklung bildet die testgeleitete Entwicklung einen generellen Designansatz und kein Testing-Szenario. Die Tatsache, dass zuerst getestet wird beziehungsweise Testfälle geschrieben werden, führt dazu, dass das Testing die Triebfeder dieses Paradigmas ist. Manche Entwickler würden keine Zeile Quellcode schreiben, bevor nicht ein Testfall für die Codezeile vorliegt. Auf diese Weise wird nicht nur der Umfang des Codes reduziert, sondern auch die Effektivität des gesamten Projektes sichergestellt – inklusive kürzerer Veröffentlichungszyklen.

TEST DRIVEN DEVELOPMENT – BEDEUTUNG FÜR DIE PROGRAMMIERUNG

Die Tests sorgen für einen Fokus auf die Ziele der Software und deren Funktionalität. Zudem ist es leichter, weitere Funktionen hinzuzufügen und die Software zu erweitern – selbst für Entwickler, die nicht am Projekt beteiligt waren. Zwar kann die testgetriebene Programmierung Entwickler mit sehr hohen Fachkenntnissen und viel Erfahrung erfordern, die Resultate sprechen jedoch auch für sich – wartungsfreie Software ist zum einen kein Kostenfaktor, der nach dem Release zum Tragen kommen könnte. Zum anderen sorgt der Ansatz für einbahnfrei funktionierende Software, die beim Endnutzer gerne angewendet wird, weil sie keine Fehler aufweist.

FEATURE DRIVEN DEVELOPMENT

FDD

FEATURE DRIVEN DEVELOPMENT DEFINITION

- FDD ist eine Methode der agilen Softwareentwicklung, die die Eigenschaften (Feature) eines Systems in den Mittelpunkt stellt. Im Feature Driven Development gibt es drei Hauptrollen, den Chefarchitekten, die Chefprogrammierer und die Entwickler. Der Entwicklungsprozess besteht aus fünf Teilprozessen:

1. Entwickle ein Gesamtmodel
2. Erstelle eine Feature-Liste
3. Plane je Feature
4. Entwurf je Feature
5. Konstruiere je Feature

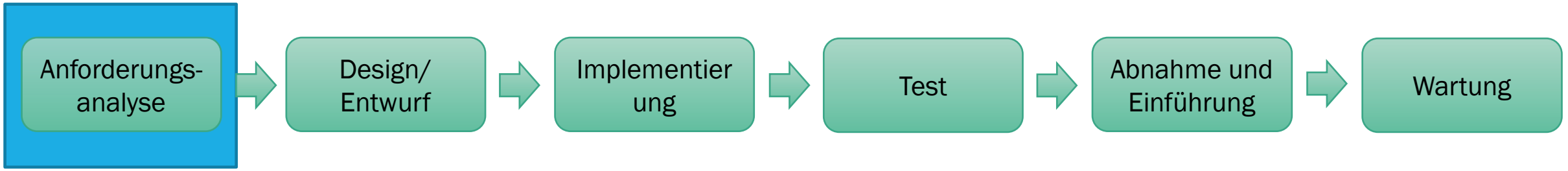
FEATURE DRIVEN DEVELOPMENT DEFINITION

Im ersten Teilprozess werden Inhalt und Umfang des Systems festgelegt, indem das System in Teilbereiche zerlegt wird und für jeden Teilbereich innerhalb einer Kleingruppe ein Fachkonzept entwickelt wird. Die vielen einzelnen Fachkonzepte müssen dann zusammengetragen, überarbeitet und aufeinander abgestimmt werden. Die Leitung dieser Phase obliegt dem Chefarchitekten, die Kleingruppen werden aus Entwicklern und Fachexperten gebildet. Aus dem so erstellten Gesamtmodell werden dann im zweiten Schritt von den Chefprogrammierern die Eigenschaften (Features) abgeleitet. Diese werden im dritten Schritt in die zu realisierende Reihenfolge gebracht und terminiert. Darüber hinaus werden für die einzelnen Teilbereiche und Eigenschaften die Verantwortlichkeiten (Owner) festgelegt. Im vierten Schritt werden die Eigenschaften anhand der festgelegten Verantwortlichkeiten Entwicklerteams zugewiesen, die für diese einen Feinentwurf erstellen. Im fünften Schritt erfolgt dann die Umsetzung anhand des Entwurfs.

REQUIREMENTS ENGINEERING

REQUIREMENTS ENGINEERING

- Findet in der Anforderungsanalyse statt



REQUIREMENTS ENGINEERING

- Requirements Engineering ist ein Prozess, in dem Kundenanforderungen für ein zu erstellendes System ermittelt, dokumentiert, überprüft und verwaltet werden
- **Ziel:** Anforderungen ermitteln, darüber nachdenken und verstehen
- **Prozess:**
  - Anforderungen ermitteln
  - Anforderungen dokumentieren
  - Anforderungen überprüfen und abstimmen
  - Anforderungen verwalten

ANFORDERUNGSARTEN

- Funktionale Anforderungen (Beschreiben, was ein System tun soll)
  - z.B. „Die Software soll basierend auf einer Formel die Kreditwürdigkeit berechnen“
  - z.B. „Die Software soll einen Mechanismus zur Identifizierung der Nutzer vorsehen“
- Nichtfunktionale Anforderungen(Einschränkende Bedingungen)
  - z.B. „Die Software soll dem Anwender innerhalb von 1 Sekunde antworten“

HÄUFIGE PROBLEME

- Stakeholder können Anforderungen oft nicht korrekt ausdrücken
- Fachsprache
- Widersprüchliche Sichten auf den Ist-Zustand
- Unklare Zielvorstellung der zu erstellenden Software
- Implizites und unbewusstes Wissen

ANFORDERUNGEN KATEGORISIEREN (KANO MODELL)

WARUM ANFORDERUNGEN KATEGORISIEREN

- Bei sehr kleinen Projekten nicht notwendig
- Bei komplexen Projekten sehr sinnvoll
  - Welche Anforderungen haben welchen Einfluss auf die Kundenzufriedenheit?

KANO MODELL

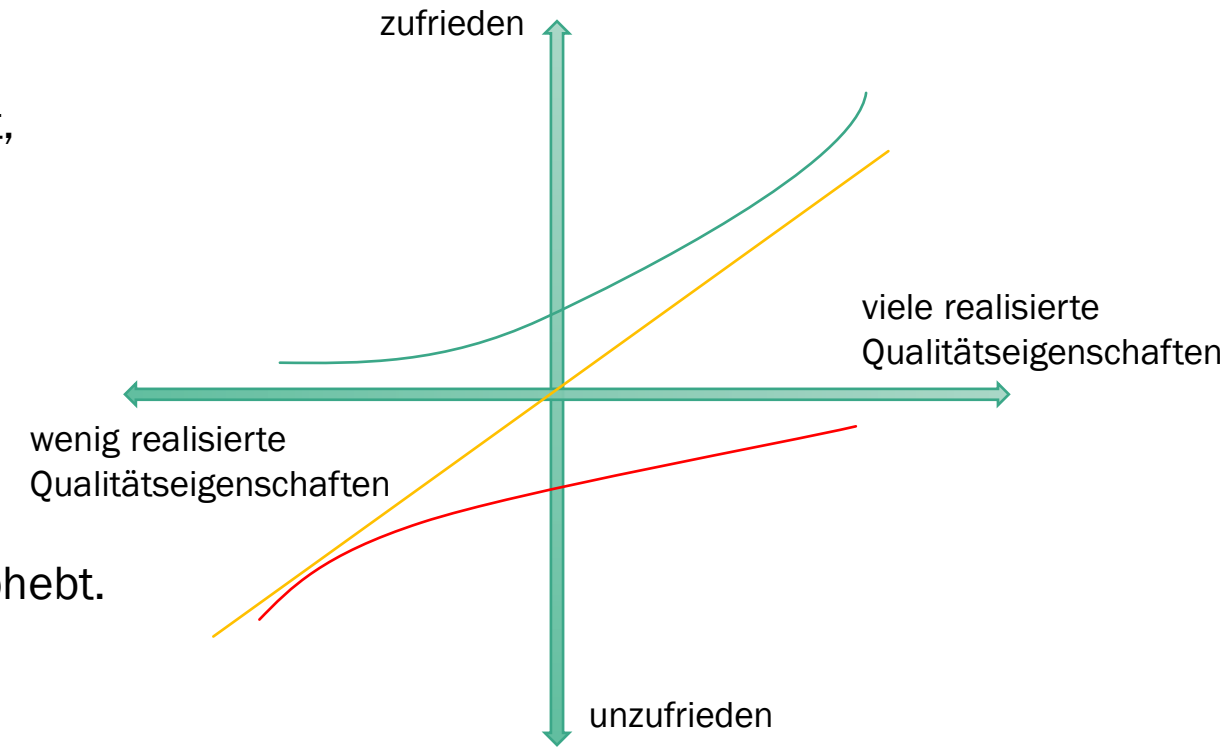
- Von Noriaki Kano (Professor an der Universität von Tokio) entwickelt
- Beschreibt Zusammenhang zwischen dem Erreichen bestimmter Eigenschaften eines Produkts und der zu erwarteten Zufriedenheit von Kunden

KANO MODELL

Basismerkmale
Merkmale, bei denen der Kunde davon ausgeht, dass sie vorhanden sind z.B. Toolleiste für Formatierungen bei einem Texteditor

Leistungsmerkmale
z.B. verschiedene Layouts für Texteditor

Begeisterunsmerkmale
Nutzenstiftende Merkmale, mit denen der Kunde nicht unbedingt rechnet,
Mit der sich das Produkt von der Konkurrenz abhebt.
z.B. Übersetzungsfunktion im Texteditor



AUFWANDSSCHÄTZUNG

WARUM AUFWANDSSCHÄTZUNG

- Softwareentwicklung kostet Geld
  - Personalkosten
  - Sachkosten
- Kosten müssen bekannt sein, um die Wirtschaftlichkeit zu prüfen
- Durch Aufwandsschätzung können Termine bestimmt werden

AUFWANDSSCHÄTZUNG METHODEN

- Expertenschätzung
  - Einzelschätzung
  - Mehrfachbefragung
  - Delphi-Methode
  - Schätz-Klausur
- Algorithmische Schätzung
  - Cocomo (Constructive Cost Modell)
  - Function Point Methode

DELPHI-METHODE

DELPHI-METHODE

- Systematisches, mehrstufiges Schätzverfahren
- Wird verwendet, wenn Prognosen aufgestellt werden müssen
- 1963 von der RAND-Corporation(amerikanische Denkfabrik) entwickelt

DELPHI-METHODE VORGEHENSWEISE

- Es werden erfahrene Experten gesucht
- Jedem Experten werden die Anforderungen vorgelegt
- Jeder Experte soll einzeln für sich eine Prognose für den Aufwand abgeben
- Danach werden alle Ergebnisse ausgewertet
- Auswertungen werden den einzelnen Experten wieder vorgelegt
  - **Wichtig:** Auswertungen sind anonymisiert, damit dominante Persönlichkeiten nicht zuviel Einfluss haben
- Jeder Experte kann eigene Prognose nun wieder anpassen
- Schritte werden so lange wiederholt, bis in einem gewissen Toleranzbereich ein Konsens entsteht

COCOMO AUFWANDSSCHÄTZUNG

WAS IST COCOMO

- COCOMO: Constructive Cost Model In der Softwareentwicklung zur Kosten- und Aufwandsschätzung verwendet
- 1981 von Barry W. Boehm entwickelt
- Algorithmisches Kostenmodell, dass auf einer Kombination von Gleichungen, statischen Modellen und Schätzungen von Parametervariablen beruht

RAHMENBEDINGUNGEN

- 17 identifizierte Kostentreiber, die in Algorithmen eingeflossen sind
- **Primärer Kostentreiber:** Delivered Source Instructions (DSI)
- **Beginn:** Entwurf des Produktdesigns
- **Ende:** Abschließende Tests durch den Kunden
- Anforderungsspezifikation darf sich im Laufe des Entwicklungsprozesses nicht mehr gravierend ändern

GROBER ABLAUF

- **Schritt 1:** Anzahl der auszuliefernden Codezeilen ermitteln
  - Einheit: KDSI (1000 Delivered Source Instructions)
- **Schritt 2:** Komplexität bestimmen
  - Organic Mode (kleine bis mittelgroße Projekte)
  - Semidetached Mode(mittelgroße Projekte) 50-350 KDSI
  - Embedded Mode(große Projekte) ab 350 KDSI
- Schritt 3: Aufwand und Projektdauer berechnen
  - **Einheit vom Aufwand:** Personenmonate (19 Arbeitstage mit jeweils 8 Arbeitsstunden)