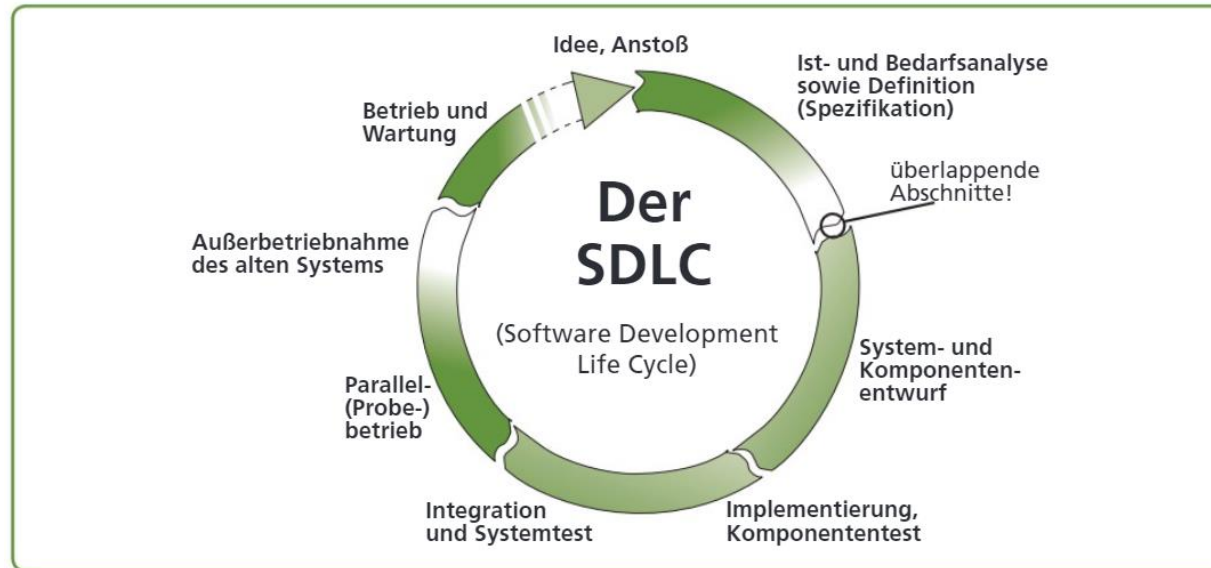


Phasenkonzepte und Phasenmodelle

- Phasenmodelle untergliedern Software-Projekte in einzelne Abschnitte (Phasen) und Vorgänge (Prozesse)
- Ziel ist es dabei, einen komplexen Ablauf planbarer zu machen
- die grundlegenden Phasen der verschiedenen Modelle werden dabei vom Software-Lebenszyklus (Software Development Life Cycle) abgeleitet

Software Development Life Cycle



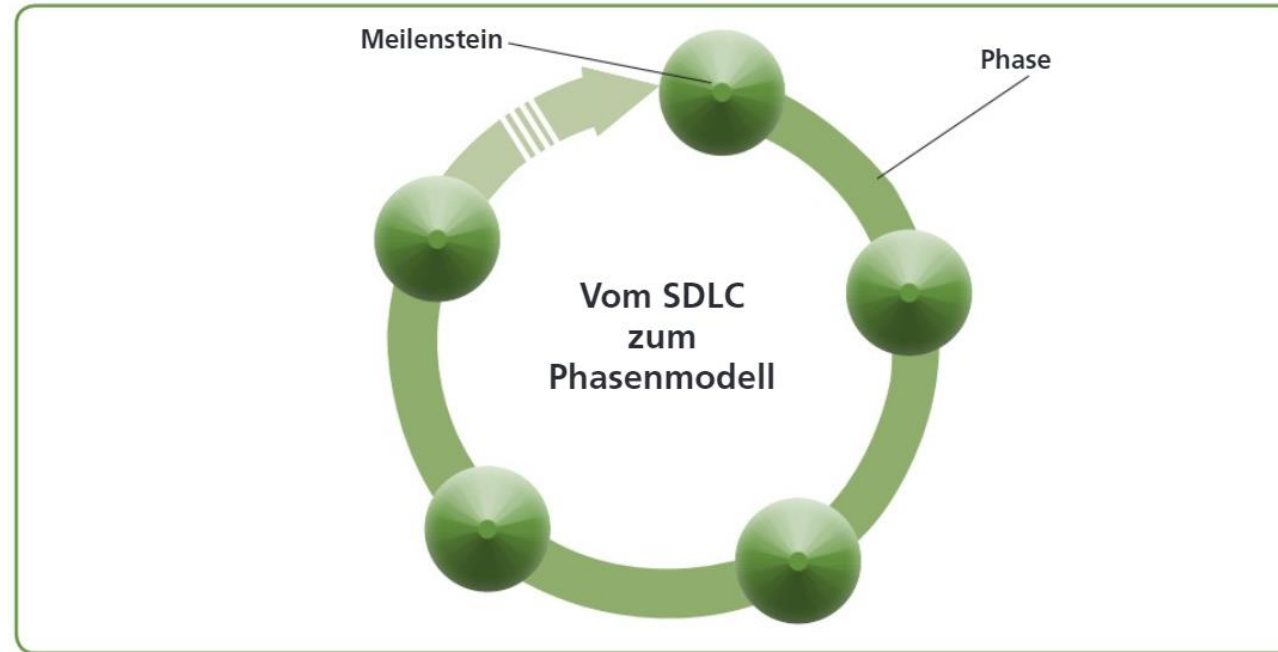
- der SDLC umfasst einen sehr viel längeren Zeitraum als das eigentliche SW-Entwicklungsprojekt
- dadurch ist er als Vorgehensmodell für die reine Entwicklung nicht geeignet

Software Development Life Cycle



- betrachtet man die Kosten während des SDLC, erkennt man, dass der Großteil der Kosten erst nach dem Ende der Entwicklung, während des Betriebes anfallen
- Ziel eines guten SW-Engineerings muss es also sein, während der Entwicklung mit einer hohen SW-Qualität dafür zu sorgen, dass die Betriebs- und Wartungskosten möglichst gering gehalten werden können

Vom SDLC zum Phasenmodell



- um vom SDLC zu einem konkreten Phasenmodell zu kommen, wird der SDLC in einzelne Meilensteine und Phasen unterteilt
- die Meilensteine grenzen die einzelnen Phasen dabei klar voneinander ab
- welche Aktivitäten in welcher Phase durchgeführt werden und wie die Phasen abgearbeitet werden, wird dann in den verschiedenen Prozessmodellen festgelegt

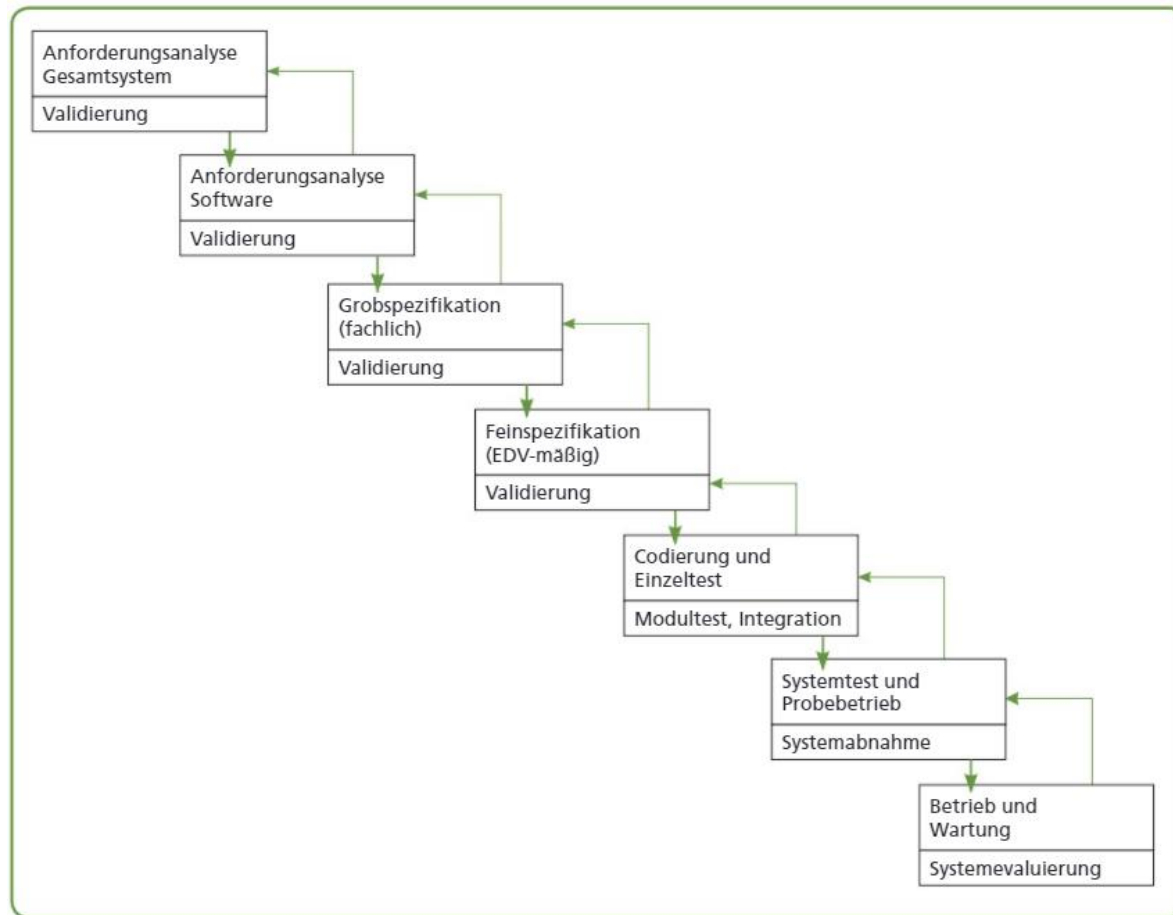
Phasenmodell, Prozessmodell

- **Phasenmodell**
 - ist der ältere Begriff
 - betont die zeitliche Abfolge von Projektabschnitten
- **Prozessmodell**
 - modernerer Begriff
 - streicht eher die Tätigkeiten (also Prozesse) während des SW-Entwicklungsprozess heraus
 - Aktivitäten im Projekt sind nicht mehr an eine strenge chronologische Abfolge gebunden
 - Aktivitäten können in mehreren Zyklen oder überlappend erfolgen
 - ein Prozessmodell gibt den Rahmen für Projekte vor und
 - legt die Anzahl der Phasen fest
 - definiert die Aufgabenschwerpunkte der Phasen bzw. Prozesse
 - regelt die Anordnung und Abfolge von Phasen und Prozessen

Phasenmodell, Prozessmodell

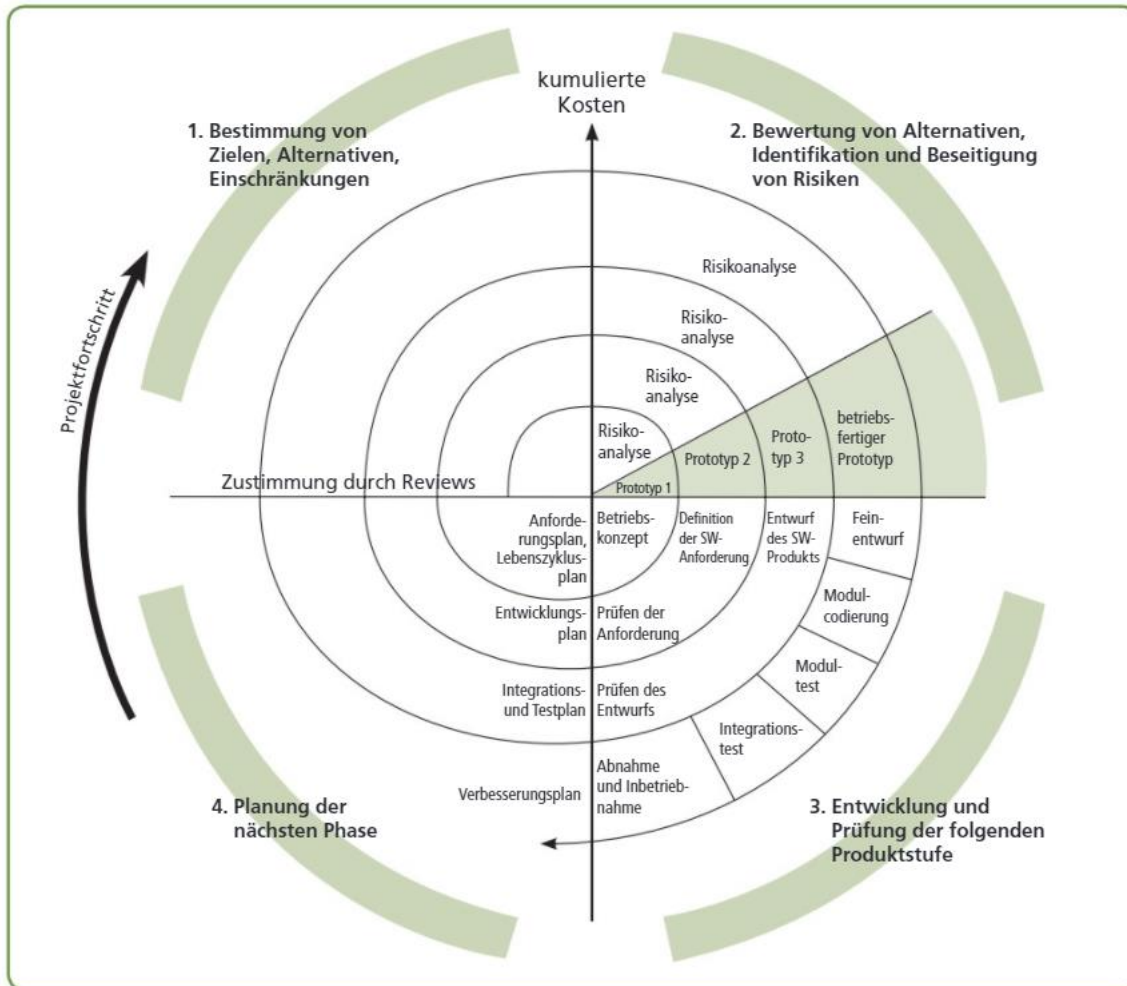
- klassischer Ablauf eines SW-Projekts
 - Analyse
 - Definition
 - Entwurf
 - Implementierung
 - Test
 - Betrieb und Wartung (schon außerhalb des Projekts)
- die verschiedenen Prozessmodelle unterscheiden sich hauptsächlich
 - in der Anzahl der Phasen
 - in der Form der Phasen-Abfolge (sequentiell, überlappend, iterativ, ...)
 - in den eingesetzten Methoden und Ergebnissen der einzelnen Phasen

Wasserfallmodell



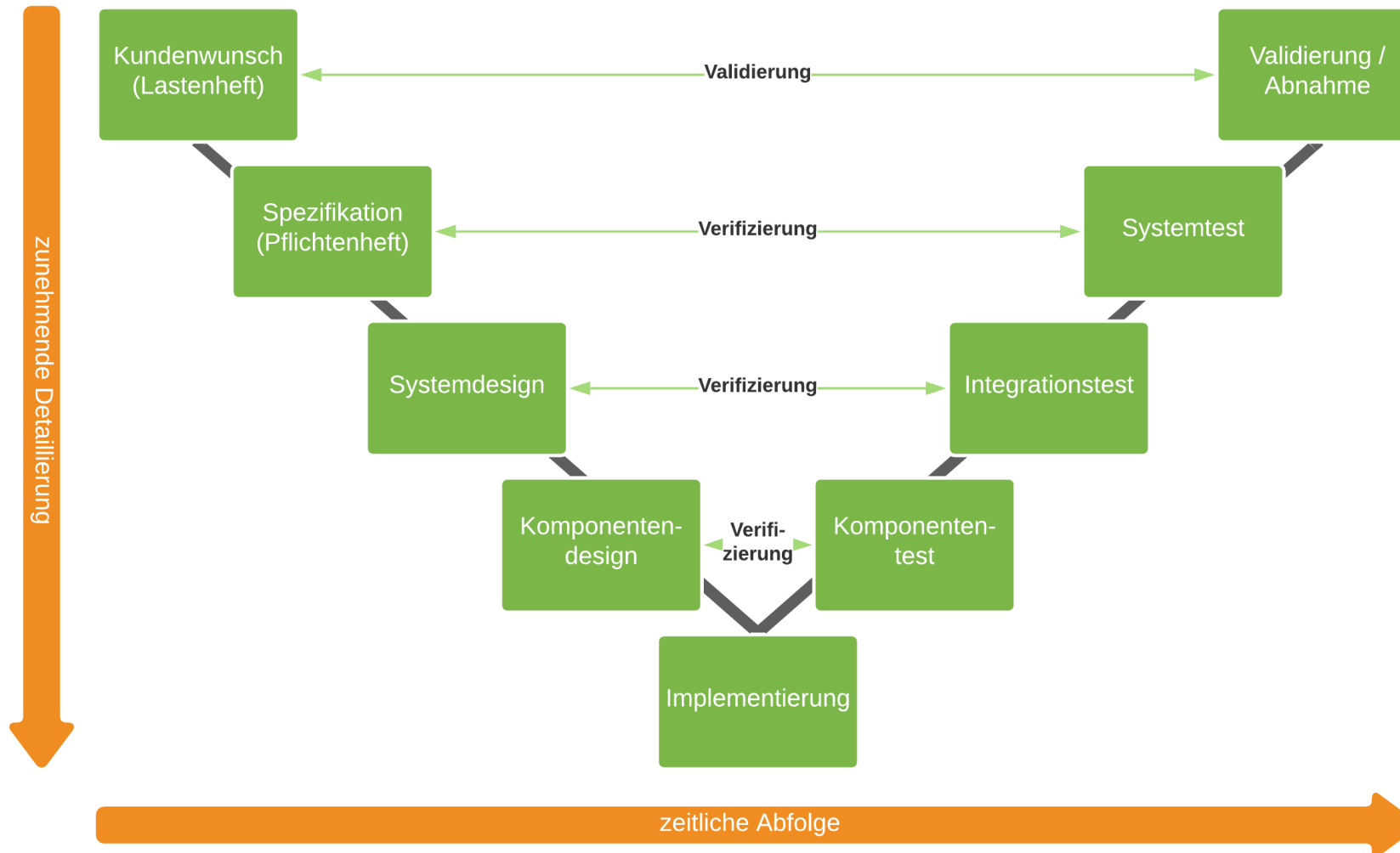
- Der Name leitet sich aus der Abfolge der Phasen ab → das Ergebnis einer Phase „fällt“, wie bei einem Wasserfall, in die nächste Phase
- Die Phasen werden **sequentiell** durchlaufen
- Rücksprünge in frühere Phasen sind nur in Ausnahmefällen (z. B. wenn ein Fehler gefunden wird) vorgesehen
- Für jede Projektphase wird genau definiert, was in dieser Phase erreicht werden muss und welche Lieferobjekte am Ende dieser Phase fertiggestellt sein müssen (z. B.: ein vom Kunden abgenommenes Lastenheft am Ende der Anforderungserhebung).
- Erst wenn alle geforderten Ergebnisse der Projektphase erreicht wurden, kann der Übergang in die nächste Phase erfolgen.
- Noch heute ein sehr häufig verwendetes Modell!

Spiralmodell



- Prozessmodell für Projekte mit hohem Risiko
- vor der konkreten Realisierung werden verschiedene Alternativen anhand von Prototypen analysiert und getestet
- das Spiralmodell hat hauptsächlich die folgenden Eigenschaften:
 - die Entwicklung der SW erfolgt evolutionär
 - zu Beginn jedes Zyklus erfolgt eine genaue Planung
 - am Ende jedes Zyklus erfolgt ein Review und es wird die Zustimmung aller Beteiligten zur Weiterführung des Projekts eingeholt
 - vor jedem Zyklus werden verschiedene Alternativen gesucht, bewertet und eine Risikoanalyse durchgeführt
 - der Bau von Prototypen unterstützen dabei Spezifikation, Entwurf und Risikoanalysen
 - Nach Beseitigung aller Risiken wird das konkrete SW-Produkt mittels Wasserfallmethode fertiggestellt

V-Modell



V-Modell

- V-Modelle werden hauptsächlich in Projekten mit hohen Qualitätssicherungsansprüchen eingesetzt
- Das V-Modell ist ein **sequenzielles** Vorgehensmodell
- der Name entsteht durch die Anordnung der einzelnen Projektphasen
- Beginnend auf der linken Seite des V wird das zu realisierende Projektergebnis vom Grobentwurf zum Feinentwurf immer detaillierter spezifiziert.
- Nach der Implementierungsphase wird die rechte Seite des V von unten nach oben durchlaufen.
- Bei diesem System hat jede Spezifikations- und Entwurfsphase auf der linken Seite des V eine entsprechende Testphase auf der rechten Seite des V, wodurch der Qualitätsgedanke in allen Phasen gefördert und fest ins Modell integriert wird.
- Im Zuge der **Verifizierung** wird dabei geprüft, ob die Anforderungen richtig umgesetzt wurden, das heißt, das umgesetzte Ergebnis wird auf Übereinstimmung mit der Spezifikation geprüft.
- Bei der **Validierung** wird geprüft, ob überhaupt die richtigen Anforderungen umgesetzt wurden. Das gesamte Produkt wird daraufhin überprüft, ob es den Erwartungen, Anforderungen und Bedürfnissen des Kunden entspricht.

2. Foliensatz

Agile Vorgehensmodelle

- allen agilen Vorgehensmodellen liegt zugrunde, dass ein **möglichst frühes und häufiges Feedback** von den relevanten Stakeholdern bzw. vom Kunden für fertiggestellte Teillieferungen eingeholt wird
- Es wird dabei davon ausgegangen, dass wertvolles und realistisches Feedback nur zu wirklich funktionierender Software (oder sehr realitätsnahen Prototypen) gegeben werden kann, weshalb danach getrachtet wird, **möglichst früh funktionierende Software** präsentieren zu können.
- folgende Grundprinzipien sollen dabei eingehalten werden
 - **iteratives Vorgehen** → es wird in kurzen Zyklen gearbeitet. In jedem Zyklus werden die klassischen Phasen (Planung, Analyse, Design, Implementierung, Test, ...) durchlaufen.
 - **inkrementelles Vorgehen** → am Ende jedes Zyklus wird ein lauffähiges Produktinkrement geliefert. Das Produkt wird dabei mit jeder Iteration um neue Features erweitert.
 - **mehr Kundenwert in kürzerer Zeit** → bei der Priorisierung der Anforderungen wird darauf geachtet, Features mit höherem Kundenwert früher auszuliefern, als Features mit geringem Kundenwert
- werden die Grundprinzipien eingehalten, kann man erreichen, dass der Kunde Teile des Produkts unter Umständen schon nutzen kann, bevor das gesamte Produkt fertiggestellt wurde

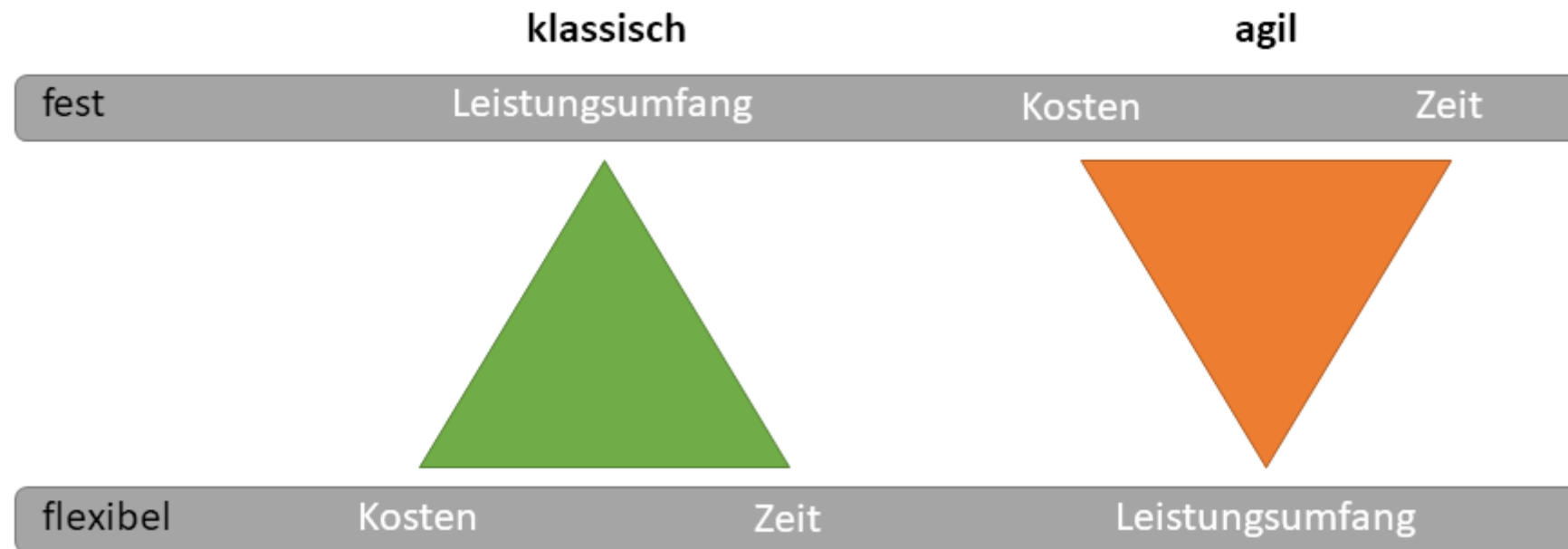
Agile Vorgehensmodelle

Unterschiede zu klassischen Modellen

- in klassischen Modellen wird versucht, den Projektlebenszyklus möglichst vorherzusehen und genau durchzuplanen
- In agilen Methoden wird von vornherein davon ausgegangen, dass man diesen Projektlebenszyklus am Anfang gar nicht genau genug vorhersehen kann und allzu genaue und starre Pläne daher nicht zum Erfolg führen
- es wird das geplant, was zu diesem Zeitpunkt bekannt ist und flexibel darauf reagiert, wenn neue Informationen bekannt werden, die man zu einem früheren Zeitpunkt nicht kennen konnte.
- Anforderungen werden nicht zu Beginn komplett erhoben, sondern immer nur so detailliert, wie im Moment notwendig
- die Anforderungen werden über die Projektlaufzeit immer detaillierter oder können sich sogar ändern
- agile Teams sind selbstorganisiert → das Team plant die Arbeit selbst und bekommen nicht einfach nur Vorgaben, die erfüllt werden müssen

Agile Vorgehensmodelle

Unterschiede zu klassischen Modellen - Projektdreieck



Agiles Manifest

- Im agilen Manifest werden Werte und Verhaltensregeln beschrieben, die den agilen Methoden zugrunde liegen und die der Arbeit von agilen Teams zugrunde liegen sollen.
- Das agile Manifest wurde im Jahr 2001 von einer Gruppe von renommierten Softwareentwicklern formuliert und unterzeichnet.
- Im agilen Manifest finden sich vier Leitsätze und zwölf agile Prinzipien.

Agiles Manifest - Leitsätze

„Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen.

Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

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

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.“

Agiles Manifest - Grundprinzipien

1. Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.
2. Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
3. Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
4. Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.
5. Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
6. Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
7. Funktionierende Software ist das wichtigste Fortschrittsmaß.
8. Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
9. Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
10. Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.
11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
12. In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

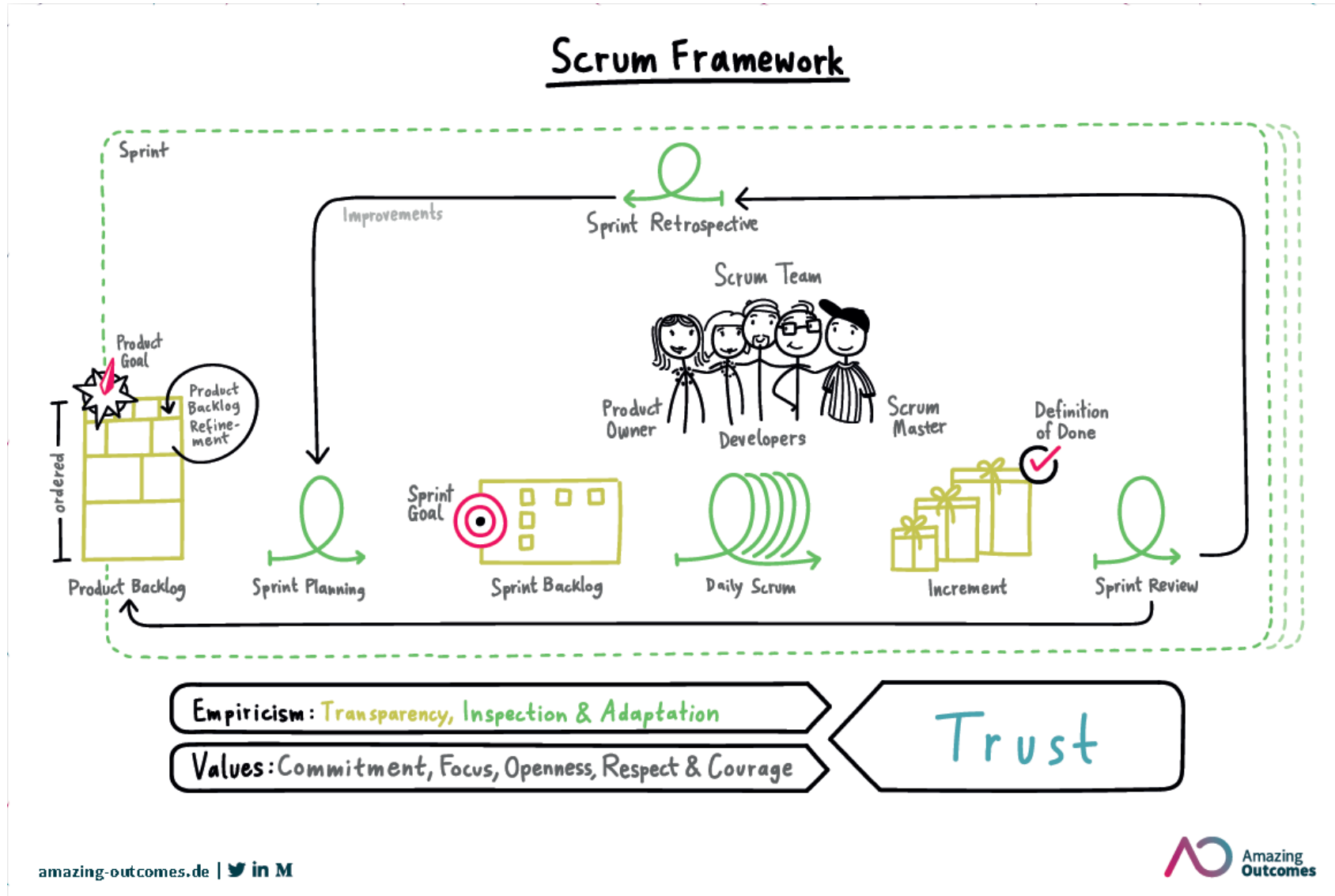
Scrum - allgemein

- Scrum an sich betrachtet sich selbst nicht als Methode oder als Prozess, sondern als ein Framework
- Die wichtigsten Funktionen des Frameworks sind die kontinuierliche Inspektion und Adaption sowohl von Arbeitsergebnissen als auch vom Prozess selbst.
- Projekte nach Scrum werden von selbstorganisierenden interdisziplinären (Experten aus allen notwendigen Fachgebieten) Teams durchgeführt
- Das Framework beinhaltet
 - 3 Rollen
 - 4 Meetings
 - 3 Artefakte

Scrum - allgemein

- 3 Rollen
 - Product Owner
 - Umsetzungsteam
 - Scrum Master
- 4 Meetings
 - Sprint planning
 - Sprint review
 - Sprint retrospective
 - Daily Scrum
- 3 Artefakte
 - Produkt Backlog
 - Sprint Backlog
 - Produkt-Inkrement

Scrum - allgemein



Scrum Rollen – Product Owner

- Der **Product Owner** verantwortet den geschäftlichen Erfolg des zu entwickelnden Produkts
- Er oder sie hat sicherzustellen, dass das richtige Produkt entwickelt wird und definiert dazu die Produkteigenschaften und verfeinert sie zu konkreten Anforderungen.
- Hauptaufgaben:
 - **Verantwortung für die Rentabilität des Produkts:** der Product Owner arbeitet mit den Kunden, Anwendern und Stakeholdern zusammen, konsolidiert deren Wünsche und bestimmt letztlich den Umfang und die Reihenfolge der gelieferten Funktionalitäten. Dabei soll stets gewährleistet sein, dass der beste Geschäftswert geliefert wird. Aufgrund dieser Verantwortung ist der Product Owner auch der Einzige, der die Reihenfolge der gewünschten Lieferungen bestimmt.
 - **Eine Vision etablieren:** der Product Owner hat ebenfalls die Aufgabe eine Produktvision zu etablieren, die kommuniziert werden kann. Dabei soll ein großes Ziel entwickelt werden, welches aufzeigt was das Produkt werden soll und wozu es entwickelt wird. Die Vision dient dem Scrum-Team sozusagen als Leitlinie durch das ganze Projekt.

Scrum Rollen – Product Owner

- Hauptaufgaben:
 - **Resultate akzeptieren oder ablehnen:** in der Rolle der oder des Produktverantwortlichen ist es auch Aufgabe eines Product Owners die vom Umsetzungsteam gelieferten Ergebnisse zu inspizieren und entweder zu akzeptieren oder zurückzuweisen.
 - **Agiles Anforderungsmanagement:** eine weitere Verantwortung ist die Pflege des Produkt-Backlogs (siehe weiter unten), also der Anforderungen an das zu entwickelnde Produkt. In agilen Projekten werden Anforderungen nicht zu Beginn gesammelt und bleiben dann weitgehend unverändert, sondern können einem stetigen Wandel unterworfen sein. Anforderungen können sich ändern, neue Anforderungen können dazu kommen, andere dafür wegfallen. Ebenso kann sich die Reihung der Anforderungen je nach Geschäftswert ändern. Daher ist es nötig den Anforderungskatalog in Form des Produkt-Backlogs ständig aktuell zu halten und wenn nötig zu überarbeiten.

Scrum Rollen – Product Owner

- Hauptaufgaben:
 - **Management von Lieferungen:** ebenso gehört es zu den Aufgaben des Product Owners, die Lieferungen an den Kunden zu planen. In jeder Entwicklungsiteration entstehen neue Produktinkremente, also Erweiterungen der bestehenden Funktion des Produkts. Nicht immer wird es sinnvoll sein, nach jeder Iteration eine neue Version an den Kunden auszuliefern. Die Zusammenfassung der Produktinkremente zu tatsächlichen Lieferungen obliegt dem Product Owner. Mit diesen Lieferungen muss der Product Owner natürlich auch die Erwartungen der Stakeholder managen, damit diese nicht unrealistisch sind.
 - **Kein Projektleiter:** der Product Owner ist kein Projektleiter! Diese Rolle gibt es im Scrum-Framework nicht. Obwohl einige Aufgaben der klassischen Projektleiterrolle beim Product Owner liegen, werden andere Aufgabe dieser Rolle auf den Scrum Master und das Entwicklerteam verteilt.

Scrum Rollen – Scrum Master

- übernimmt die Aufgabe, dafür zu sorgen, dass das Team möglichst ohne Störung und so effizient wie möglich seine Arbeit erledigen kann.
- Hauptaufgaben:
 - *Team Coach*: hauptsächlich hilft der Scrum Master dem Scrum-Team, Probleme zu lösen, die im Team entstehen. Er oder sie übernimmt dabei die Rolle des Moderators, Mentors, Trainers und Mediators. Der Scrum Master sorgt dabei dafür, dass das Team effizient und effektiv arbeitet, hilft bei Konflikten innerhalb des Teams und schafft eine förderliche Atmosphäre im Team. Er oder sie unterstützt das Team bei der Selbstreflektion und bei seiner Entwicklung.
 - *Team beschützen*: der Scrum Master schützt das Team vor negativen Einflüssen von außen, wie zum Beispiel ständiges Abziehen von Personen für andere Tätigkeiten, Änderungen von Anforderungen, die schon in Entwicklung sind, Störungen durch Personen von außen, etc. Er oder sie schützt das Team aber auch vor Fehlverhalten des Product Owners und gestaltet die Zusammenarbeit zwischen dem Product Owner und dem Umsetzungsteam möglichst reibungslos.

Scrum Rollen – Scrum Master

- *Barrieren entfernen:* der Scrum Master muss Barrieren entfernen, die das Team daran hindern, effektiv und effizient die geplanten Aufgaben zu erledigen. Diese Barrieren können innerhalb des Teams, zum Beispiel durch mangelnde Fähigkeiten und Kenntnisse oder falsche Einstellung oder außerhalb des Teams, zum Beispiel in Form von organisatorischen Hindernissen entstehen. Er oder sie hilft dem Team, selbstorganisierend zu arbeiten.
- *Führung ohne diszipliniäre Macht:* der Scrum Master „führt“ das Team, in dem er sich um das Wohlergehen und die Motivation des Teams kümmert. Er oder sie hat dabei aber keine diszipliniäre „Macht“ über die Teammitglieder, da das Team selbstorganisiert seine Arbeit erledigt.
- *Veränderungen in der Organisation:* vor allem in Organisationen, in denen noch nicht nach dem Scrum-Prinzip gearbeitet wurde, werden Veränderungen durch die neue Arbeitsweise auftreten (müssen). Bei diesem Veränderungsprozess sollte der Scrum Master unterstützend mitwirken.

Scrum Rollen – Umsetzungsteam

- setzt die vom Product Owner erhobenen und priorisierten Anforderungen um und ist somit für das „Wie?“ in der Entwicklung zuständig
- Hauptaufgaben und Eigenschaften:
 - *Anforderungen in Software umsetzen:* die Hauptverantwortung des Umsetzungsteams ist die Umsetzung der gestellten Anforderungen in qualitativ hochwertige lauffähige Software. Diese Umsetzung erfolgt innerhalb von fest definierten, eher kurzen Iterationen, sogenannten Sprints. Innerhalb eines Sprints entsteht ein lauffähiges Produktinkrement.
 - *Technische Fähigkeiten pflegen:* um die Umsetzung der Software durchführen zu können, müssen im Team alle notwendigen technischen Fähigkeiten vorhanden sein und diese auch laufend gepflegt und auf dem aktuellen Stand gehalten werden.

Scrum Rollen – Umsetzungsteam

- Hauptaufgaben und Eigenschaften:
 - *Funktionsübergreifend (interdisziplinär)*: ein Scrum-Umsetzungsteam ist immer interdisziplinär zusammengesetzt, das heißt im Team befinden sich nicht nur Programmierer, sondern auch Tester, Designer, Architekten und andere notwendige Experten. Optimalerweise sind diese Rollen jedoch nicht an einzelne Personen gebunden, sondern teilen sich so weit als möglich auf alle Team-Mitglieder auf. Dies bedeutet, dass im Idealfall jedes Teammitglied zumindest ein Grundwissen im Bereich der anderen Teammitglieder besitzt.
 - *Selbstorganisation*: innerhalb eines Sprints organisiert das Team sich und seine Arbeit selbst. Außerhalb des Teams existiert kein Manager oder Teamleiter, der den einzelnen Personen im Team sagt, was sie zu tun haben oder wie sie diese Aufgaben erledigen sollen. Das Team entscheidet selbst, wie das Ziel des Sprints am besten erreicht werden kann. Um das Team bei dieser Selbstorganisation zu unterstützen und einen Lerneffekt zu erzielen, werden am Ende jedes Sprints sogenannte Retrospektiven abgehalten, in denen mögliche Verbesserungen identifiziert und für den nächsten Sprint geplant werden können.

Scrum Rollen – Umsetzungsteam

- Hauptaufgaben und Eigenschaften:
 - *Vollzeit und 7 +/- 2 Personen:* Mit wenigen Ausnahmen sollten alle Mitglieder des Umsetzungsteams Vollzeit im Team mitarbeiten. Dadurch kann die Kontinuität aufrechterhalten und Effizienzverlust durch Multitasking verhindert werden. Um eine effiziente Kommunikation im Team zu gewährleisten ist eine eher kleinere Größe des Teams vorteilhaft.
 - *Qualität der Lieferung:* das Umsetzungsteam ist verantwortlich für die Qualität der gelieferten Software.

Scrum Meetings – Sprint planning

- findet jeweils zu Beginn eines Sprints statt
- In diesem Meeting entscheidet das Team, wie viele User Stories im Sprint umgesetzt werden.
- Ebenfalls wird in diesem Meeting das WIE zur Umsetzung festgelegt, also wie die Anforderungen technisch konkret umgesetzt werden sollen.
- Am Ende dieses Meeting gibt jedes Teammitglied sein Commitment zum geplanten Sprintumfang und -ziel ab.

Scrum Meetings – Sprint review

- am Ende des jeweiligen Sprints wird im Sprint review Meeting das Ergebnis des Sprints, also das neue Produktinkrement präsentiert.
- In diesem Meeting sind zumindest das Umsetzungsteam, der Product Owner, der Scrum Master und alle relevanten Stakeholder anwesend.
- Im Sinne der Transparenz, die in den agilen Methoden einen hohen Stellenwert hat, ist der Teilnehmerkreis allerdings nicht vom Framework beschränkt, sondern kann beliebig ausgeweitet werden.
- Dieses Meeting ist auch der Ort, wo dem Team erstes Feedback zum jeweiligen Produktinkrement gegeben werden kann.

Scrum Meetings – Sprint retrospective

- die Retrospektive wird ebenfalls am Ende des Sprints durchgeführt und stellt einen wichtigen Teil des Scrum-Prozesses dar.
- Anwesend sind das Umsetzungsteam, der Scrum Master (als Moderator) und der Product Owner.
- In diesem Meeting stellt sich das Team die Frage, was im abgelaufenen Sprint gut und was eventuell nicht so gut gelaufen ist.
- Dabei hat jedes Teammitglied die gleiche Stellung und das gleiche Sprachrecht.
- Werden Probleme oder Defizite aufgedeckt, werden diese adressiert und Lösungen gesucht, die dann für die Zukunft umgesetzt werden sollen.
- Transparenz, Offenheit und Feedbackkultur spielen hier eine wesentliche Rolle, damit das Team lernen und sich weiterentwickeln kann.

Scrum Meetings – Daily Scrum

- dieses Meeting wird täglich, typischerweise am Beginn des Arbeitstages und in Form eines Standup-Meetings durchgeführt.
- Dieses Meeting sollte nicht länger als 15 bis 30 Minuten dauern.
- Sollten Diskussionen entstehen, die nicht in diesem Zeitrahmen zu lösen sind, werden diese im Anschluss außerhalb dieses Meetings geklärt.
- Im Daily Scrum beantwortet jedes Teammitglied die folgenden Fragen:
 - Woran habe ich gestern gearbeitet?
 - Was ist mein Plan für heute?
 - Gibt es etwas, dass mich daran hindert, meinen Plan einzuhalten?

Scrum Artefakte – Produkt Backlog

- Im **Produkt Backlog** werden in Scrum-Framework die Anforderungen an das zu liefernde Produkt aufgelistet.
- Meist sind diese Anforderungen in Form von User Stories erfasst.
- Ein Feature des Produkts wird dabei aus Sicht des Users beschrieben.
- Wichtig dabei ist, das beschrieben ist, WER WAS WARUM haben möchte
- eine User Story beschreibt, was das Feature beinhalten soll, wer das Feature braucht und warum dieses Feature gebraucht wird.
- Zusätzlich wird beschrieben, welche Akzeptanzkriterien erfüllt sein müssen, damit die Anforderung erfüllt ist.
- Eine User Story beschreibt also immer WAS umgesetzt werden soll und nicht WIE das technisch zu erfolgen hat.

Scrum Artefakte – Produkt Backlog

- Beispiel einer User-Story:

Als User des Webshops möchte ich mich mit E-Mail-Adresse und Passwort einloggen können, damit ich mein Kundenkonto einsehen kann.

Akzeptanz-Kriterien:

- *es wird ein Eingabe-Feld „Email-Adresse“ angezeigt*
- *es wird ein Eingabe-Feld „Passwort“ angezeigt*
- *es wird ein Button „Login“ angezeigt*
- *Bei Klick auf den Button werden die Credentials geprüft*
 - *wenn E-Mail-Adresse und Passwort korrekt → User wird auf das Kundenkonto weitergeleitet*
 - *wenn Eingaben nicht korrekt → es wird eine Fehlermeldung angezeigt*

Scrum Artefakte – Produkt Backlog

Das Produkt Backlog folgt folgenden Grundsätzen:

- ***Das Produkt Backlog ist sortiert:***
 - die Liste der Anforderungen ist immer in der, vom Product Owner gewünschten, Reihenfolge sortiert.
 - Diese Reihenfolge kann sich im Lauf der Entwicklung ändern, falls zum Beispiel neue Anforderungen dazukommen oder sich durch Feedback vom Kunden die Wichtigkeit von gewissen Anforderungen ändert.
- ***Das Produkt Backlog wächst und verändert sich:***
 - durch das, im Scrum Prozess verankerte, regelmäßige Feedback des Kunden können neue Anforderungen dazu kommen oder sich bestehende Anforderungen ändern.
 - Dadurch ist das Produkt Backlog eine „lebende“ sich ständig verändernde Liste.

Scrum Artefakte – Produkt Backlog

Das Produkt Backlog folgt folgenden Grundsätzen:

- ***Das Produkt Backlog enthält nur ausreichend Details:***
 - Die Anforderungen im Produkt Backlog haben immer nur so viele Details, wie im Augenblick gerade benötigt werden.
 - User Stories werden vom Product Owner gemeinsam mit dem Umsetzungsteam im Laufe des Entwicklungsprozesses immer weiter verfeinert, je mehr Erkenntnisse über das Produkt bestehen.
 - Darin besteht ein großer Unterschied zu klassischen Methoden, bei denen am Anfang des Projekts ein Anforderungsdokument in Form eines Lasten- und Pflichtenhefts erstellt wird, in dem alle Anforderungen bereits ausdefiniert sein müssen.
 - In Scrum werden immer nur die Anforderungen bis ins Detail ausdefiniert, die in den nächsten Sprints umgesetzt werden sollen.
 - Somit hat man immer die Möglichkeit aus schon abgearbeiteten Anforderungen zu lernen.

Scrum Artefakte – Produkt Backlog

Das Produkt Backlog folgt folgenden Grundsätzen:

- ***Das Produkt Backlog ist geschätzt:***
 - Alle User Stories, welche in den nächsten Sprints umgesetzt werden sollen (meist zwei bis drei in die Zukunft) sind auf Aufwand geschätzt.
 - Diese Schätzung wird vom Umsetzungsteam vorgenommen.

Scrum Artefakte – Sprint Backlog


- bezeichnet die Anforderungen, die vom Umsetzungsteam im aktuellen Sprint umgesetzt werden sollen
- Dabei werden aus dem Produkt Backlog, beginnend bei der am höchsten Priorisierten, der Reihe nach so viele User Stories in den Sprint Backlog übernommen, wie das Team in einem Sprint schaffen kann
- Wie viele User Stories das sind, hängt vom geschätzten Aufwand der Stories und der durchschnittlichen Geschwindigkeit des Teams (Velocity) über die letzten Sprints ab.
- Arbeitet das Team schon länger zusammen, ist diese Abschätzung, wie viele Anforderungen im Sprint umgesetzt werden können, meist sehr einfach.
- **Wichtig:** die Entscheidung, wie viele Anforderungen im Sprint umgesetzt werden sollen, wird nicht vom Product Owner vorgegeben, sondern vom Umsetzungsteam selbst getroffen

Scrum Artefakte – Produktinkrement

- Am Ende jedes Sprints steht ein potenziell auslieferbares **Produktinkrement**.
- Folgende Eigenschaften haben sich als Minimum für ein auslieferbares Produktinkrement etabliert :
 - *Funktional vollständig umgesetzt*
 - *Getestet und frei von bekannten Fehlern*
 - *Integriert und auf einer realistischen Systemumgebung vorgeführt*
- Dadurch wird in Scrum der gesamte Entwicklungsprozess End-to-end in jedem Sprint in kurzer Zeit durchlaufen.

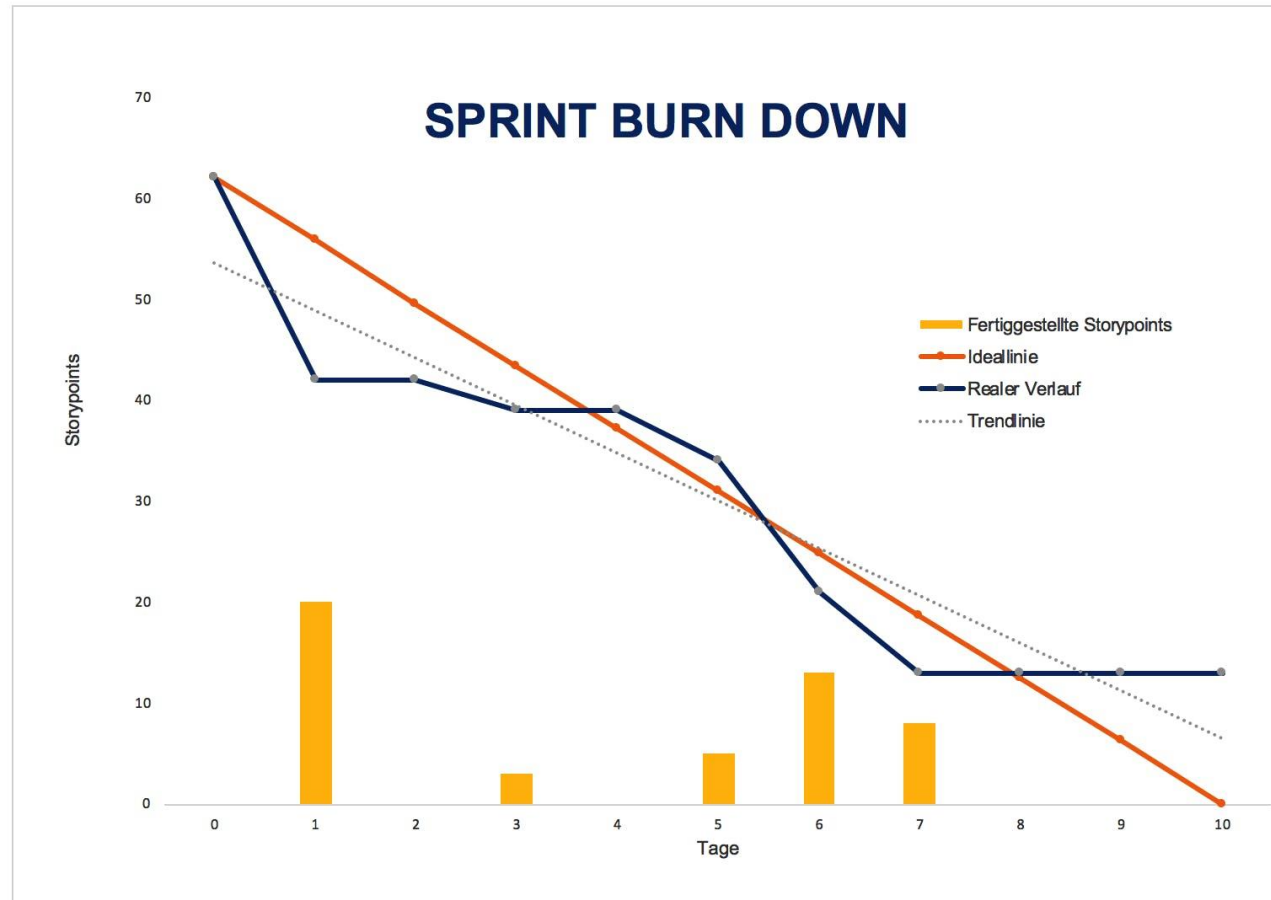
Scrum Fortschrittskontrolle

- Der Projektfortschritt kann in Scrum z. B. folgendermaßen überprüft werden:
 - User-Stories im Backlog → der Kunde oder Auftraggeber kann jederzeit sehen, welche User-Stories schon erledigt und welche noch offen sind
 - innerhalb eines Sprints mittels Scrum-Board → hier wird der Sprint-Backlog mittels User-Stories und Tasks dargestellt

Story / PBI	To Do	In Progress	Done
<div>XX14. As a user I want so that.... 5 SP</div>	<div>Dev.. 5 Hrs</div> <div>Revi... 7 Hrs</div> <div>Test.. 4 Hrs</div>	<div>Analy. 8 Hrs</div>	<div>Design 2 Hrs</div>
<div>XX12. As a user I want so that.... 3 SP</div>	<div>Dev.. 4 Hrs</div> <div>Revi... 3 Hrs</div> <div>Test.. 7 Hrs</div>	<div>Analy. 2 Hrs</div> <div>Dev.. 3 Hrs</div>	
<div>XX21. As a user I want so that.... 8 SP</div> <div> Agile Digest</div>	<div>Dev.. 3 Hrs</div> <div>Revi... 3 Hrs</div> <div>Analy. 2 Hrs</div> <div>Test.. 7 Hrs</div>		

Scrum Fortschrittskontrolle

- Der Projektfortschritt kann in Scrum z. B. folgendermaßen überprüft werden:
 - Burndown Chart → zeigt die Abarbeitung der für den Sprint vorgesehenen Tasks



Feature Driven Development - FDD

- Wurde 1997 von Jeff de Luca (und in weiterer Folge auch in Zusammenarbeit mit Peter Coad) als ein evolutionärer Ansatz zur Umsetzung der agilen Prinzipien entwickelt
- Als **Feature** wird ein Merkmal, eine Funktionalität eines Programms bezeichnet, welche es dem Benutzer ermöglicht, seine fachlichen Aufgaben zu erfüllen (und somit einen Mehrwert schafft)
- In FDD werden Features nach dem **Schema** **<Aktion>** **<Ergebnis>** **<Objekt>** beschrieben – Beispiele:
 - *Erstellen einer Rechnung für einen Auftrag*
 - *Berechne Saldo/Zinsen eines Kontos*
 - *Zeige verfügbare Mitarbeiter in Liste an*
- Features müssen binnen 2 Wochen realisierbar sein → ansonsten muss das Feature weiter zerteilt werden

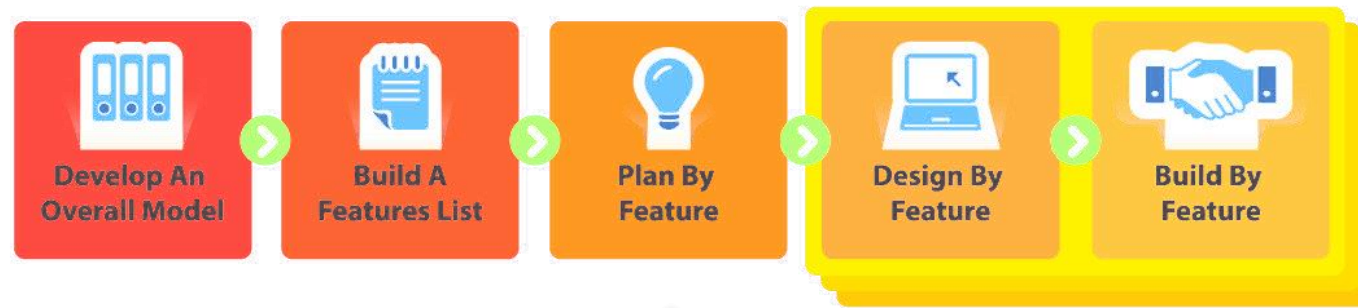
Feature Driven Development - FDD

Definiert **6 zentrale Rollen**:

- **Chief Architect (CA)**: Erfahrener Analytiker und Modellierer, der das Gesamtmodell verantwortet und dessen Entwicklung leitet
- **Chief Programmer (CP)**: erfahrene Entwickler, die an der Anforderungsanalyse und Modellentwicklung mitwirken und bei der Umsetzung ein Team von 3-6 Programmierern leiten
- **Class Owner (CO)**: Entwickler, der für eine bestimmte Klasse und deren Implementierung zur Realisierung der Features verantwortlich ist
- **Domain Experts (DE)**: Fachexperten (oder Business Analysts) jenes Geschäfts-/Organisationsbereichs, für welchen die Software im Rahmen des Projekts entwickelt werden soll; definieren sämtliche Features
- **Project Manager (PM)**: verantwortlich für die (administrative) Projektabwicklung (Budget/Termine, Reporting/Controlling, Ressourcenmanagement, Zieldefinition, ...)
- **Development Manager (optional)**: verantwortet die gesamte laufende (day-to-day) Entwicklung und löst aktuell auftretende Ressourcenkonflikte; Aufgaben können alternativ auch durch PM und CA wahrgenommen werden

Feature Driven Development - FDD

Definiert **5 Prozesse**:



Feature Driven Development - FDD

FDD-Prozess #1: **Entwickle ein Gesamtmodell** (Startup-Phase)

- Fachexperten und Chefprogrammierer definieren unter der Leitung des Chefarchitekten den Systemumfang und –kontext auf grober Ebene
- Das Team (trägt zusammen und) studiert verfügbare Referenz- oder Anforderungsdokumente, funktionale Anforderungen (traditionell oder als Use-Case), Datenmodelle und Richtlinien
- Es folgen detaillierte anwendungsfachliche Darstellungen für jeden Bereich, der modelliert werden soll, in Kleingruppen mit anschließender Review und Diskussion
- Es entsteht ein (erstes/grobes) abstraktes Modell in Form von Klassen- und Sequenzdiagrammen

Feature Driven Development - FDD

FDD-Prozess #2: **Erstelle eine Featureliste** (Startup-Phase)

- Identifikation/Auflistung aller Features, die zu den Anforderungen gehören, durch die Chefprogrammierer
- Dabei wird der **Anwendungsbereich** (Domain, zB „Bank[wesen]“) in **Fachgebiete** (Subject Area, zB „Darlehensbereich“), in darin enthaltene **Geschäftsaktivitäten** (Business Activity, zB „Darlehensbeantragung“) und je Geschäftsaktivität in einzelne Schritte (**Feature**, zB „Besicherung bestimmen“, „Haushaltsrechnung erstellen“, „Bonität prüfen“, ...) strukturiert bzw. zerlegt (= funktionale Dekomposition)

Feature Driven Development - FDD

FDD-Prozess #3: **Plane je Feature** (Startup-Phase)

- Der Projektmanager, der Entwicklungsmanager und die Chefprogrammierer planen die Reihenfolge, in der die Features implementiert werden sollen, basierend auf Feature-Abhängigkeiten, der Arbeitsbelastung im Entwicklungsteam und der Komplexität der Features => Es entsteht ein Plan für die Umsetzung
- Jede Geschäftsaktivität wird einem Chefprogrammierer zugeordnet
- (Schlüssel)Klassen werden Entwicklern (Class Owner) zugewiesen (Erinnerung: Chefprogrammierer sind auch Entwickler)
- Für jede/-s Geschäftsaktivität/Fachgebiet wird ein Fertigstellungstermin festgesetzt (Terminplan)

Feature Driven Development - FDD

FDD-Prozess #4: **Entwurf je Feature** (Construction-Phase)

- Je Feature wird ein Feature-Entwurfspaket (Design-Package) erstellt
- Aufgrund der Priorisierung und Terminisierung aus FDD-Prozess #3 wird eine Menge von Features für die Entwicklung in der nächsten Iteration (immer 2 Wochen) eingeplant
- Chefprogrammierer leiten dabei sog. Feature-Teams, die sich aus jenen Entwicklern zusammensetzen, die für die Umsetzung der beteiligten/notwendigen Klassen verantwortlich sind
- Die Klassendiagramme (aus FDD-Prozess #1) werden ebenso verfeinert (bspw. alle Attribute und Methoden genau definiert) wie die Sequenzdiagramme für die zugewiesenen Features
- Entwickler „schreiben“ (= lassen sich durch die Entwicklungswerkzeuge generieren) Klassen- und Methodenrumpfe
- Im Rahmen einer Entwurfsinspektion (Design Inspection) erfolgt eine Prüfung und die Freigabe zur Codierung

Feature Driven Development - FDD

FDD-Prozess #5: **Konstruiere je Feature** (Construction-Phase)

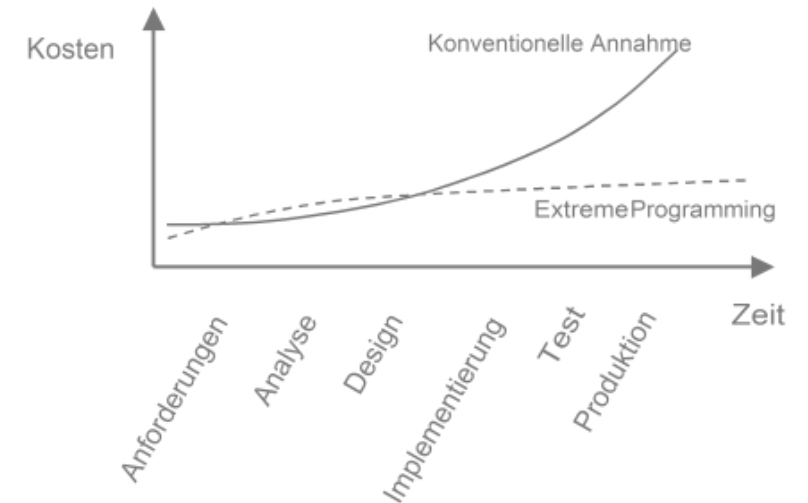
- Je Feature (und auf Basis des zuvor erstellten Entwurfspakets) implementieren die Klassenbesitzer, die Anteile ihrer Klassen, die für das Feature notwendig sind
- Für den entwickelten Code werden dann Unit-Tests geschrieben und ausgeführt und der Code wird inspiziert (Code Inspection)
- Nach erfolgreicher Code-Inspektion wird der Code für den Build freigegeben

Feature Driven Development - FDD

- Die FDD-Prozesse #1-3 werden einmalig im Rahmen der sog. **Startup-Phase** (Dauer idR: 2-3 Wochen - FDD will also „Just Enough Design Initially“ und nicht „Big Design Up-Front“ liefern)
- Die tatsächliche Dauer der Startup-Phase und deren Ergebnisse, ermöglichen idR eine gute Schätzung der gesamten Projektkosten/-dauer (und somit uU auch die Vereinbarung eines Fixpreises)
- Die FDD-Prozesse #4-5 werden in Iterationen von max. 2 Wochen immer wieder durchlaufen – man bezeichnet dies als **Construction-Phase**
- **Fortschrittskontrolle** wird anhand der Anzahl der fertiggestellten Features durchgeführt

XP – Extreme Programming

- Entwickelt von **Kent Beck**
- Stellte damals einen revolutionären Ansatz der Softwareentwicklung dar
- Versucht die (zunehmenden) **Änderungskosten** über die Projektlaufzeit **gering** zu **halten** (logarithmischer Verlauf im Vergleich zu einem exponentiellen Verlauf bei klassischen Vorgehensmodellen wie bspw. Wasserfallmethode)
- Agilität/Flexibilität bezgl. Anforderungen, kontinuierliche/direkte Kommunikation mit dem Kunden, selbstorganisierende Teams und kurze Iterationszyklen spielen hier ebenso eine wichtige Rolle wie in Scrum – dabei liegt der Fokus vor allem auf den **Engineering Practices** (während Scrum eher Management und Organisation in den Mittelpunkt stellt)



XP – Extreme Programming

Vorgehen:

- Das SWE-Projekt wird in **Releases** (Funktionen, die insgesamt und für sich geschlossen die Bereitstellung einer neuen Version des Produktes rechtfertigen) unterteilt
 - Der Kunde formuliert¹⁾ (mit Hilfe des Teams) die zu realisierenden **User Stories** – (für) diese werden:
 - geschätzt (Planning Poker) und priorisiert (Nutzen/Wert zu Aufwand/Risiko²⁾) – in XP ist diese Reihenfolge iGgs zu Scrum (strikt) einzuhalten
 - **Akzeptanzkriterien** für die spätere Abnahme definiert

XP – Extreme Programming

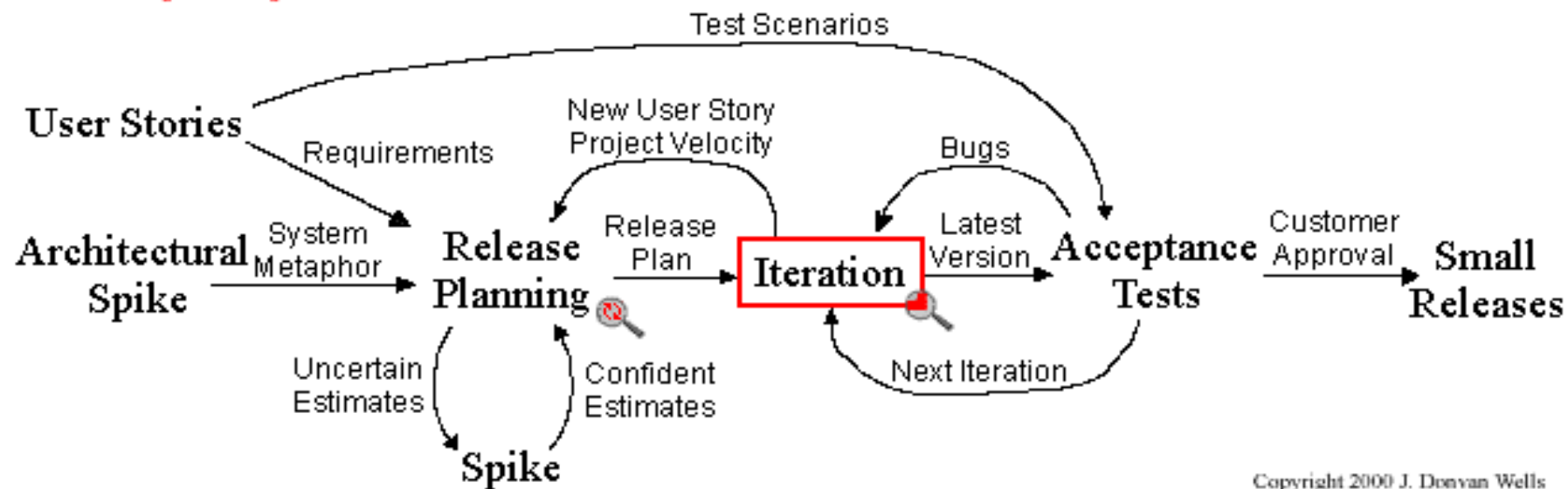
Vorgehen:

- Die Entwicklungsarbeit wird in kurzen Iterationen (1-2, max. 4 Wochen, wird zu Beginn des Projekts einmalig festgelegt – tendenziell kürzer als in Scrum) organisiert – pro **Iteration**:
 - werden zu Beginn User Stories (aus der Releaseplanung) ausgewählt und in **Tasks** (s. Scrum) unterteilt
 - Für die jeweiligen Akzeptanzkriterien werden vor Beginn der eigentlichen Codierung **Unit-Tests** geschrieben (TDD ... Test-Driven Development)
 - Der Fortschritt wird in täglich stattfindenden **Standup-Meetings** besprochen (vgl. Daily Scrum)
 - Am Ende müssen alle Unit-Tests und auch alle Akzeptanzkriterien erfüllt sein
 - In XP kann es iGgs zu Scrum auch während einer Iteration zu Änderungen (bspw. im Architekturmodell oder den User Stories) kommen

XP – Extreme Programming



Extreme Programming Project



Copyright 2000 J. Donovan Wells

XP – Extreme Programming

Rollen:

- (Entwickler-) **Team**:
 - Muss über **breites technisches Fachwissen** (Architektur, Codierung, Datenhaltung, UI-Design, Test, ...) verfügen
 - Jedes Teammitglied soll in der Lage sein, Tätigkeiten aller anderen zu übernehmen – es soll eine **collective code ownership** entstehen
- **Kunde**:
 - Wird repräsentiert durch einen „Delegierten“ des Auftraggebers, der in engem/ständigem Kontakt mit dem Team steht, Anforderungen definiert und priorisiert und die Umsetzung/Ergebnisse überprüft (Feedbacks, Akzeptanzkriterien, Reviews, ...)

XP – Extreme Programming

Rollen:

- Weitere (mögliche) Rollen:
 - **Coach**: hilft dem Team die XP-Praktiken richtig anzuwenden, um eine bestmögliche Performance zu erreichen und achtet auf die Einhaltung der Prinzipien und Werte
 - **Projektmanager**: übernimmt klassisches Projektmanagement

XP – Extreme Programming

Grundwerte:

- Planning
 - User Stories sind schriftlich
 - Im Release planning Meeting wird festgelegt, was in der Iteration umgesetzt wird
 - Mache öfters kleine Releases
 - Das Projekt wird in kurze Iterationen unterteilt
- Managing
 - Das Team bekommt einen eigenen gemeinsamen Arbeitsraum
 - Die Arbeitsgeschwindigkeit muss so gewählt werden, dass sie auf längere Zeit durchhaltbar ist
 - Jeder Tag startet mit einem Standup Meeting
 - Die Umsetzungsgeschwindigkeit (Velocity) wird ständig gemessen
 - Teammitglieder wechseln ihre Rollen, um Knowhow-Engpässe zu vermeiden
 - Falls der Prozess nicht funktioniert, wird er repariert

XP – Extreme Programming

Grundwerte:

- Designing
 - Mache alles möglichst einfach
 - Suche eine System-Metapher
 - Verwende CRC-Karten für Design-Sessions
 - Baue einfache Prototypen (sog. Spike solutions), um technische Entscheidungen mit weniger Risiko treffen zu können
 - Keine Funktionalität wird gebaut, bevor sie gebraucht wird
 - Die Software wird refactored wann immer dies notwendig und möglich ist

XP – Extreme Programming

Grundwerte:

- Coding
 - Der Kunde ist ständig für Fragen verfügbar
 - Programmcode folgt vereinbarten Standards
 - Der Unit-Test wird immer zuerst entwickelt
 - Programmcode für die Produktion wird immer im Pair-Programming entwickelt
 - Nur ein Entwicklerpaar auf einmal integriert neuen Code ins Gesamtsystem
 - Integriere oft
 - Setze einen dedizierten Integration-Computer auf
 - Verwende „collective ownership“

XP – Extreme Programming

Grundwerte:

- Testing
 - Der gesamte Code hat Unit-Tests
 - Der gesamte Code muss alle Unit-Tests passieren, bevor er released werden kann
 - Wenn ein Fehler gefunden wird, werden dafür Tests erstellt
 - Akzeptanz Tests werden oft durchgeführt und die Testergebnisse werden veröffentlicht

Kanban

Geschichte:

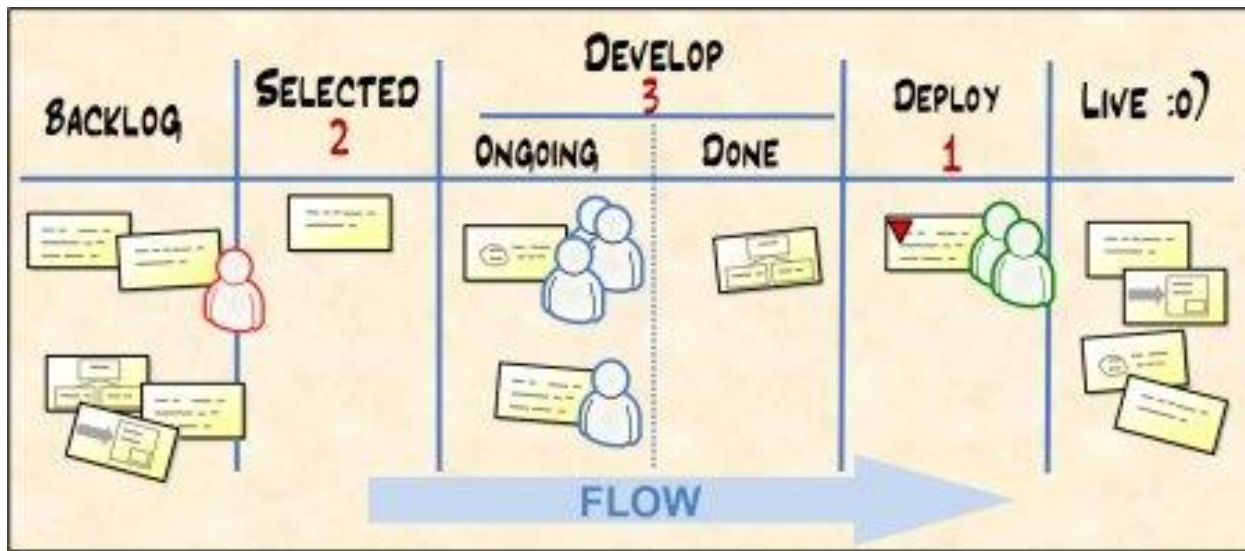
- **Kanban** (japanisch für Signal[karte]) ist ein in Japan (ursprünglich 1947 von Taiichi Ohno in der japanischen Toyota Motor Corporation) entwickeltes System zur flexiblen, dezentralen Steuerung von Produktionsprozessen nach dem **Pull-Prinzip**, dh die Materialbereitstellung orientiert sich hierbei ausschließlich am Bedarf einer verbrauchenden Stelle im Produktionsablauf (also keine Produktion auf Vorrat, sondern Just-In-Time)



Kanban

- Propagiert einen schlanken Produktionsprozess (**lean production**) und versucht **Verschwendung** (japanisch: muda) zu **minimieren** – dazu zählen:
 - Überproduktion
 - Lager(kosten)
 - Wartezeiten
 - Bewegung von Mitarbeitern/Teilen/Maschinen im/zwischen Prozess/-en
 - Korrekturen/Nacharbeiten/Defekte
 - Mehrleistung (Over-Processing, Featuritis), die Kunde nicht benötigt/honoriert
- wird oft ergänzt durch Maßnahmen zur **kontinuierlichen Verbesserung (Kaizen** - jap. kai „Veränderung, Wandel“, zen „zum Besseren“ => „Veränderung zum Besseren“) - diese manifestieren sich bspw auch im **PDCA**-Zyklus (Plan-Do-Check-Act) oder im **DMAIC**-Zyklus (Define-Measure-Analyze-Improve-Control) der 6-Sigma-Methode im Bereich des Qualitätsmanagements

Kanban - Adaption f. d. Vorgehen in SW-Projekten nach D. J. Anderson, 2007



Kanban-Regeln

- ✓ Pull statt Push
- ✓ Visualisierung der Arbeitsabläufe und Probleme durch Verwendung eines Kanban-Boards
- ✓ Work-in-Progress ist beschränkt (WiP-Limit je Lane)
- ✓ Erhebung/Protokollierung von Prozessdaten (bspw cycle-time = Fertigstellungsdauer einer Task)

3. Foliensatz

RUP – Rational Unified Process

- **Fakten & Hintergrund:**
 - Entwickelt bei **Rational Software** (seit 2003 verkauft an IBM) und **1998** erstmals vorgestellt
 - Wesentliche Entwicklung erfolgte durch Ivar Jacobson, Grady Booch und Jim Rumbough (The Unified Software Development Process, 1999), die parallel die UML (Unified Modeling Language) entwickelten
 - **UML** = grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von SW-Systemen aus verschiedenen Perspektiven (Struktur, Verhalten, ...)
 - Mittlerweile gibt es verschiedene „Varianten“ (Open UP, Agile UP, ...)

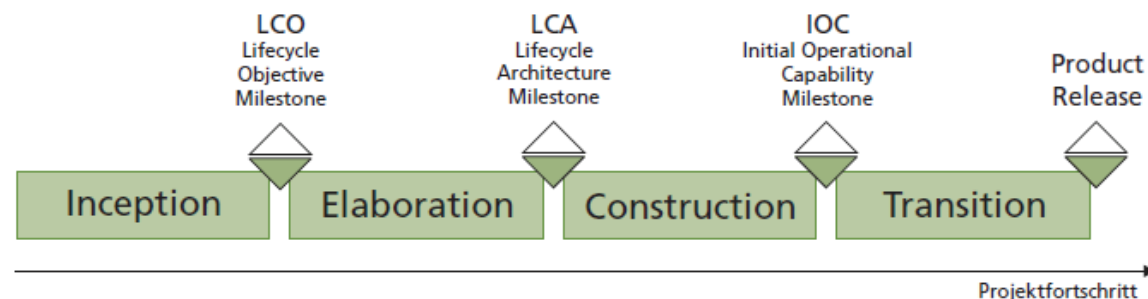
RUP – Rational Unified Process

- Merkmale:
 - Der (Rational) Unified Process ist ein **Metamodell** (also ein Modell mit dem man unterschiedliche Modelle beschreiben kann) **für Vorgehensmodelle zur SW-Entwicklung**
 - Ist **Architektur-zentriert**, basiert auf UML und ist somit **objektorientiert**
 - Baut auf **Use-Cases** zur Anforderungsbeschreibung/-analyse
 - Besteht aus **4 Phasen** (an deren Ende jeweils ein inhaltlich klar definierter Meilenstein steht) und **9 Workflows**
 - Jede Phase besteht (idR) aus mehreren **Iterationen**, in denen die Workflows in unterschiedlicher Intensität durchlaufen werden und an deren Ende ein **Build** (=lauffähiges Produktinkrement) vorliegt

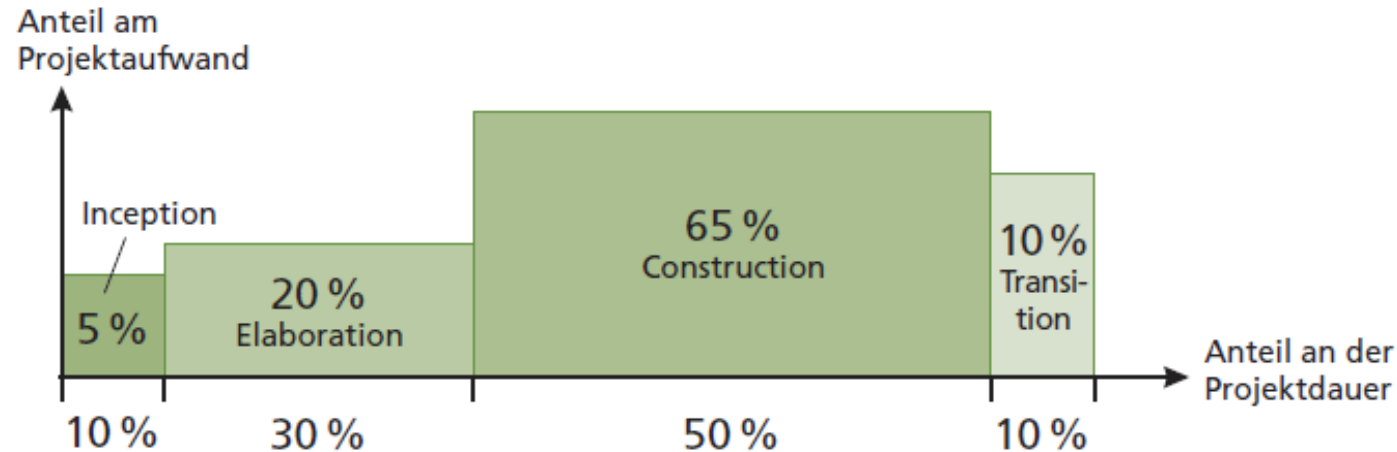
RUP – Rational Unified Process

- **Phasen:**

- **Inception:** Definition des Projektumfangs (Projektabgrenzung), Definition der wichtigsten Anforderungen (~20% aller Use Cases), (erste) Risikoidentifizierung/-analyse, Wirtschaftlichkeitsanalyse (Business Case), (grober) Termin-/Ablaufplan, (erste) Architekturprototyp(en), Machbarkeitsprüfung
- **Elaboration:** Erfassung/Ausarbeitung der (Mehrzahl an) Use-Cases ($\geq 80\%$), Erstellung und Validierung der SW-Architektur (inkl. Prototypen), detaillierte Risikoanalyse, Planung der Iterationen (Beschreibung, Rollen/Verantwortlichkeiten/Artefakte/Ergebnisse, Ziele, Dauer/Termine)
- **Construction:** Erstellung der lauffähigen Produktinkremente (Builds) inkl. Tests, Erstellung der Dokumentation (techn. Dokumentation, Benutzerhandbücher, ...)
- **Transition:** Übergabe/Installation/Konfiguration an/beim Kunden, Funktions-/Leistungsabnahme => final build/release



RUP – Rational Unified Process



- Die Inception-Phase soll bewusst kurz gehalten werden (keine exzessive up-front Spezifikation!).
- Die tatsächliche Dauer der Inception- (und Elaboration-) Phase kann als Basis für eine Projektion zur Berechnung der Gesamtprojektdauer verwendet werden

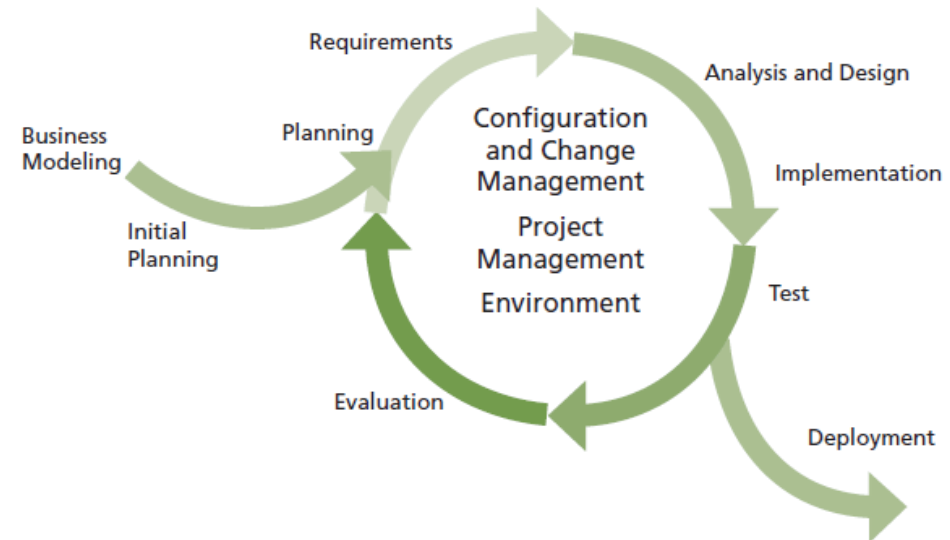
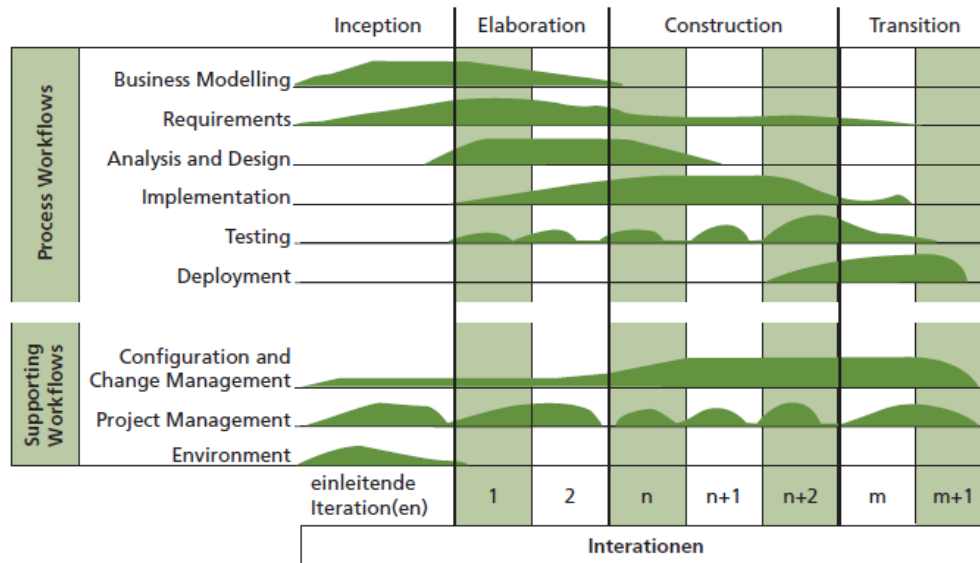
RUP – Rational Unified Process

- **Process Workflows:**
 - **Business Modeling:** beschreibt das betriebliche Umfeld und v.a. jene Geschäftsprozesse und Organisationsbereiche (SOLL/IST), die das (zu entwickelnde) SW-System unterstützen/realisieren soll
 - **Requirements:** Beschreibung der Anforderungen in Form von Use-Case-Diagrammen (= Anwendungsfalldiagramm)
 - **Analysis & Design:** Erstellung eines „Bauplans“ der Software aus unterschiedlichen Perspektiven
 - Struktur bspw anhand von Klassen-, Komponenten, Paket- und Verteilungsdiagrammen
 - Verhalten bspw anhand von Anwendungsfall-, Zustands-, Aktivitäts- und Sequenzdiagrammen
 - **Implementation:** Erstellung lauffähiger Software, iterativ in Form lauffähiger Produktinkremente (sog. Builds) inkl. Tests
 - **Test:** Planung, Design, Implementierung und (falls möglich automatisierte) Durchführung verschiedener Tests (Unit-Tests, Integrations- und Systemtests)
 - **Deployment:** Erstellung, Verteilung und Installation/Konfiguration von Builds/Releases

RUP – Rational Unified Process

- **Supporting Workflows:**
 - **Configuration & Change Management:** Umgang mit Change Requests, Erstellung von Baselines (= abgesicherter Ausgangspunkt vor Änderung), Versionierung, ...
 - **Project Management:** umfasst klass. PM-Aufgaben wie Controlling/Monitoring, Risikomanagement, Planung, ...
 - **Environment:** qualitätssichernde Maßnahmen, Reflexion, „optimale“ Entwicklungsumgebung/-werkzeuge, ...

RUP – Rational Unified Process

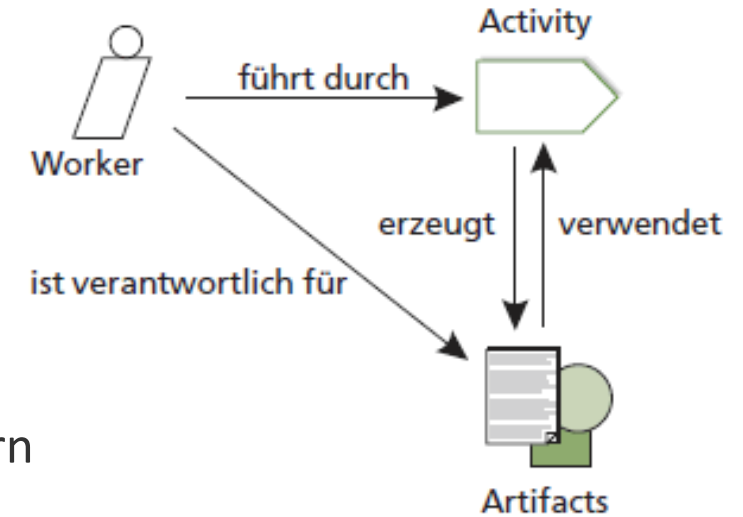


- Iteration dauern idR 2-3 Wochen
- hierbei werden jeweils die Workflows des RUP durchlaufen
- Eine Iteration endet mit dem Deployment eines lauffähigen Produktinkrements (Build).
- Zur Bestimmung der Anzahl an Iterationen je Phase kann die 6-plus/minus-3-Regel verwendet werden, d.h. für mittlere Projekte 6 Iterationen (1+2+2+1), für kleine Projekte 3 Iterationen (0+1+1+1) und für große Projekte 9 Iterationen (1+3+3+2)

RUP – Rational Unified Process

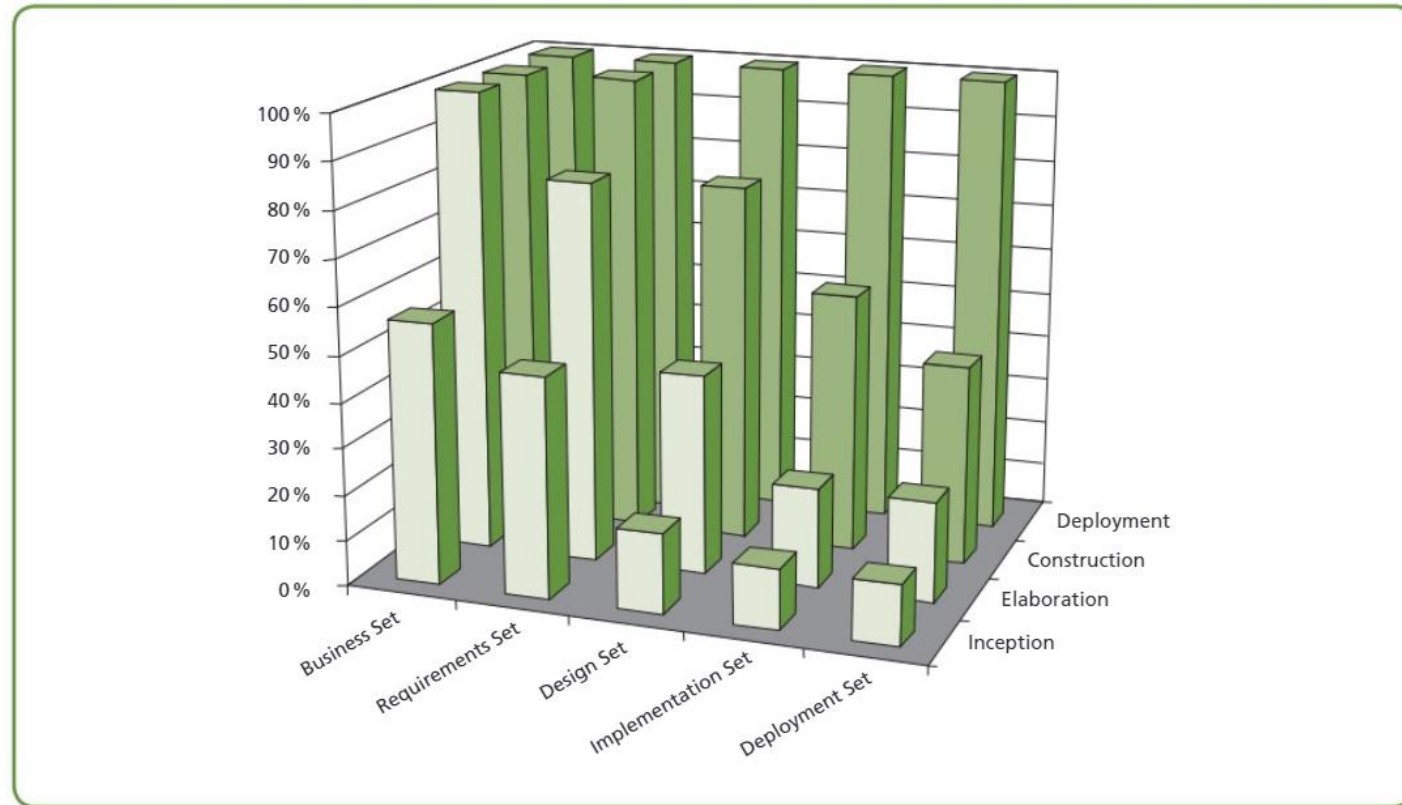
Bei der detaillierten Beschreibung der Workflows werden folgende Elemente beschrieben:

- **Worker**
 - Personen, die im Prozess eine bestimmte Aktivität durchführen
- **Activity**
 - kann mit einem Arbeitspaket verglichen werden
 - muss vollständig ausgeführt werden, um ein sinnvolles Ergebnis zu liefern
 - am Ende jeder Aktivität steht ein Artefakt
- **Artifact**
 - beschrieben als „...Teil an Information, das produziert, modifiziert oder vom Prozess genutzt wird...“
 - kann ein Modell, ein Modellelement oder ein Dokument sein
 - Bsp: Architekturdiagramm, Screendesigns, ...
 - alle Artefakte eines Prozesses werden einem Set zugeordnet (zb. Design-Set)



RUP – Rational Unified Process

Während des Projektverlaufs vervollständigen sich die einzelnen Sets, entsprechend den Projektphasen:



4. Foliensatz

Systemanalyse und Anforderungen

- Am Beginn der meisten SW-Projekte stehen eine Erhebung und Analyse des IST-Zustandes
- danach müssen den Anforderungen an die neue SW erhoben werden
- die erhobenen Anforderungen bilden dann die Grundlage für eine Kosten- und Aufwandsschätzung
- Erhebungsmethoden für Anforderungsanalysen sind zum Beispiel:
 - Interview
 - Fragebogen
 - Beobachtung
 - Selbstaufschreibung
 - Dokumentenauswertung
 - CRC-Karten

Interview

- dient der Gewinnung von **qualitativen Informationen** und der Etablierung einer konstruktiven Zusammenarbeit
- **Vorteile:**
 - Möglichkeit von Zusatzfragen zur Vertiefung und zum besseren Verständnis
 - Möglichkeit zur Steigerung der Motivation des Befragten
- **Nachteile:**
 - hoher Zeitaufwand
 - hohe Anforderung an die Qualifikation des Interviewers
 - Störung der/des Interviewten bei der Aufgabenerfüllung
- vor dem Interview müssen einige Entscheidungen getroffen werden:
 - wie soll die Untersuchungssituation geschaffen werden? → Einzel- oder Gruppeninterview?
 - wie sollen die Informationen gewonnen werden? → welche Art von Fragen sollen gestellt werden?
 - wie sollen die Informationen dokumentiert werden?
 - wie sollen die Daten am Ende ausgewertet werden können? → händisch, automatisch, strukturiert, nicht strukturiert?
 - welche Stellung haben die Befragten im Unternehmen?

Interview - Interviewpartner

- vor dem Interview wird entschieden, wer aller daran beteiligt sein soll
- Interviewer → wer soll das Interview führen?
- Anzahl der Interviewten
 - Einzelinterview → es wird immer nur eine Person auf einmal befragt
 - Gruppeninterview → es werden mehrere Personen gleichzeitig befragt
 - Einzelinterview ist aus folgenden Gründen besser:
 - heikle Fragen werden in der Gruppe eher ungern beantwortet
 - es können Meinungsverschiedenheiten zwischen den Befragten entstehen
 - sind die formellen oder auch informellen Führer anwesend, kann es schwierig werden, offene und ehrliche Antworten zu erhalten

Interview - Bewegungsspielraum

- wie standardisiert ist das Interview?
- standardisiertes Interview:
 - die Fragen sind fix vorgegeben und werden allen Interviewten in der selben Art und Reihenfolge gestellt
 - zusätzliche Fragen sind nicht gestattet
 - sehr schwierig, wenn
 - nicht alle Befragten das gleiche Vokabular verstehen (z. B.: Ärzte und medizinische Laien)
 - für den Untersuchungsgegenstand kein einheitliches Vokabular existiert, dh. Begriffe erklärt werden müssten
- gut geeignet
 - für die Erhebung von quantitativen Daten (z. B.: Alter, Anzahl Besuche auf unserer Website, Häufigkeit von sportlicher Betätigung, ...)
 - wenn Interviewer und Befragter einheitliche Begrifflichkeiten verwenden

Interview - Bewegungsspielraum

- halb-standardisiertes Interview:
 - die Fragen sind grundsätzlich vorgegeben können aber an die Gegebenheiten angepasst werden
 - zusätzliche Fragen können gestellt werden
 - die Reihenfolge der Fragen kann geändert werden
 - Fragen können auch weggelassen werden
- nicht standardisiertes Interview:
 - der Interviewer bekommt keine Fragen vorgeschrieben, sondern einen Katalog an Informationen, die durch das Interview erhoben werden sollen
 - die Art der Fragestellung ist dem Interviewer komplett freigestellt

Interview – Beziehung zum Befragten

- welche Art der Beziehung zwischen Interviewer und Befragten wird angewandt?
- hartes Interview
 - der Interviewer ist betont sachorientiert und autoritär
 - es wird angenommen, dass der Befragte nicht die Wahrheit sagt und deswegen ständig Druck ausgeübt
 - Fragen werden in schneller Abfolge gestellt, damit der Befragte keine Zeit hat, nachzudenken
 - Bsp: Vernehmung bei der Polizei
- weiches Interview
 - es wird ein freundschaftliches Vertrauensverhältnis zum Befragten aufgebaut
 - der Befragte bestimmt den Ablauf der Interviews
- neutrales Interview
 - liegt zwischen reiner Sachorientierung und zu persönlicher Befragung
 - es soll eine angenehme Atmosphäre erzeugt und kein Druck ausgeübt werden
 - es soll jedoch nicht zu persönlich werden
- bei der Anforderungserhebung wird das neutrale Interview vorherrschen

Interview – Frageformen

- wie wird gefragt?
- unterschieden wird grundsätzlich in zwei Typen
 - offene Fragen
 - geschlossene Fragen
- offene Fragen:
 - es werden keine Antwortmöglichkeiten vorgegeben
 - Bsp: „Beschreiben Sie in eigenen Worten ihren Tagesablauf.“
 - eignen sich sehr gut, wenn
 - nicht alle Antwortalternativen bekannt sind
 - der Interviewer weniger Informationen besitzt, als die Befragten
 - in Organisationsaufgaben werden diese vor allem bei den Voruntersuchungen eingesetzt

Interview – Frageformen

- geschlossene Fragen:
 - die Antwortmöglichkeiten sind vorgegeben
 - folgende Subtypen:
 - Identifikationstyp → es sollten konkrete Fakten wiedergegeben werden
 - Bsp: Wieviele Stück werden am Tag produziert? Wer erteilt Ihnen Anweisungen?
 - Selektionstyp → Antwort wird aus vorgegebenen Alternativen gewählt
 - Bsp: Welchen Onlineshop nutzen Sie am häufigsten?
 - Amazon
 - Shöpping.at
 - Kaufhaus Österreich
 - e-tec
 - ja/nein – Typ → nur ja/nein Antworten möglich
 - Bsp: Haben Sie schon einmal bei Amazon eingekauft?

Interview – Frageformen

- geschlossene Fragen:
 - Vorteile von geschlossenen Fragen:
 - Einheitlichkeit der Antworten
 - können besser ausgewertet werden (vor allem automatisiert)
 - Befragte werden nicht überfordert
 - genauere quantitative Angaben
 - Sinn der Frage kann aus den Antworten hervorgehen
 - werden eher beantwortet
 - Nachteile von geschlossenen Fragen:
 - unter Umständen trifft keine der Antwortmöglichkeiten zu, es muss jedoch eine gewählt werden
 - Antworten werden beeinflusst
 - unterschiedliche Interpretationen der Fragestellung werden nicht berücksichtigt

Interview – Frageformen

- unabhängig von offen oder geschlossen können Fragen noch in direkte und indirekte Fragen unterteilt werden
- direkte Fragen:
 - der Tatbestand wird direkt angesprochen
 - Bsp: „Ist unser Webshop gut gemacht?“
- indirekte Fragen:
 - die gewünschte Information wird nicht direkt abgefragt
 - Bsp: „Konnten Sie in unserem Webshop Ihre gewünschten Einkäufe tätigen?“
 - indirekte Fragen sind gut bei heiklen Themen, wie z. B:
 - Beurteilung der Qualifikation von Mitarbeitern oder Vorgesetzten
 - Beurteilung des Betriebsklimas

Interview – Grundsätze für Fragestellungen

- einfache Formulierungen verwenden
- eindeutige und präzise Formulierungen
- keine Überforderung des Befragten
- wenn möglich an konkrete Erfahrungen der Befragten anknüpfen
- Vermeidung von Suggestivfragen
- Kontrollfragen einbauen

Interview – Phasen

- Einleitungsphase
 - dient der Herstellung eines günstigen Gesprächsklimas
 - Vorstellung der Beteiligten
 - Erläuterung des Interviewzwecks
- Informationsgewinnungsphase
 - Durchführung der Befragung
- Ausklangphase
 - dient dem weiteren Aufbau von positiven Beziehungen zu den Befragten
 - gibt eventuell Ausblick auf die weiteren Aktivitäten

Interview – Interviewort

- wo wird das Interview durchgeführt?
- der Ort des Interviews sollte für den Befragten möglichst angenehm sein
- bei Organisationsaufgaben können Interviews am Arbeitsplatz des Befragten durchgeführt werden
 - Vorteile:
 - der Ort ist dem Befragten vertraut
 - der Interviewer lernt den Arbeitsplatz kennen und kann so zusätzliche Informationen gewinnen
 - Nachteile:
 - mögliche Anwesenheit anderer Mitarbeiter
 - möglicherweise fühlt sich der Mitarbeiter am Arbeitsplatz unwohl
 - nicht alle Arbeitsplätze sind für Interviews geeignet (z.B.: Werkshalle)

Interview – Dokumentation

- Interviews können entweder schriftlich oder audiovisuell festgehalten werden
- dies kann erfolgen
 - nachträglich → Gedächtnisprotokoll
 - schriftlich während des Interviews
 - Aufzeichnung mittels Diktiergerät, Kamera, ...

Fragebogen

- ein Fragebogen ist im Grunde ein Interview ohne Interviewer
- dies bewirkt, dass
 - die Befragungssituation nicht beeinflusst werden kann
 - die Befragten nicht zur Mitarbeit motiviert werden können
 - Unklarheiten nicht beseitigt werden können
- ein Fragebogen entspricht dem standardisierten Interview
- beim Einsatz eines Fragebogens sollte immer ein sogenannter Pre-Test mit einer kleinen Testgruppe durchgeführt werden, um den Fragebogen abzutesten
- beim Einsatz in der Anforderungserhebung bietet sich der kombinierte Einsatz von Interview und Fragebogen an
 - Erfassung der grundsätzlichen Gegebenheiten des Ist-Zustandes mittels Fragebogen
 - Ergänzung und Vertiefung der Informationen durch den Einsatz von Interviews
- Vorteile Fragebogen:
 - schriftliche Ergebnisse
 - geringe Kosten
- Nachteile Fragebogen:
 - mögliche Missverständnisse
 - bei offenen Fragen mühsame Auswertung

Beobachtung

- mittels Beobachtung können Informationen aus Befragungen überprüft und eventuell ergänzt werden
- dient der Erhebung bestimmter Erkenntnisse → ich muss wissen, was ich beobachten will
- ist systematisch geplant und nicht vom Zufall abhängig
- wird systematisch dokumentiert
- kann einer Prüfung und Kontrolle hinsichtlich Gültigkeit und Verlässlichkeit unterzogen werden
- durch Beobachtung können nur Erkenntnisse über tatsächliches Verhalten gewonnen werden → keine Erkenntnisse darüber, wie das Verhalten sein sollte
- **Vorteile:**
 - Erfassung der Daten ist unabhängig von der Fähigkeit und Bereitwilligkeit der beobachteten Personen
 - Objektive Erfassung der Daten ist möglich
- **Nachteile:**
 - Informationen über Sinnzusammenhänge, Ursachen, Zielsetzungen, Motivation können nicht erhoben werden
 - unter Umständen hoher Zeitaufwand (= hohe Kosten)
 - wirklich aussagekräftige Information wäre manchmal nur mit verdeckter Beobachtung zu erheben → diese ist allerdings arbeitsrechtlich nicht zu rechtfertigen

Selbstaufschreibung

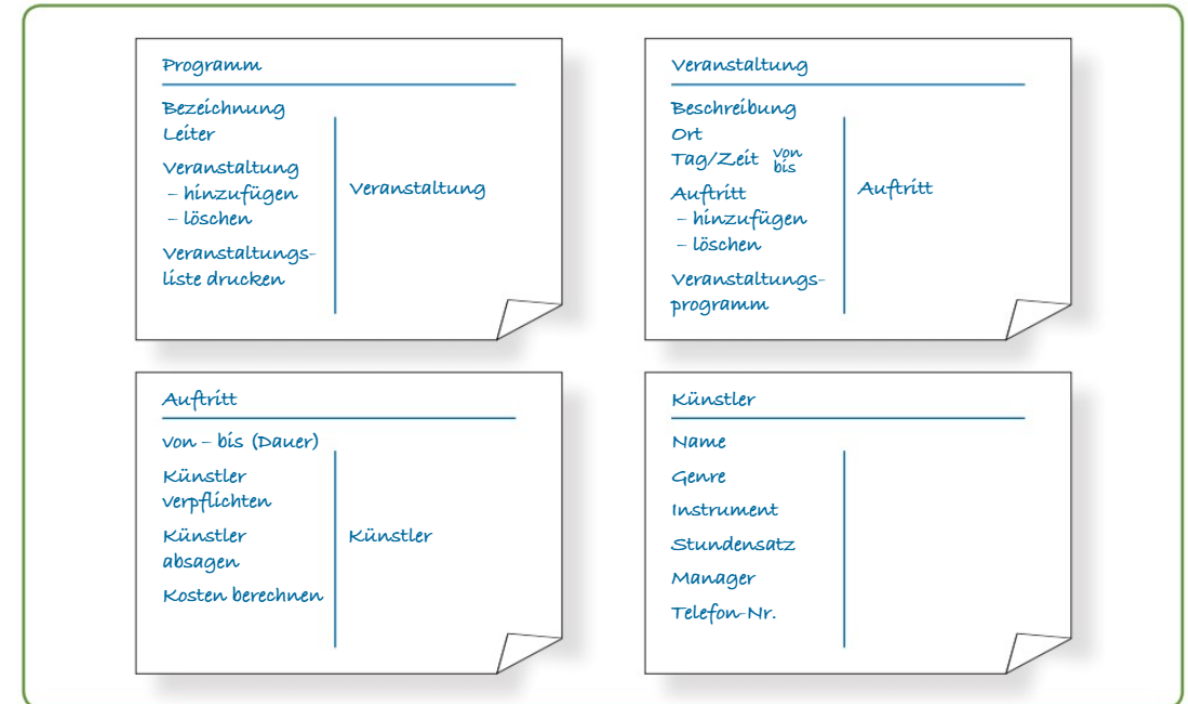
- strukturierte Aufzeichnung relevanter Ereignisse und Handlungen
- dient Erfassung des Ist-Zustands durch die Aufgabenträger
- erfolgt ohne direkte Mitwirkung des Systemplaners
- folgende Schritte:
 - Aufnahmefestlegung
 - Aufnahmevorbereitung
 - Mitarbeiterinformation
 - Durchführung
 - Auswertung
- Vorteile:
 - Möglichkeit der Totalaufnahme
 - Entlastung des Systemplaners
- Nachteile:
 - Möglichkeit der bewussten Verfälschung
 - Widerstände vonseiten des Aufgabenträgers

Dokumentenauswertung

- meist leicht verfügbare Informationsquelle
- Relevanz und Aktualität sollte immer hinterfragt werden
- die Information kann unter anderem aus folgenden Quellen kommen:
 - Dokumentation des bestehenden Systems
 - bestehende Unterlagen über das System sollen die Erhebung des Ist-Zustandes erleichtern
 - Dokumente müssen allerdings aktuell, vollständig und noch mit dem tatsächlichen Ist-Zustand konsistent sein
 - Vorhandene Studien über das bestehende System
 - eventuell wurden zu früherer Zeit schon Lösungsalternativen für Probleme bzw. Konzepte für Erweiterungen erstellt aber dann nicht umgesetzt
 - solche Unterlagen können Informationen für Use-Cases und Problemdefinitionen liefern
 - Belege und Formulare
 - es sollen alle Belege und Formulare im zu untersuchenden Bereich erhoben werden
 - diese dienen dann als Ausgangsbasis für Anforderungen bezüglich Ein- und Ausgaben des Systems

CRC-Karten

- CRC ist die Abkürzung für Class Responsibility Collaboration
- als Hilfsmittel bei dieser Methode werden einfache Karten verwendet
- Ziel: Aufbau eines Klassenmodells
- jede Klasse soll mindestens durch ihre Verantwortlichkeiten und Kooperationen beschrieben werden
 - Verantwortlichkeiten → welche Attribute und Methoden soll die Klasse abdecken?
 - Kooperationen → zu welchen Klassen bestehen Verbindungen?
- Ausgangsbasis: Anwendungsfälle für unsere SW
- die Entwicklung der Karten erfolgt in Gruppen zu 5 bis 6 Personen
- einfaches Beispiel für eine Veranstaltungsverwaltung siehe rechts



CRC-Karten

- eine Sitzung sollte sich immer auf einen Teilaspekt der SW konzentrieren
- typischer Ablauf:
 - Zusammenstellen der Gruppe → es müssen die richtigen Personen anwesend sein
 - Vorbereiten der Sitzung → Meetingraum mit notwendiger Ausstattung, CRC-Karten, Stifte, ...
 - Gemeinsames Finden der Klassen → zb. mittels Brainstorming
 - jede Klasse wird auf einer Karte vermerkt und einem Teilnehmer übergeben
 - jeder Teilnehmer ordnet seiner Klasse Verantwortlichkeiten zu
 - anschließend werden Szenarien durchgespielt und geprüft, welche Klassen dabei Verantwortung für welchen Teilaspekt übernehmen und welche Kooperationen zwischen den Klassen notwendig sind
 - falls ein Szenario nicht sofort abgebildet werden kann, werden die Klassen dementsprechend modifiziert
 - die letztgültige Version der Klassen wird dokumentiert
- Vorteile:
 - keine technischen Hilfsmittel notwendig
 - erleichtert die kreative Beschäftigung mit dem System
 - betont den objektorientierten Ansatz
- Nachteile:
 - Ergebnis eignet sich nur als Grobentwurf

5. Foliensatz

Pflichtenheft

- das Pflichtenheft ist das zentrale Dokument in der Vereinbarung zwischen Auftraggeber und Projektteam
- dokumentiert die **Wünsche des Auftraggebers** und die **Pflichten des Auftragnehmers**
- bildet die vertragliche Basis, die den Umfang und die Funktionalitäten der zu erstellenden Software beschreibt
- legt technische Rahmenbedingungen fest
- legt Verfahren und Termine des Projektablaufs fest
- bildet die Basis für
 - den Aufbau der Angebote
 - einen objektiven Angebotsvergleich
 - Vertragsverhandlungen und Vertrag

Pflichtenheft

- Bei Anforderungsdokumenten wird zwischen **Lastenheft** und **Pflichtenheft** unterschieden
- **Lastenheft:**
 - wird vor dem Pflichtenheft erstellt
 - wird vom Auftraggeber erstellt
 - beschreibt **WAS** zu erbringen ist und **WOFÜR**
- **Pflichtenheft:**
 - baut auf dem Lastenheft auf
 - wird im Projektverlauf erstellt
 - beschreibt detailliert
 - die technischen Merkmale
 - die gewünschte Lösung
 - die Rahmenbedingungen der Umsetzung
 - **WIE** und **WOMIT** die Anforderungen umzusetzen sind (z. B. Prozessmodelle, zeitlicher Rahmen, Budget, ...)
 - dient oft als Grundlage für die Ausschreibung der gewünschten Leistung

Pflichtenheft

- Bei Ausschreibungen wählt der Auftraggeber aus den eingelangten Offerten das beste Angebot aus
- die Auswahl erfolgt nach einem, vorher festgelegten, Kriterienkatalog und Verfahren
- das Pflichtenheft bildet dabei die Basis für den Vertrag zwischen Auftraggeber und Bestbieter
- der Kriterienkatalog zur Auswahl wird meist parallel zum Pflichtenheft erstellt
- Kriterien können dabei in drei Kategorien unterteilt werden:
 - Muss- (oder auch K.O. -) Kriterien:
 - müssen auf jeden Fall erfüllt sein
 - wird ein solches Kriterium nicht erfüllt, scheidet das Angebot aus
 - Soll-Kriterien:
 - es sollte eine möglichst gute Erfüllung angestrebt werden
 - die Erfüllung dieser Kriterien verbessert die Bewertung des Angebots
 - Abgrenzungskriterien:
 - sollen klarer beschreiben, welche Ziele bewusst nicht erreicht werden sollen
 - dienen zur Information des Anbieters zur richtigen Dimensionierung des Angebots

Pflichtenheft – Aufbau und Inhalt

- grundlegende Idee des Pflichtenhefts
 - Darstellung der gegenwärtigen Situation
 - klare Formulierung von Zielen und Anforderungen
 - Vorgabe des Offertaufbaus
- dient zur besseren Vergleichbarkeit mehrerer Angebote
- das Pflichtenheft soll alle relevanten Informationen bieten aber dabei **keine Lösungen vorschlagen**
- es wird vorgegeben in welcher Qualität zu realisieren ist aber nicht, wie die genaue Implementierung aussehen soll

Pflichtenheft – Aufbau und Inhalt

Kapitel des Pflichtenhefts

1. Beschreibung der Ausgangslage
 - Vorstellung des Unternehmens
 - Beschreibung der Produkte und Kundenstruktur
 - Überblick über die IT-Landschaft und Grund für die Beschaffung der neuen SW
2. IST-Zustand
 - alle für das Projekt relevanten Bereiche werden dargestellt
 - dargestellt sollten werden
 - Aufbauorganisation
 - Ablauforganisation, inklusive Beschreibung der relevanten Geschäftsprozesse
 - eingesetzte Applikationen
 - bestehende Systemplattform (Betriebssysteme, Technologien, Server, Datenbanken, Netzwerk, ...)

Pflichtenheft – Aufbau und Inhalt

Kapitel des Pflichtenhefts

3. Zielsetzung

- enthält die Ziele, die mit der Beschaffung erreicht werden sollen
- enthält ebenso die Prioritäten dieser Ziele
- Ziele sollten so formuliert werden, dass sie später auch überprüfbar sind
- Ziele können unterschieden werden in:
 - nutzenrelevante Ziele → Kosteneinsparungen, Verbesserung der Qualität und Effizienz, ...
 - Systemziele
 - Vorgehensziele → Prozesse, Termine, ...

Pflichtenheft – Aufbau und Inhalt

Kapitel des Pflichtenhefts

4. Anforderungen (Soll-Zustand)

- beschreibt detailliert die Eigenschaften, die der Kunde vom neuen System erwartet
- die Anforderungen werden unter folgenden Aspekten beschrieben:
 - Anforderungen an die Applikationssoftware → z. B.: Bedienung und Benutzerfreundlichkeit, Datenschutz und Datensicherheit, fachliche Use-Cases
 - Anforderungen an die Systemplattform → z. B.: Architekturvorgaben, Kommunikationsinfrastruktur, Rechnersysteme
 - Anbieterbezogene Anforderungen → notwendige Referenzen des Anbieters, gewünschte Dienstleistungen (z. B.: Schulung, Wartung, ...), Projektorganisation, Termine, Gewährleistung

Pflichtenheft – Aufbau und Inhalt

Kapitel des Pflichtenhefts

5. Mengengerüst

- beschreibt die zu erwartenden Datenmengen und Verarbeitungshäufigkeiten
- gliedert sich z. B. in:
 - Datenbewegungen
 - Datenbestände
 - Anzahl gleichzeitiger Benutzer
 - ...
- soll dem Anbieter die Möglichkeit bieten, das System richtig zu dimensionieren

Pflichtenheft – Aufbau und Inhalt

Kapitel des Pflichtenhefts

6. Aufbau und Inhalt der Offerte

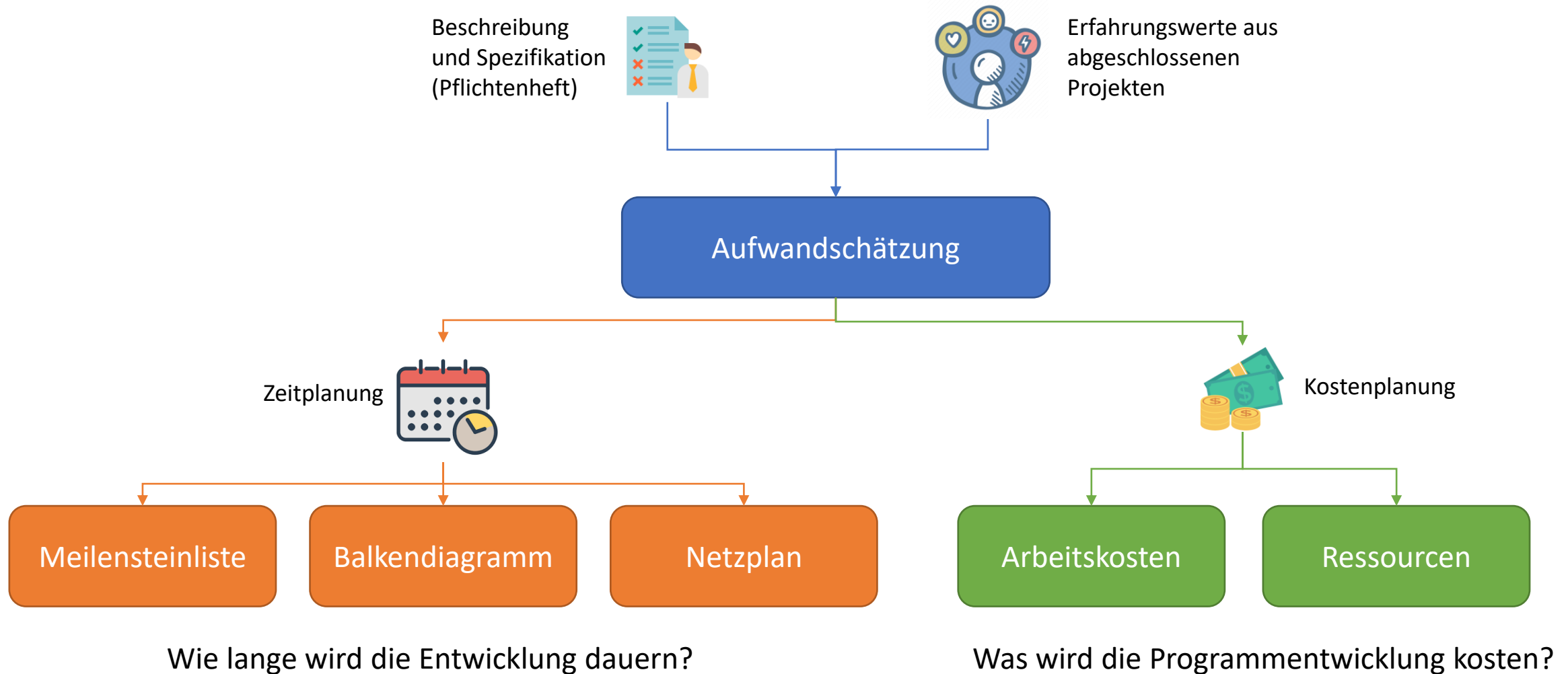
- bei einer Ausschreibung ist es wichtig, dass die einzelnen Offerte leicht vergleichbar sind
- hier wird vorgegeben, wie die Angebote strukturiert sein sollen

7. Administratives

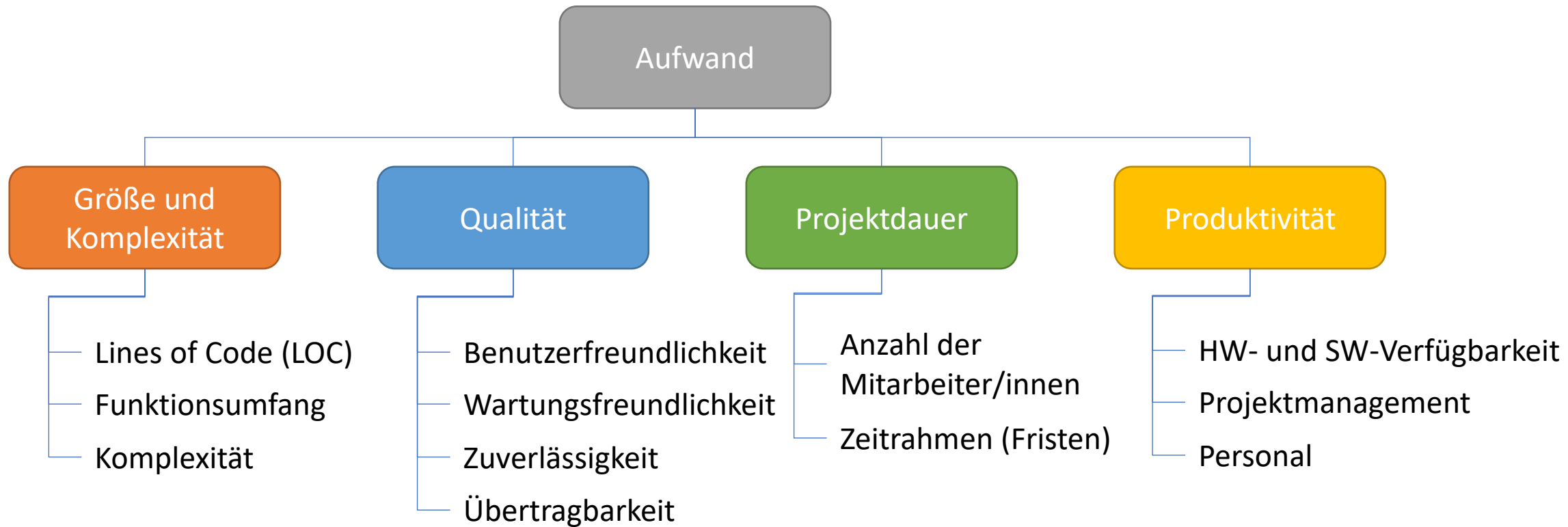
- Vertraulichkeit, Rückgabe, Copyright
- Evaluationsschwerpunkte
- Verteiler
- Budgetrahmen
- Rückfragen zum Pflichtenheft
- Termine
- Abgabe der Offerte

6. Foliensatz

Grundlagen der Aufwandschätzung

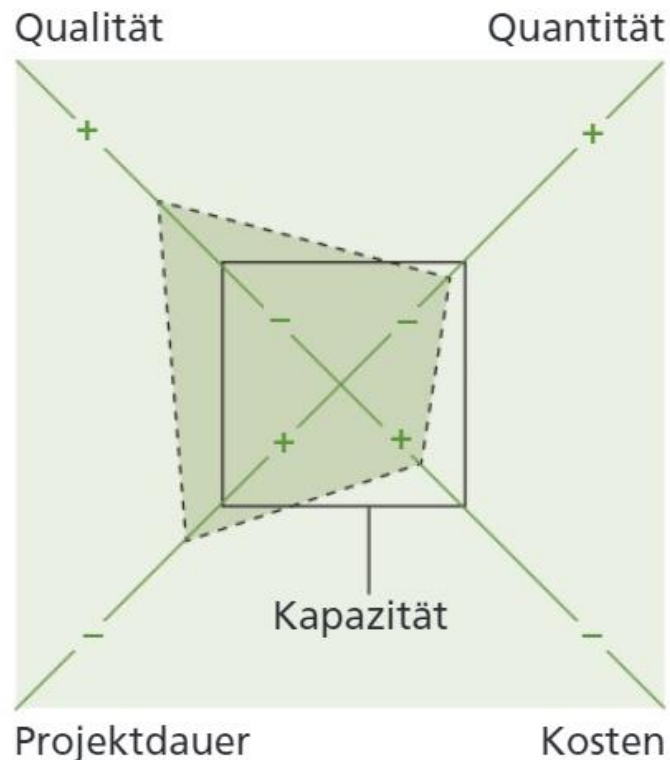


Einflussgrößen auf den Projektaufwand



Einflussgrößen auf den Projektaufwand

- die vier Einflussgrößen können nie unabhängig voneinander betrachtet werden
- eine Änderung bei einer Einflussgröße bedingt immer auch Änderungen bei den anderen
- kann z.B. mittels des „Teufelsquadrates“ dargestellt werden:



- Die Fläche des Quadrats muss immer gleich bleiben
- soll beispielsweise die Qualität gesteigert werden, muss die Quantität reduziert werden und/oder die Kosten gesteigert werden

Größe und Komplexität

- die **Größe** des SW-Systems ist eines der wichtigsten Merkmale für die Aufwandschätzung
- kann beschrieben werden mittels:
 - durch die Anzahl und Größe der notwendigen Klassen, sowie deren Zusammenhänge
 - durch die Anzahl der Programmzeilen (Lines of Code, LOC)
 - hat diverse Nachteile:
 - Überbetonung der Codierung → Analyse und Spezifikation werden im Aufwand nicht berücksichtigt
 - sehr abhängig von Programmiersprache und Programmierstil
 - durch den Funktionsumfang
 - von der Programmiersprache unabhängiger als LOC
 - reflektiert die Funktionalität des Endprodukts
 - allerdings muss unbedingt die Komplexität der einzelnen Funktionen in Betracht gezogen werden

Größe und Komplexität

- die **Komplexität** von Funktionen kann auf folgende Arten erfolgen_
 - durch subjektive Einschätzung → z. B. in „leicht“, „mittel“ und „schwer“
 - durch verschiedene Komplexitätsmaße → das sind Maßzahlen, die nach bestimmten Kriterien berechnet werden

Qualität

- die Qualität von Software kann unter anderem anhand folgender Kriterien gemessen werden:
 - Benutzerfreundlichkeit
 - Wartungsfreundlichkeit
 - Zuverlässigkeit
 - Funktionserfüllung
 - Zeit-/Verbrauchsverhalten
 - Übertragbarkeit

Qualität

- **Benutzerfreundlichkeit**
 - soll möglichst geringen Bedienungsaufwand garantieren
 - subjektiv positive Beurteilung der Bedienbarkeit durch den Benutzer
- **Wartungsfreundlichkeit**
 - soll geringen Zeit- und Kostenaufwand für Erkennung und Korrektur von Fehlern liefern
 - soll geringen Zeit- und Kostenaufwand für die Durchführung von Änderungen liefern
 - beinhaltet
 - Fehlerfreiheit
 - Modularität
 - Lesbarkeit
 - Einfachheit

Qualität

- **Zuverlässigkeit**
 - das fehlerfreie Funktionieren der Software
 - berücksichtigt auch, wie schwerwiegend die Folgen eines Fehlers sind
 - beinhaltet
 - Vollständigkeit
 - Konsistenz
 - Robustheit
 - Einfachheit
- **Funktionserfüllung**
 - gibt an, inwieweit die Software die geforderten Funktionalitäten erfüllt

Qualität

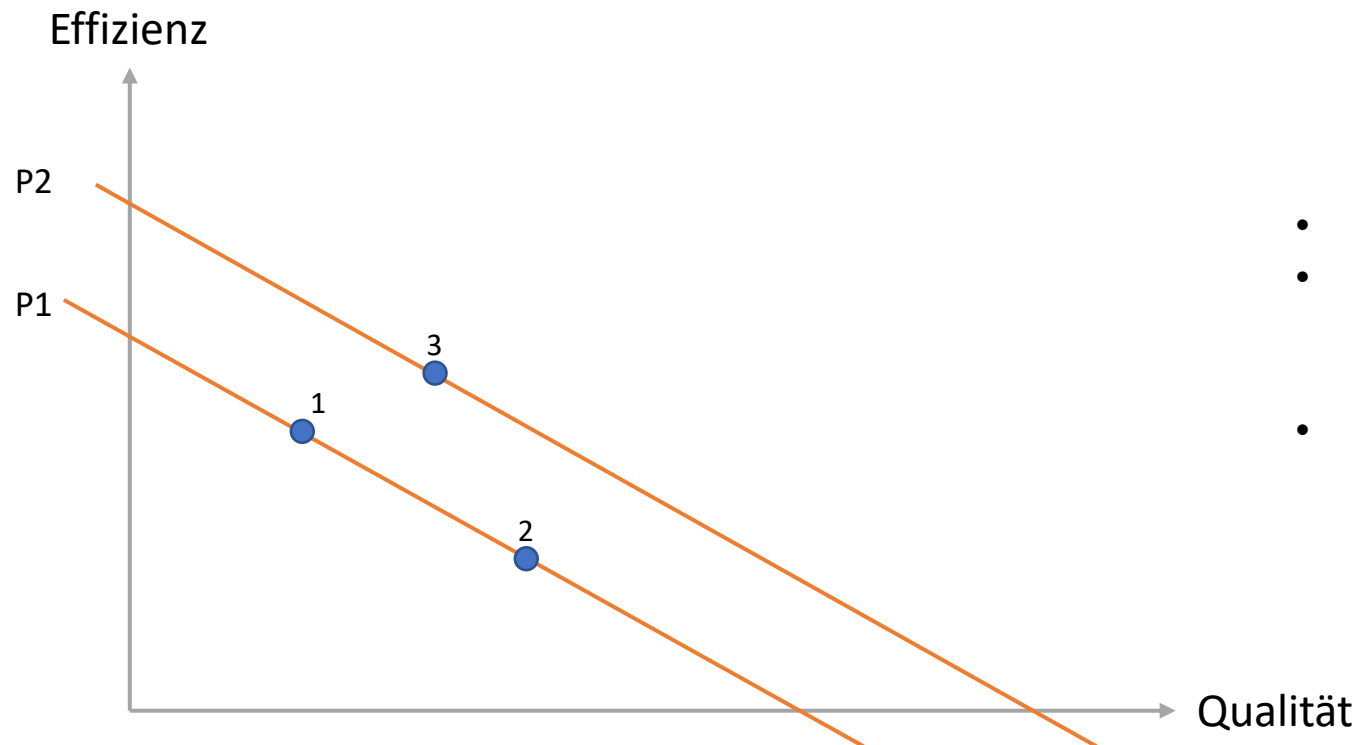
- **Zeit-/Verbrauchsverhalten**
 - wie schnell arbeitet die Software?
 - wie sehr werden die Ressourcen belastet?
 - beinhaltet
 - Performance
 - Durchsatz
 - Antwortzeitverhalten
- **Übertragbarkeit**
 - gibt an, inwieweit die Software für den Einsatz mit ähnlichen Aufgabenstellungen oder in geänderter technischer bzw. organisatorischer Umgebung geeignet ist
 - beinhaltet
 - Portabilität
 - Wiederverwendbarkeit
 - Verknüpfbarkeit

Projektdauer und Teamgröße

- die Projektdauer wird in Personenmonaten geschätzt → Dauer des Projekts, wenn eine Person daran arbeiten würde
- die optimale Anzahl an Mitarbeitern kann zumindest näherungsweise durch diverse Faustformeln errechnet werden
 - optimale Mitarbeiteranzahl = $\sqrt{\text{geschätzter Aufwand in PM}}$
 - z. B.: geschätzter Aufwand 16 Monate → $\sqrt{16} = 4$
 - optimale Mitarbeiteranzahl ist also 4 → Projektdauer wäre dann 4 Monate
 - optimale Projektdauer = $2,5 * (\text{Aufwand in PM})^s$
 - $s = 0,38$ für Batchsysteme
 - $s = 0,35$ für Dialogsysteme
 - $s = 0,32$ für Echtzeitsysteme
 - z. B.:
 - geschätzter Aufwand 16 Monate für einen Webshop (= Dialogsystem)
 - $2,5 * (16)^{0,35} = 6,6 \rightarrow 7 \text{ Monate} \rightarrow 2,3 \text{ Mitarbeiter}$

Produktivität

- bezeichnet sowohl die Qualität der geleisteten Arbeit als auch die Effizienz bei der Durchführung dieser Arbeiten
- Produktivität kann nur gesteigert werden, wenn beide Faktoren verbessert werden



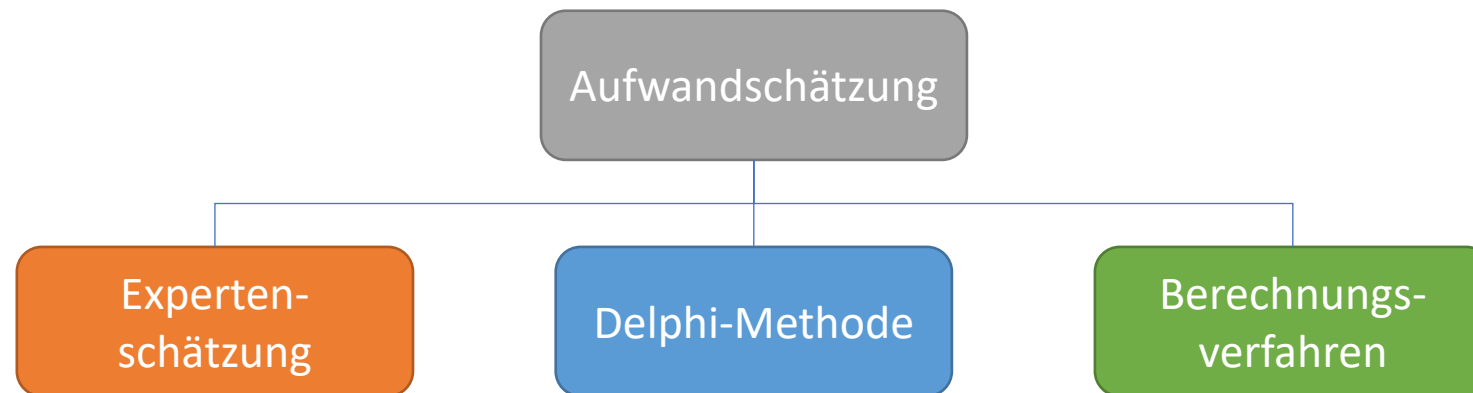
- P1 und P2 stellen Produktivitätsniveaus dar
- wird nur ein Einflussfaktor geändert, ändert sich die Produktivität nicht → Punkt 1 zu Punkt 2, wenn die Qualität gesteigert die Effizienz aber nicht erhöht wird
- um zu Punkt 3 zu gelangen müssen Qualität und Effizienz erhöht werden

Produktivität

- Haupteinflüsse auf die Produktivität sind
 - Personalqualität
 - Hardware- und Software-Verfügbarkeit
 - Organisations- und ablaufbedingte Einflüsse
- die Produktivität z. B. durch die folgenden Maßnahmen erhöht werden
 - Schulung der Mitarbeiter
 - bessere SW-Entwicklungstools
 - bessere Kommunikation
 - Verbesserung von organisatorischen Abläufen

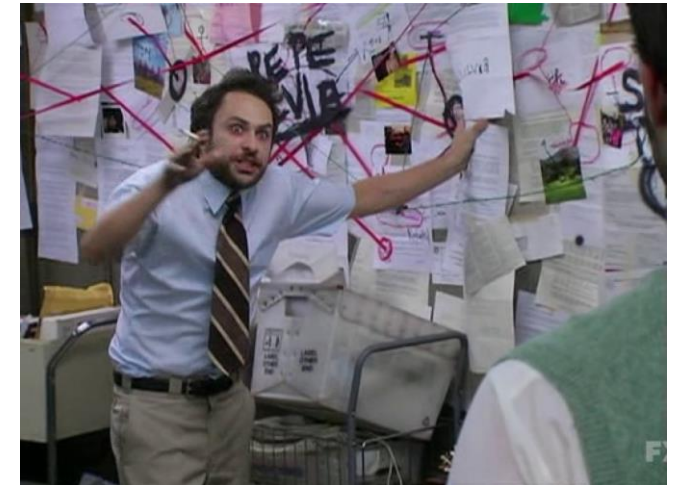
Berechnungsverfahren in der Aufwandschätzung

- Hauptproblem besteht darin, dass man zur Zeit der Schätzung meist noch nicht alle Anforderungen im Detail kennt
- die Schätzung sollte daher während der Projektlaufzeit mehrmals wiederholt werden



Expertenschätzung

- die an sich beste Methode zur Aufwandschätzung
- die Schätzung wird durch einen Experten/eine Expertin durchgeführt
- wichtig dabei ist folgendes:
 - der Experte/die Expertin hat Erfahrung mit ähnlichen Projekten
 - der Experte/die Expertin hat Erfahrung im Anwendungsgebiet des Kunden
 - der Experte/die Expertin kennt das Projektteam (bezüglich Produktivität)



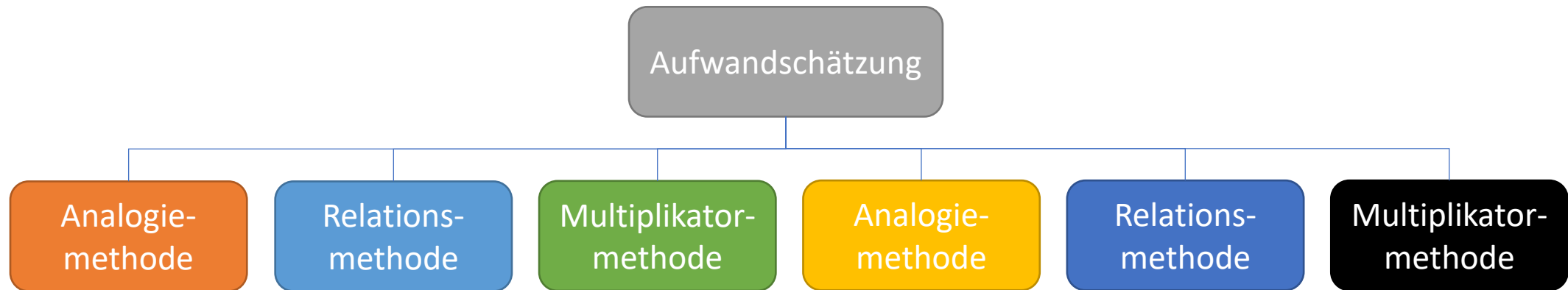
Delphi-Methode

- mehrere Fachleute geben eine Prognose in schriftlicher Form ab
- die gesammelten Schätzungen werden an alle Teilnehmer verteilt
- die Teilnehmer präzisieren oder modifizieren ihre Schätzungen
- nach mehreren Runden liegt eine gesicherte Endschätzung vor
- für SW-Projekte ist diese Methode normalerweise zu aufwändig



Berechnungsverfahren

- Ziel ist, Expertenwissen und –erfahrung in messbaren Größen auszudrücken
- der Aufwand soll durch mathematische Formeln berechnet werden
- meist werden die Berechnungsmethoden nicht einzeln sondern in Kombination angewendet



Analogiemethode

- **Beschreibung**
 - Grundlage bilden abgeschlossene Projekte
 - für bestimmte Faktoren werden die Unterschiede zum zu schätzenden Projekt ermittelt
 - die Kosten für die alten Projekte werden anhand dieser Unterschiede hochgerechnet
- **Bewertung**
 - wenig formalisiert
 - Erfahrungen des Schätzenden sind wesentlich
 - genaue Aufzeichnungen über die alten Projekte sind erforderlich

Relationsmethode

- **Beschreibung**
 - ähnlich der Analogiemethode
 - für die einzelnen kostenträchtigen Faktoren werden Indizes extrahiert
 - mithilfe dieser Indizes werden aus den Kosten der alten Projekte die Schätzungen für das aktuelle Projekte hochgerechnet
- **Bewertung**
 - Auswahl der Faktoren ist kritisch
 - genaue Aufzeichnungen über die alten Projekte sind erforderlich

Multiplikatormethode

- **Beschreibung**
 - Projekt wird in Teile oder Subsysteme zerlegt
 - den Teilen wird ein bestimmter Aufwand zugeordnet
 - durch Multiplikation wird der Gesamtaufwand errechnet
- **Bewertung**
 - aufwändige Zerlegung
 - Schätzung der Subsysteme wie bei der Analogie- bzw. Relationsmethode

Gewichtungsmethode

- **Beschreibung**
 - es müssen die Einflussfaktoren ermittelt werden, die für die Projektkosten wesentlich sind
 - Gewichtungsfaktoren drücken die Auswirkungen dieser Faktoren auf die Projektkosten aus
 - Bestandteil fast aller detaillierten Verfahren
- **Bewertung**
 - Gewichtung der Einflussfaktoren ist abhängig von Erfahrungen des/der Schätzenden

Methode der parametrischen Schätzgleichungen

- **Beschreibung**
 - basiert ebenfalls auf der Analyse bereits abgeschlossener Projekte
 - es werden mithilfe von Korrelationsanalysen Einflussfaktoren ermittelt, die Auswirkung auf die Projektkosten haben
 - zur Aufwandschätzung wird eine Gleichung mit Koeffizienten aus der Korrelationsanalyse aufgestellt
- **Bewertung**
 - gute Ergebnisse können nur bei sehr vielen ähnlichen Vergleichsprojekten erzielt werden

Prozentsatzmethode

- **Beschreibung**
 - durchschnittliche Aufwandsverteilung auf die einzelnen Projektphasen wird ermittelt
 - danach kann aus bereits abgeschlossenen Phasen auf die verbleibenden Phasen geschlossen werden
 - alternativ kann eine Phase detailliert geschätzt und dann auf die anderen Phasen hochgerechnet werden
- **Bewertung**
 - eine genaue Definition der Phasen ist notwendig
 - Vergleichsprojekte müssen eine möglichst ähnliche Aufwandsverteilung in den Phasen haben

7. Foliensatz

Verfahren der Aufwandschätzung

IBM-Handbuch-Verfahren

- ältestes Verfahren (1968)
- orientiert sich stark am „Eingabe-Verarbeitung-Ausgabe“ Schema
- auf Grundlage des Feinkonzepts wird der Aufwand für die Programmierung geschätzt
- Formel:
 - $T_{pp} = (T_{ea} + T_v) * (T_k + T_e)$
 - $T_{pp} \rightarrow$ Aufwand für die Programmierung
 - $T_{ea} \rightarrow$ Summe des Programmieraufwands für Ein-/Ausgabeprozesse
 - $T_v \rightarrow$ Summe des Programmieraufwands für die Verarbeitungen
 - $T_k \rightarrow$ Problemkenntnisfaktor der beteiligten Programmierer
 - $T_e \rightarrow$ Programmiererfahrungsfaktor

Verfahren der Aufwandschätzung

Boeing-Verfahren

- stammt aus dem Jahr 1977
- je nach Modultyp werden verschiedene Produktivitäten angenommen (z. B. 8 PM pro 1000 Befehle für Ausgaberoutinen)
- mithilfe eines Schlüssels wird der Gesamtaufwand auf die einzelnen Entwicklungsphasen verteilt

Maßzahlenverfahren

- als Grundlage für die Aufwandschätzung dienen systematisch erfasste Daten und Auswertungen aus früheren Projekten
- die Daten aus den früheren Projekten werden in Teilprodukte eingeteilt, deren Kosten laufend erfasst werden
- den Teilprodukten werden Maßzahlen zugeordnet (z. B. LOC für Software, DIN A4 Seiten für Anforderungsdokumente, ...)
- aus den erfassten Kosten der Vorprojekte können dann Einheitskosten für die einzelnen Maßzahlen errechnet und damit ähnliche Projekte abgeschätzt werden

Verfahren der Aufwandschätzung

COCOMO (Constructive Cost Model)

- verwendet die Gewichtungsmethode und parametrische Schätzgleichungen
- für große bzw. sehr große Projekte geeignet
- für kleinere Projekte liefert die Methode zu hohe Werte

Verfahren der Aufwandschätzung

Function Point Verfahren

- wurde 1979 entwickelt
- eher für die Aufwandschätzung kaufmännischer Anwendungen geeignet, weniger für technische oder betriebssystemnahe Anwendungen
- das Anwendungssystem wird dabei aus der Sicht des Benutzers betrachtet
- kann nur für gesamte Anwendungen angewendet werden, nicht für einzelne Module
- schätzt den Aufwand von der Bedarfsanalyse bis zur Übergabe
- ist in 5 Schritte gegliedert:
 1. Quantifizierung des Umfangs – Ermitteln von „Functions“
 2. Bewertung des Schwierigkeitsgrads der Functions durch „Points“
 3. Analyse des Einflusses von sieben vorgegebenen Faktoren
 4. Berechnung der bewerteten Function Points
 5. Ermittlung des Aufwands mittels historischer Daten

Verfahren der Aufwandschätzung

Function Point Verfahren

1. Quantifizieren des Funktionsumfangs
 - zuerst werden die Funktionstypen (Functions) eines Projekts analysiert
 - es werden fünf Funktionstypen unterschieden:
 - Eingabedaten
 - Ausgabedaten
 - Datenbestände
 - Referenzdaten
 - Abfragen

Verfahren der Aufwandschätzung

Function Point Verfahren

2. Bewertung des Schwierigkeitsgrads durch „Points“
 - alle Funktionstypen können im Projekt mehrfach vorkommen
 - für jedes Auftreten wird eine Einteilung getroffen in
 - leicht
 - mittel
 - schwer
 - die Einteilung erfolgt mittels vorher definierter Tabellen (Bsp. siehe Buch S. 263)

Verfahren der Aufwandschätzung

Function Point Verfahren

3. Analyse der Einflussfaktoren

- zusätzlich zu den Function Points müssen noch weitere Einflussfaktoren miteinbezogen werden:
 1. Verflechtung mit anderen Anwendungssystemen → bewertet auf Skala von 0 bis 5
 2. Dezentrale Verwaltung der Daten → bewertet auf Skala von 0 bis 5
 3. Transaktionsrate → bewertet auf Skala von 0 bis 5
 4. Verarbeitungslogik → bewertet auf Skala von 0 bis 30
 - summiert sich aus den folgenden Größen:
 - schwierige/komplexe Rechenoperationen: 0 – 10
 - umfangreiche Kontrollverfahren: 0 – 5
 - viele Ausnahmeregelungen: 0 – 10
 - schwierige, komplexe Logik: 0 - 5

Verfahren der Aufwandschätzung

Function Point Verfahren

3. Analyse der Einflussfaktoren

- zusätzlich zu den Function Points müssen noch weitere Einflussfaktoren miteinbezogen werden:
 5. Wiederverwendung in anderen Anwendungen
 - gibt an, wie stark eine Wiederverwendung der Programme in anderen Anwendungen zu berücksichtigen ist:
 - 0-10% → 0 Punkte
 - 10-20% → 1 Punkte
 - 20-30% → 2 Punkte
 - 30-40% → 3 Punkte
 - 40-50% → 4 Punkte
 - über 50% → 5 Punkte
 6. Datenbestandskonvertierungen → bewertet auf Skala von 0 bis 5
 7. Parametrisierbarkeit durch den Benutzer → bewertet auf Skala von 0 bis 5

Verfahren der Aufwandschätzung

Function Point Verfahren

4. Berechnung der bewerteten Function Points

$$\text{BFP} = (\text{Summe FP}) * (0,7 + \text{EF} * 0,01)$$

5. Ermittlung des Aufwands

- Aufgrund von Erfahrungen aus früheren Projekten wird eine Funktionstabelle (BFP, Aufwand in PM) erstellt
- aus dieser Tabelle kann der Aufwand abgelesen werden

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

- neueres Verfahren zur Aufwandschätzung
- geht von Anwendungsfällen (Use Cases) aus
- wird im RUP verwendet
- wie beim Function Point Verfahren werden auch hier bestimmte Merkmale gezählt und bewertet
- Unterschied zum FP-Verfahren:
 - UCP-Verfahren geht nicht von Funktionen der Software, sondern von Anwendungsfällen aus User-Sicht aus
 - liegt damit näher an der aktuellen Analyse- und Entwurfstechnik
- Vorteil der Methode:
 - Aufwandschätzung kann schon in einem frühen Stadium des Projekts durchgeführt werden, da die SW-Spezifikation noch nicht so weit fortgeschritten sein muss, wie beim FP-Verfahren
 - vor allem bei kürzeren SW-Projekten kommt FP-Verfahren oft zu spät

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

- folgende Faktoren werden erhoben und bewertet:
 - Anzahl und Komplexität der Anwendungsfälle (Use Cases)
 - Anzahl und Komplexität der Akteure (entweder Personen oder andere beteiligte Systeme)
 - technische Einflussfaktoren in der Systementwicklung
 - Einflussfaktoren zu Kompetenz und Performance des Entwicklungsteams
- Herausforderung bei der Methode ist die Modellierung der Use Cases und finden der essenziellen Use Cases für die Schätzung

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

1. Actor Einfluss (UAW – Unadjusted Actor Weight)
 - es werden die Akteure des Systems gezählt
 - Akteure können User oder auch andere Systeme sein

Kategorie	Beschreibung	Faktor	Beispiel	Anzahl (Annahme)	gesamt
einfach	anderes System mittels definierter API	1	Die Anzahl der Akteure werden durch Abzählen in den Use-Case-Diagrammen ermittelt	5	5
mittel	anderes System mittels Protokoll (z.b. XML, HTTP)	2		2	4
komplex	User über graphische Oberfläche	3		3	9
UAW gesamt = $A_e \cdot 1 + A_m \cdot 2 + A_k \cdot 3$				UAW gesamt	18

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

2. Use Case Einfluss (UUCW – Unadjusted Use Case Weight)

- es werden die Anzahl der Use Cases und deren Komplexität ermittelt
- die Komplexität eines Use Cases ergibt sich aus der Anzahl der Transaktionen, die im Use Case abgedeckt werden

Kategorie	Beschreibung	Faktor	Beispiel	Anzahl (Annahme)	gesamt
einfach	1-3 Transaktionen	5	Die Transaktionen in den einzelnen Use Cases werden gezählt.	10	50
mittel	4-7 Transaktionen	10		14	140
komplex	über 8 Transaktionen	15		7	105
UUCW gesamt = $U_e \cdot 5 + U_m \cdot 10 + U_k \cdot 15$				UUCW gesamt	295

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

3. Technikbedingte Einflussfaktoren (TCF – Technical Complexity Factor)

- jeder einzelne Faktor wird mit einem Wert zwischen 0 (kein Einfluss) und 5 (sehr starker Einfluss) bewertet

Nr.	Beschreibung	Gewicht	Beispiel	zugeordneter Wert	gesamt
T1	verteiltes System	2	System läuft auf zentralem Rechner	0	0
T2	spezielle Anforderungen an Durchsatz und/oder Antwortzeiten	2	ausreichend für geübten Benutzer	3	6
T3	hohe Online-Performance für den Benutzer	1	hohe Online-Performance erforderlich	5	5
T4	komplexe interne Berechnungen	1	kaum	1	1
T5	Code muss wiederverwendbar sein	1	nein	0	0
T6	einfache Installation erforderlich	0,5	Installation erfolgt durch Spezialisten	1	0,5
T7	hohe Benutzerfreundlichkeit erforderlich	0,5	sollte sehr benutzerfreundlich sein	5	2,5
T8	Portabilität des Codes	2	nicht erforderlich	0	0
T9	leichte Änderbarkeit	1	sollte leicht adaptierbar sein	4	4
T10	Parallelverarbeitung	1	keine Parallelverarbeitung	0	0
T11	spezielle Anforderungen an die Sicherheit	1	hohe Sicherheit gegen Missbrauch	5	5
T12	ermöglicht Direktzugriff durch Dritte	1	Kunden können auch direkt online buchen	5	5
T13	macht spezielle Schulungseinrichtungen nötig	1	nein, wenige Benutzer, benutzerfreundlich	1	1
				Total technical factor(Tfactor)	30

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

3. Technikbedingte Einflussfaktoren (TCF – Technical Complexity Factor)
 - $TCF = 0,6 + (0,01 * T_{\text{faktor}})$
 - im Beispiel: $TCF = 0,6 + (0,01 * 30) = 0,9$

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

4. Einflussfaktoren aus der Entwicklungsumgebung (EF – environment factor)

- hier werden Einflüsse aus der Projektumgebung berücksichtigt
- z. B.: Motivation und Kompetenz der Teammitglieder
- jeder einzelne Faktor wird mit einem Wert zwischen 0 (kein Einfluss) und 5 (sehr starker Einfluss) bewertet

Nr.	Beschreibung	Gewicht	Beispiel	zugeordneter Wert	gesamt
E1	routinierter Umgang mit dem RUP	1,5	Die meisten Teammitglieder haben schon im RUP gearbeitet	4	6
E2	Erfahrung im Anwendungsbereich	0,5	erstes Buchungssystem dieser Art	1	0,5
E3	Erfahrung in der Objektorientierung	1	gute Kompetenz in Objektorientierung	4	4
E4	Spezialist für Analyse ist verfügbar	0,5	kein spezieller Analytiker im Team	1	0,5
E5	Motivation	1	hochmotiviertes Team	5	5
E6	Anforderungen sind stabil	2	es werden keine Änderungen der Anforderungen erwartet	2	4
E7	Teilzeitbeschäftigte im Team	-1	nur zwei Junior-Programmierer sind teilzeitbeschäftigt	1	-1
E8	schwierige Programmiersprache	-1	Programmierung in Java	1	-1
				Total environment factor (Efactor)	18

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

4. Einflussfaktoren aus der Entwicklungsumgebung (EF – environment factor)
 - $EF = 1,4 + (-0,03 * Efactor)$
 - im Beispiel: $EF = 1,4 + (-0,03 * 18) = 0,86$
5. Berechnung der gewichteten Use Case Points (AUCP – Adjusted Use Case Point):
 - $AUCP = (UAW + UUCW) * TCF * EF$
 - im Beispiel: $AUCP = (18 + 295) * 0,9 * 0,86 = 242,262$

Verfahren der Aufwandschätzung

Use-Case-Points-Verfahren

6. Berechnung des Aufwands in Personenstunden

- einem AUCP wird eine bestimmte Zahl an Personenstunden (normalerweise 20) zugeordnet

Berechnet Größe	Berechnung	Anmerkung
Gesamtaufwand in Personenstunden	$GA = AUCP * 20 = 242,262 * 20 = 4.845,25 \text{ Ph}$	
Aufwand in Personenmonaten	$PM = 4845 : 167 \approx 29$	167 Stunden / Monat 38,5 h / Woche
Teamgröße	$TG = \sqrt{PM} = \sqrt{29} \approx 5,4$	Teamgröße mit 5 Personen reicht aus
Projektdauer	$PD = PM : TG = 29 : 5 \approx 6 \text{ Monate}$	
Aufwand je Iteration (in RUP 2 wöchig)	$ITaufw = 5 \text{ Personen} * 38,5 \text{ h/Woche} * 2 = 385 \text{ Ph}$	
Anzahl notwendiger Iterationen	$ITanz = GA : ITaufw = 4845 : 385 \approx 13$	
abzuarbeitende UUCW je Iteration	$UUCW : ITanz = 295 : 13 \approx 23$	