



Bernd Oestereich
- Geschäftsführer -



Christian Weiss
- Senior Consultant -

Featurebasiertes Multi-Releasemanagement in iterativen Großprojekten

© 2006 by oose GmbH

Abstract:

Große und lang laufende Projekte mit 50 - 200 Mitarbeitern die iterativ und (soweit für die Größenordnung überhaupt möglich) halbwegs agil vorgehen wollen, stehen vor großen Herausforderungen bei der Frage wie Anforderungen, Iterations-Builds, Multi-Releaseplanung, Meilensteinplanung, Teilprojekt- und Teampfanplanung zusammenspielen können. Wie kann in diesem komplexen Kontext Planung, Steuerung und Controlling eines Projektes realistisch und offensiv betrieben werden? Die Vortragenden zeigen typische Stolpersteine und Fehler auf und berichten über ihre Lösungsansätze hierzu.

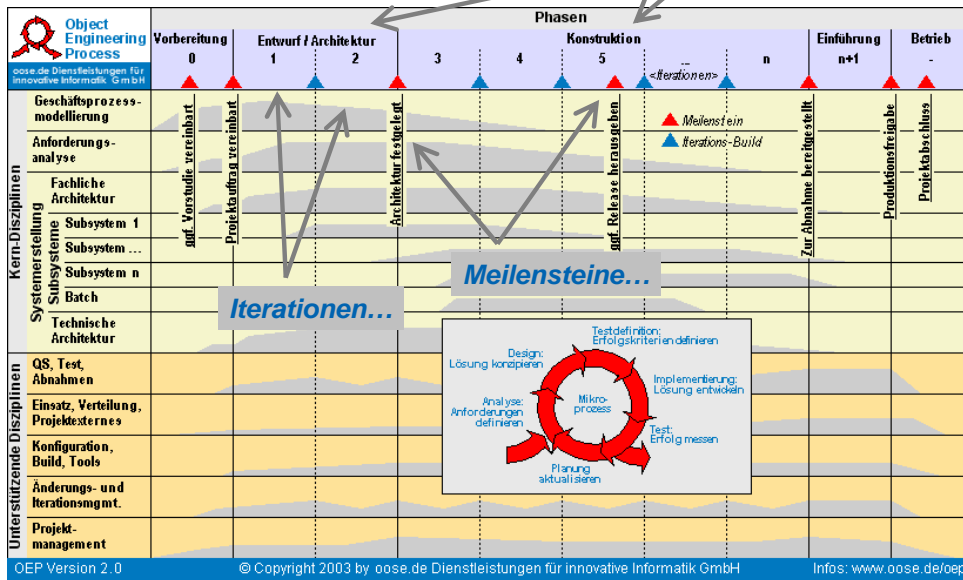
- *Zielpublikum: Manager, Projektleiter*
- *Voraussetzungen: Projektmanagement-Erfahrungen*
- *Schwierigkeitsgrad: mittel*

Bernd Oestereich ist Geschäftsführer der oose GmbH und Autor zahlreicher international verlegter Buch- und Zeitschriftenpublikationen. Mit seinen Publikationen sowie seiner Beratungs- und Schulungstätigkeit gibt er immer wieder wichtige Impulse für die Softwareentwicklungsbranche im deutschsprachigen Raum. Zur OOP erscheint sein neues Buch zum Thema dieses Vortrages.

Christian Weiss beschäftigt sich seit 1991 mit objektorientierter Softwareentwicklung u.a. als leitender Entwickler und Berater. Auch an Hochschulen referiert er regelmäßig. Seine Schwerpunkte liegen im Bereich objektorientierter Analyse- und Designtechniken sowie der Unterstützung des Projektmanagements von Großprojekten.

OEP: ein typisches Iteratives Vorgehen...

Phasen...



Weitere Informationen unter: <http://www.oose.de/oep>

© 2006 by oose GmbH

Vorbereitung: Problem verstehen, Planungs- und Entscheidungsbasis schaffen

System-Kontext erstellen, primäre Anwendungsfälle finden, Risiken und Erfolgskriterien definieren, Realisierungsalternativen untersuchen etc.

Architektur: Lösungsansatz verstehen, Architektur festlegen

Architekturrelevante Anwendungsfälle detaillieren, Fachliche Architektur entwickeln, Prototypen für hochpriorisierte Anwendungsfälle erstellen usw.

Konstruktion: Lösung erstellen

Anwendungsfälle detaillieren, Design-Modelle erstellen, Programmieren, Testen, Benutzerhandbücher erstellen u.v.m.

Einführung: Lösung einführen

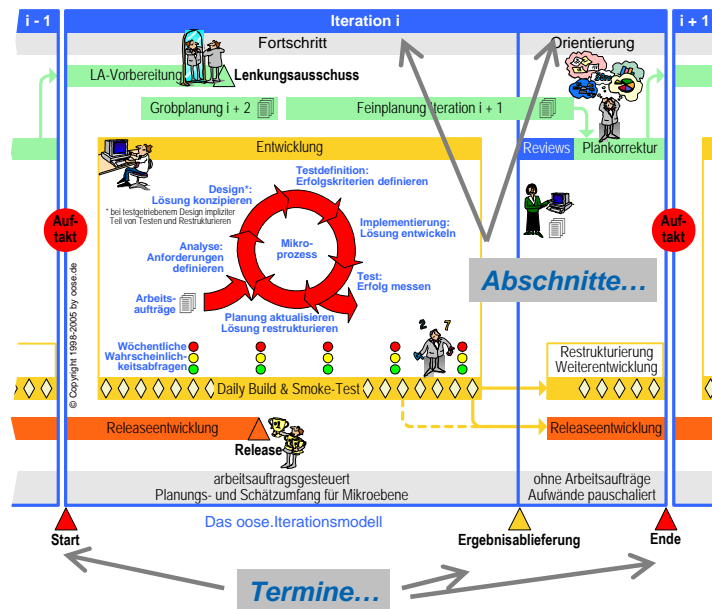
Test und Abnahme in Zielumgebung sowie Schulung und Rollout durchführen, Inbetriebnahme

Eine **Phase** ist ein zeitlich bzw. sachlogischer Gliederungsabschnitt eines Projekts. Eine Phase fasst eine Menge von Aktivitäten und Ergebnissen zu einer Planungs- und Kontrolleinheit zusammen. Am Ende jeder Phase steht gewöhnlich ein Meilenstein, der die in der Phase zu erzielende Inhalte definiert.

Eine **Iteration** ist ein in ähnlicher Weise mehrfach vorkommender Zeitabschnitt in einem Prozess. Iterationen sind in der Regel Timeboxen.

Ein **Meilenstein** (*Milestone*) definiert einen für das Gesamtprojekt äußerst wichtigen Termin, zu dem eine Menge von Ergebnissen in vorgegebener Detaillierung und Vollständigkeit nachprüfbar und formal dokumentiert vorliegen muss. Liegen die Ergebnisse zum geplanten Termin nicht vor, wird der Meilenstein verschoben. Meilensteine sind Hilfsmittel zur Planung und Überwachung eines Entwicklungsprozesses, d.h. je weniger existieren desto größer die Zielfokussierung.

Mikro-Prozess: Iteration im Detail



Je Iteration: Fortschrittsabschnitt + **Orientierungsabschnitt**

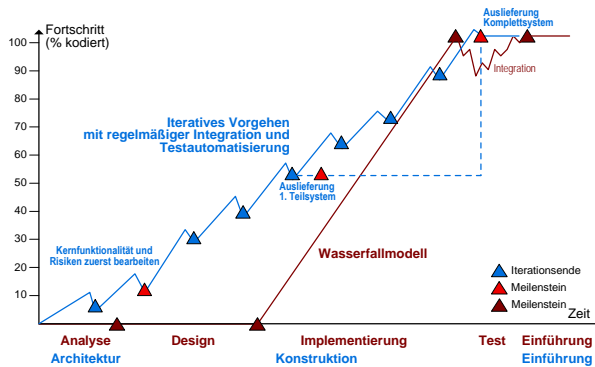
Je Iteration: Start, Auftakt, **Abgabetermin** und (formales) Ende

Iteration = Timebox

- Abgabetermin steht fest und ist unverrückbar
- Inhalt einer Iteration genau definiert und für den Zeitraum realistisch
- Abgabetermin ist wichtiger als der Lieferinhalt
- Während laufender Iterationen: möglichst keine Störungen von Vorgesetzten
- Keine von außen kommenden Änderungen der Anforderungen, Liefervereinbarungen oder Termine während einer Iteration
- Im wöchentlichen **Jour Fix** berichten Aufgaben-Verantwortliche über Fortschritte

Vorteile des Iterativen Vorgehens

- Besseres Risikomanagement
- Flexiblere Auslieferung
- Sichtbarer Projektfortschritt
- Optimierter Entwicklungsprozess
- weniger dokumentenlastig
- Höhere Kundenakzeptanz
- ...



Nachteile:

- anspruchsvoll
- es muss geplant werden
- Kunde sieht, was man tut
- Projektstillstand nur kurz verheimlichbar... ;-)

© 2006 by oose GmbH

Weitere Vorteile:

- Integrierte Teststrategien
- Fortlaufende Planung
- Gleichmäßigere Mitarbeiterauslastung
- Motiviertere Mitarbeiter

Weitere Nachteile:

- Umdenken erforderlich (ggf. Kulturschock)
- Kunde könnte früher aussteigen

Typische Symptome beim Iterativen Vorgehen



Symptom

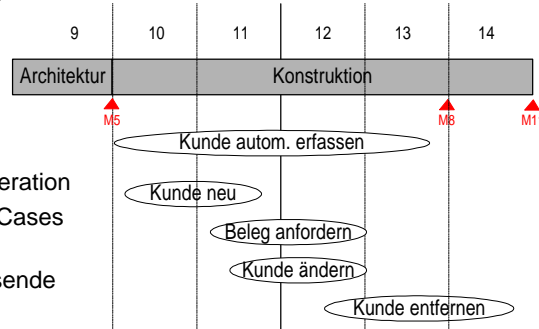
1. Mangelnde Produktorientierung:



- Vorherrschende Denkweise: „Wir sind für das Thema XY zuständig“
- eher zufälliges Gesamtergebnis nach Teilergebnis-Integration
- Herrchen-Hund-Syndrom:
Softwarearchitektur = Team-Struktur

2. Nie fertige Use Cases:

- Use Cases oft zu komplex für eine Iteration
- ständig Korrekturen realisierter Use Cases
→ Gefühl: „nie wirklich fertig“
- kein sinnvolles Ganzes am Iterationsende



Use Cases als Planungseinheiten **ungeeignet!**

© 2006 by oose GmbH

1. Mangelnde Produktorientierung:

- Teams bearbeiten fachliche Themen (nicht Produkte). Mitarbeiter identifizieren sich vorrangig mit dem Team, in dem sie arbeiten und verfolgen daher vorrangig den eigenen Erfolg (und nicht die Produkteinführung). Verantwortung für Einführungserfolg wird abgegeben („...dafür sind andere zuständig...“).
- Unkoordinierte Schnittstellen-Absprachen und laufende Änderungen an vereinbarten Schnittstellen produzieren eher ein zufälliges Gesamt-Produkt nach der Integration von Teamergebnissen. Dies wird bei parallelen Releases zum echten Problem.
- Herrchen-Hund-Syndrom: Softwarestruktur sieht am Ende aus wie die Teamstruktur. Projektorganisation wird starr und ist kaum noch an Erfordernisse anpassbar.

2. Nie fertige Use Cases:

- Use Cases überspannen oft Subsysteme und Teams und sind beim Großprojekten nur extrem selten wirklich innerhalb einer Iteration vollständig fertigstellbar (Analyse + Design + Programmierung + Test). Trotzdem werden sie als Planungseinheit verwendet, so dass ständig „Nach-Aufwände“ durch Korrekturen gebucht werden. Das Gefühl „nie fertig zu werden“ erzeugt sukzessive Unglaubwürdigkeit.
- Use Cases sind oft unglücklich über die Iterationen verteilt, z.B. Iteration 1+2: *Kunde neu*; danach in Iteration 3+4 dann *Kunde ändern* usw. Es ist unklar, was genau nach Iteration 1 „fertig“ ist. Ein auslieferbares Produkt entsteht auf diese Weise erst am Ende der Konstruktionsphase. Wozu dann Iterationen?

Typische Symptome beim Iterativen Vorgehen



Symptom

3. Rhythmusstörungen bei Zulieferleistungen:



- Vorherrschende Denkweise: „*Was müssen wir tun? Wie lange brauchen wir?*“
- Lange Terminlisten, Erhebliche Synchronisationsprobleme zwischen Teams
- Iterationsenden verstreichen wirkungslos (bzw. werden verschoben)
- Macht des Gleichtakts verspielt (z.B. Verbesserung der Schätzgenauigkeit)

4. Fehlende Transparenz:

- Projekterfolg ist, wenn Dokumente gemäß Vorgehensmodell vorliegen
- Dokumente wiederholen sich je Iteration: *Datenmodell A, B, C...*
- Ziel-Erreichungsgrad beliebig darstellbar
→ Objektiver Projektfortschritt nicht messbar



Fertige Dokumente suggerieren **trügerische Sicherheit!**

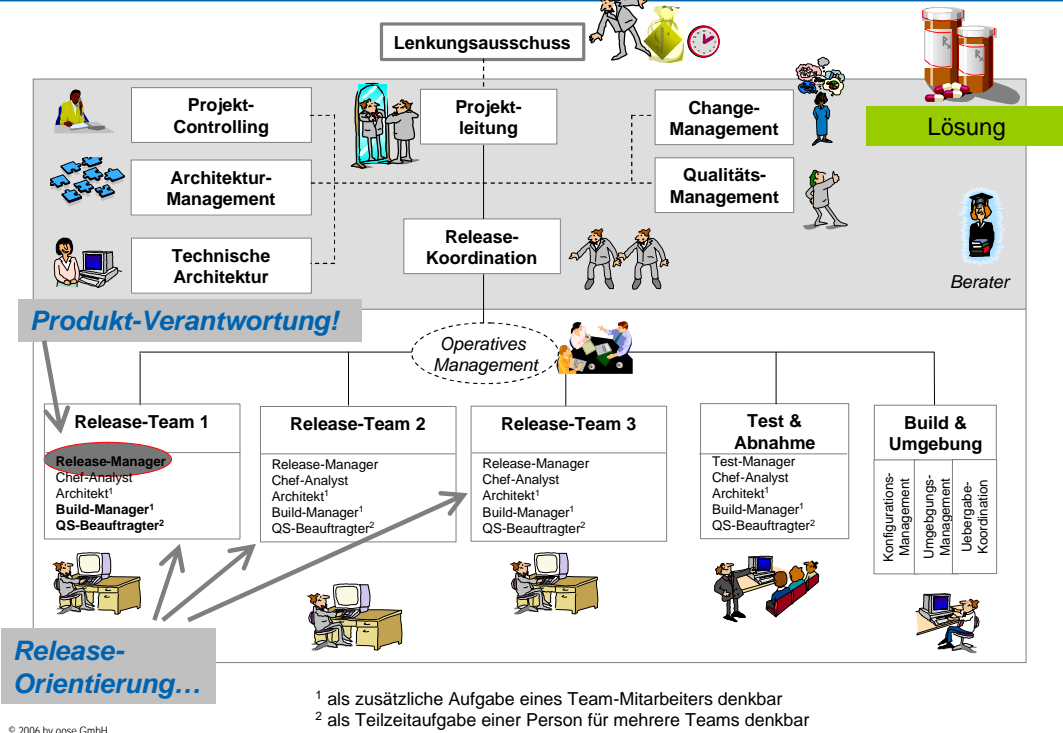
© 2006 by oose GmbH

3. Rhythmusstörungen bei Zulieferleistungen:

- Teams richten Iterationslänge an geschätzten Aufwänden für bevorstehende Aufgaben aus – und zwar jedes Team für sich!
- Dadurch gibt es haufenweise Iterationen mit endlos langen Terminlisten (wann hat welches was fertig?). Die Arbeit wird nicht rechtzeitig fertig, der Termin wird verschoben (die Iteration verlängert), und das abhängige Team kommt nicht weiter.
- Iteratives Vorgehen wird irgendwann verworfen: „siehste - funktioniert ja doch nicht“

4. Fehlende Transparenz:

- Messung des Projekterfolgs orientiert sich an Dokumenten (und nicht an funktionierender Software), die zum Selbstzweck erzeugt werden, weil es das Vorgehensmodell so will.
- Dokumente wiederholen sich von Iteration zu Iteration (Konzept A, B, C) und werden kopiert, um der Planung zu genügen. Das Controlling verkümmert zum unliebsamen Formalismus.
- Die Zielqualität (bzw. Umfang) der Dokumente ist unverbindlich oder gar nicht festgelegt; der Erreichungsgrad daher beliebig darstellbar. Ein objektiver Fortschritt (im Sinne lauffähiger Software) ist nicht wirklich messbar.
- Fehlende Verbindlichkeit fördert Projektdurchhänger und baut keinen Druck auf.



Neue Denkweise: „Ich mache Release XY (und bearbeite Thema A).“

HINWEIS: „**Release**“ im Sinne von produktreif (siehe Grundbegriffe im Anhang)

Projektorganisation an geplante Releases orientieren, d.h.

- je Release 1 Team (Mitarbeiter wechseln später in andere Release-Teams)
- Verbindliche Vorgabe von Pflicht-Rollen in jedem Release-Team (für Abstimmung mit Querschnittsfunktionen wie QM, Architektur usw.)
- Je Release 1 Person benennen, die Termin- und Budgettreue verantwortet (= Release-Manager)
- Release-Koordination koordiniert parallele Release- und Querschnittsteams

Features als Planungseinheiten



Neue Denkweise: „*Welche (Teil-)Funktionalität könnten wir bereits fertig stellen?*“



Lösung

Kriterien/-Eigenschaften von Features:

- zusammengehörige Anforderungen
- in funktionierende Software umsetzbar (Ausführbarkeit)
- eigenständig testbar
- Produktcharakter für den Anwender (*Business Value*)
- als Planungseinheiten besser geeignet als Use Cases (da beliebig aufteilbar)
- im Nominalstil formuliert („-ung“)



Ableitung aus Betrachtung von

- zusammengehörigen Use Cases
- Schnittstellen zu Fremdsystemen
- GUI-Masken
- Fachkonzepten
- Produktkartons im Kaufhaus ☺

© 2006 by oose GmbH

Ein **Feature** ist eine in funktionierende Software umgesetzte und eigenständig testbare Sammlung zusammengehöriger Anforderungen.

- Zusammengehörige Anforderungen bilden oft einen gesamten Lebenszyklus eines Objektes oder wesentliche Teile eines Geschäftsprozesses ab; sie dürfen aber unvollständig sein.
- Ein Feature muss ausführbar sein – ein Konzept oder ein Datenmodell sind nicht ausführbar
- Es muss ein Testprotokoll geben, welches die Ausführung nachweisen kann (idealerweise wiederholbar).
- Ein Feature kann technischer Natur sein (die Datenbank-Anbindung von XY...), aber es muss den Business Value unterstützen. Der Fachbereich darf im Zweifel darüber entscheiden. Notwendige Tätigkeiten, die kein Feature sind, bekommen keine Öffentlichkeitswirkung und bleiben was sie sind: notwendige Tätigkeiten.
- Mit ein bisschen Phantasie lassen sich Features beliebig granular aufteilen (im Gegensatz zu Use Cases, die immer zeitlich zusammenhängend bleiben müssen).
- Use Cases werden mit Substantiv + Verb beschrieben (bspw. *Kfz reservieren*), Features werden zwecks besserer Abgrenzung nur im Nominalstil beschrieben (bspw. *rudimentäre Kundenverwaltung*)

Feature-Arten und Beispiele

Release-Features

- iterations-unabhängig
- bilden höchste Ebene der Produktstruktur
- finden sich oft auf Produktkartons



Iterations-Features

- sind eine Teilfunktionalität genau eines Release-Features
- in einer Iteration fertigstellbar

FEAT-1	Interessenten-Ersterfassung
FEAT-2	Dialoggesteuerte Kunden-Verwaltung
FEAT-3	Automatische Partnerübernahme
FEAT-4	Bonitäts-Ermittlung
FEAT-5	Adress- & Dublettenbereinigung
FEAT-6	Beleg-Anforderung
FEAT-7	Nachweis-Prüfung
FEAT-8	Formular-Erstellung

FEAT-1.1	<i>nur primäre Personendaten sowie eine Anschrift ohne Details</i>
FEAT-1.2	<i>mit Kontakt-Beschreibung, Geburtsdaten, mehrere Anschriften jedoch ohne Namenszusätze</i>
FEAT-1.3	<i>mit detailliertem Kontaktjournal und Namenszusätzen</i>

© 2006 by oose GmbH

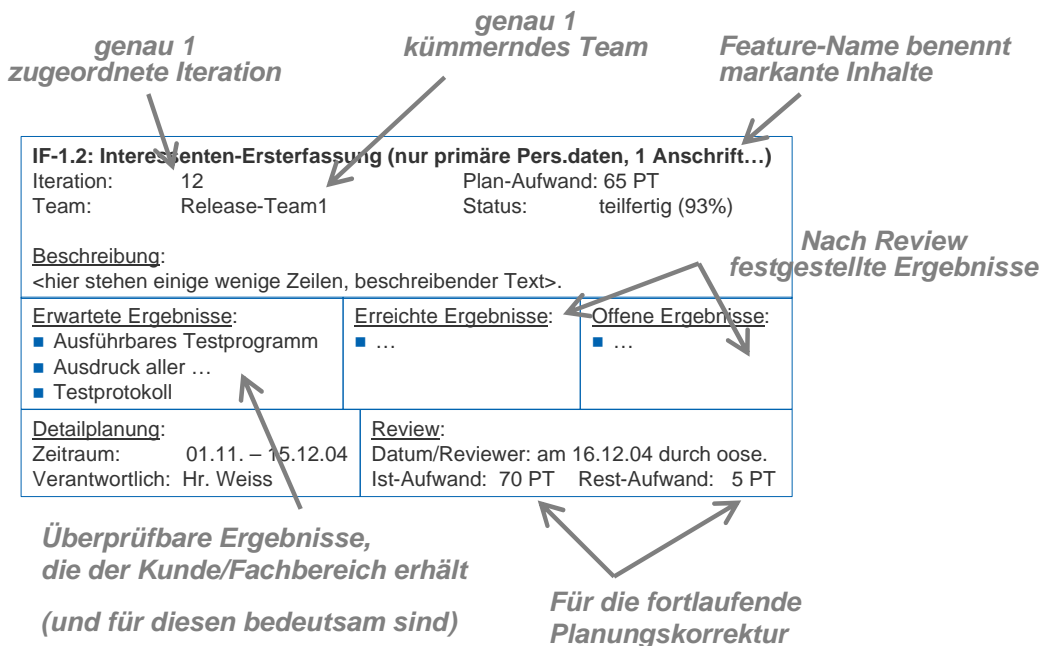
Release-Features

- Höchste Ebene der Produktstruktur sein, sie beschreiben von außen wahrnehmbare Eigenschaften einer Software (im Sinne eines Produkts)
- Für das Finden spielt die **Wertigkeit** eine große Rolle, d.h. ein Release-Feature repräsentiert eine für den Anwender/Kunden bedeutsame Funktionalität oder Eigenschaft. Es sind also komplexe Abläufe, wie auch herausragende Teilschritte möglich (z.B. wichtige Prüfungen o.ä.), genauso wie nicht-funktionale Merkmale, z.B. Sicherheit u.v.m.

Iterations-Features

- Sind jeweils genau einem Release-Feature zuordenbar
- *fertigstellbar* meint hier: mindestens Entwurf, Realisierung und Test. Die Anforderungsanalyse ist oft eine oder mehrere Iterationen vorgelagert (muss aber nicht)
- Beim Schneiden spielt die **Komplexität** die wesentliche Rolle, da ein Iterations-Feature in einer Iteration fertigstellbar sein muss. M.a.W.: je kürzer die vorgegebene Iterationsdauer, desto kleiner die Iterations-Features (und desto mehr)

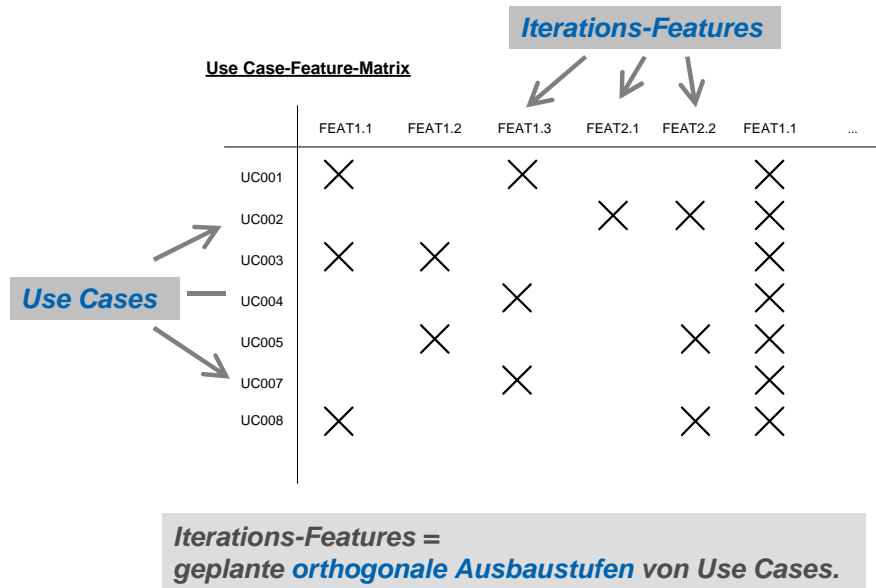
Feature-Beschreibung



© 2006 by oose GmbH

- **Team:** genau 1 Team kümmert sich, d.h. es trägt die Erfolgsverantwortung (was nicht heißen muss, dass dieses Team den Hauptteil der Arbeit daran haben muss)
- **Beschreibung:** ersetzt keine Anforderungs-Spezifikation. Beschreibung nur soweit wie nötig, um die betroffenen Anforderungsteile wie Use Case-Schritte identifizieren zu können.
- **Erwartete Ergebnisse:** UML-Modelle, Konzepte und dergl. sollen hier keine Rolle spielen (oder zumindest nur eine unbedeutende). Wichtig sind Ergebnistypen, die bedeutsam für den Kunden und für den Nachweis des Funktionierens sind.
- **Review:** Am Iterationsende werden Ist-Aufwand und Rest-Aufwand ermittelt, um ein Controlling zu ermöglichen (später mehr dazu) und die fortlaufende Planung anzupassen..
- **Offene Ergebnisse/Status:** Offene Ergebnisse eines Iterations-Features werden zu einem neuen Iterations-Feature und der Iterationsplan entsprechend angepasst. (Alternativ kann ein vorhandenes zukünftiges Iterations-Feature erweitert werden, wenn es inhaltlich passt)

Use Cases versus Features



© 2006 by oose GmbH

Beispiel:

UC1 Neuen Kunden aufnehmen

UC2 Kunden-Daten ändern

UC3 Kunde aus dem Bestand nehmen

UC4 Kunden zeitweise sperren

UC5 Kunden-Sperre aufheben

IF1 nur primäre Personendaten sowie eine Anschrift ohne Details

IF2 mit Kontakt-Beschreibung, Geburtsdaten, mehrere Anschriften jedoch ohne Namenszusätze

IF3 mit detailliertem Kontaktjournal und Namenszusätzen

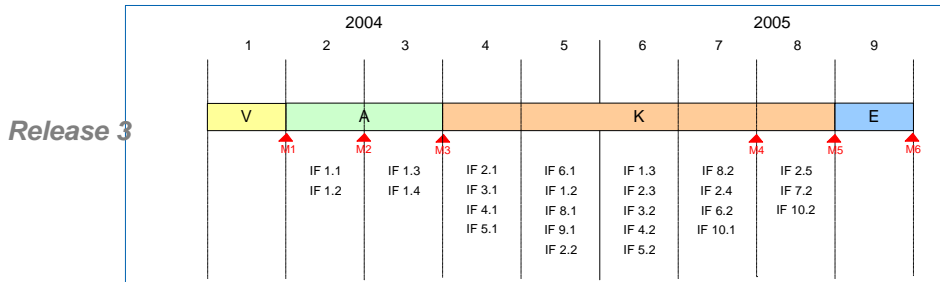
IF1-3 gelten nur für UC1-2 (nicht jedoch für UC3-5)

Dieser Zusammenhang ist auch als **Nachteil** des featurebasierten Planens zu verstehen, da eine zusätzliche Ebene unterschieden wird, d.h. **Planungsebene** und **Anforderungsebene** werden getrennt (was die Komplexität des Vorgehens erhöht)

Zeit vor Inhalt



Neue Denkweise: „*Bis wann haben wir Zeit?*
Was schaffen wir bis dahin?“



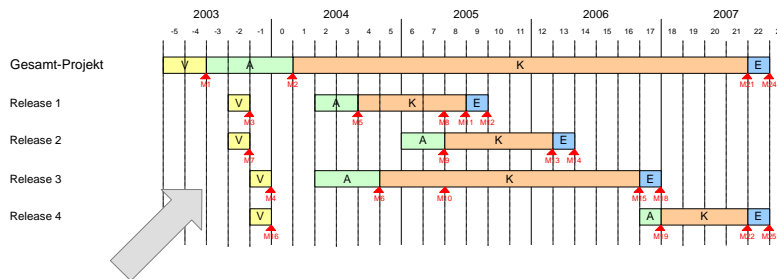
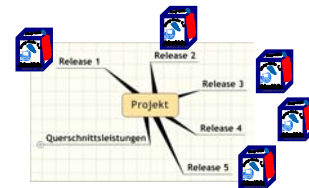
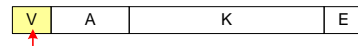
- **Transparenz** und öffentliche Vorausschau
- Je Iterationsende: Iterationsplan an neue Erkenntnisse anpassen

© 2006 by oose GmbH

- Denkprinzip: **Zeit vor Inhalt!** Es wird geliefert: egal was. Auch ein leeres Dokument ist ein Ergebnis.
- Der **Iterationsplan** zeigt je Release auf, welche Iterations-Features wann fertig sind (d.h. in welcher Iteration der Akzeptanztest stattfindet). Er zeigt nicht, wie das Team intern seine Arbeit organisiert, um das hinzubekommen, sondern er zeigt, wann der Kunde mit welcher Funktionalität rechnen kann.
- Je Release existiert zwar ein eigener Iterationsplan, ABER: für diese Release gelten dieselben Iterationsnummern und -termine!

1. Releaseplan aufstellen

- Releases und Querschnittsleistungen definieren
- Phasen festschreiben, Meilensteine beschreiben
- Projektstrukturplan beginnen



Abgekoppelte Vorbereitungsphasen der Releases

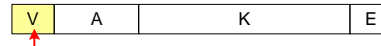
Sämtliche Teams im Gleichtakt!

- Iterationen frühzeitig für alle unverrückbar einmeißeln

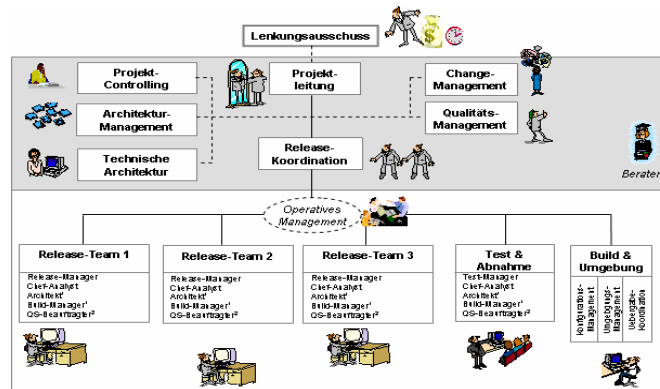
© 2006 by oose GmbH

- **Querschnittsleistungen** sind z.B. Projektmanagement, Release-Koordination, Build-Management usw.
- **Meilensteine:** Je Meilenstein ein Dokument, in welches die erwarteten Ergebnisse messbar beschrieben sind (ggf. unterschieden nach Muss- und Kann-Ergebnissen)
- **Projektstrukturplan** (engl. *work breakdown structure*, wbs) bricht ein Projekt hierarchisch in seine Bestandteile auf:
 1. Ebene: Releases und Querschnittsleistungen
 2. Ebene: Release-Features
 3. Ebene: Iterations-Features
 4. Ebene: Arbeitsaufträge
- Im Prinzip kann der **Iterationskalender** nach Abschluss der Vorbereitungsphase des Gesamtprojekts bis zum Laufzeitende festgelegt werden. Iterationstermine dürfen niemals verschoben werden!
- **Abgekoppelte Vorbereitungsphasen:** Architekturphase des Gesamt-Projekts beinhaltet alle Vorbereitungsphasen, d.h. der Vorlauf des Gesamt-Projekts ist entsprechend länger
- Wichtig: **Iterationstermine** gelten gleichermaßen für das gesamte Projekt! Die Dauer einer Iteration richtet sich nach der Anzahl der Projektmitarbeiter und dem Grad der Standortverteilung.
- Erfahrungsgemäß stellt sich sehr bald (nach einigen Wochen bis zu 2-4 Monaten) ein sehr mächtiger **Gleichtakt** im Gesamtprojekt ein, der bei gutem Gelingen bis zum Lenkungsausschuss und sogar in die Linie hineinstrahlt.

2. Projektorganisation etablieren



- 1 Team je zeitgleich zu entwickelndes Release
- Produktverantwortung übertragen
- **weniger ist mehr**: nicht zu früh in die Breite gehen!
- für die Release-Vorbereitungen nur die besten Mitarbeiter
- sofort Testen & Build-Management etablieren



© 2006 by oose GmbH

- Wird an 3 Releases zeitgleich gearbeitet, brauche ich 3 Release-Teams. Später wechseln Mitarbeiter je nach Fachgebiet das (Release-)Team. Mitarbeiter eines Teams sitzen idealerweise im selben Raum.
- Folge: Die **Projektorganisation** ist **dynamisch** und passt sich den erforderlichen Produktstrukturen an (und nicht umgekehrt).
- Das Release-Team ist **verantwortlich** für eine benutzbare, lauffähige Software, d.h. die Verantwortung endet nicht mit Abgabe einer kompilierfähigen Version an ein Integrationsteam.

Verantwortung heißt nicht, dass man es selbst tun muss, sondern nur, dass einem der Kopf abgerissen wird, wenn es nicht läuft...

- Es ist sinnfrei (und teuer!), **Busladungen von Entwicklern** zum Projektstart vorzufahren und mit den Hufen scharen zu lassen, wenn die Vorbereitungen noch nicht abgeschlossen sind.

Ein gutes Projekt beginnt mit einer guten Vorbereitung – das gilt für risikoreiche Großprojekte ganz besonders!

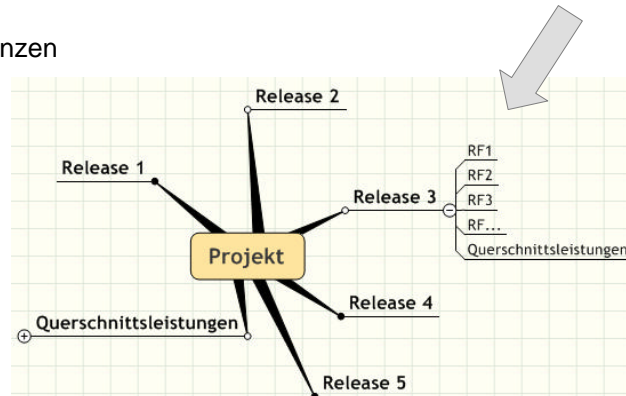
- Erfahrungsgemäß kommt dem Testen und dem Build-/Umgebungsmanagement regelmäßig zu wenig Bedeutung zu. **Testautomation** und ein eingespieltes **Umgebungsmanagement** aufzubauen ist sehr zeitaufwändig und mühevoll – fangen sie am besten sofort damit an!

3. Release-Features definieren

V	A	K	E
---	---	---	---

Je Release:

- Release-Features identifizieren (Richtwert: ca. 7-50)
- Beschreibungen anfertigen
- Inhalte mit dem Kunden abstimmen
- Aufwände schätzen
- Projektstrukturplan ergänzen



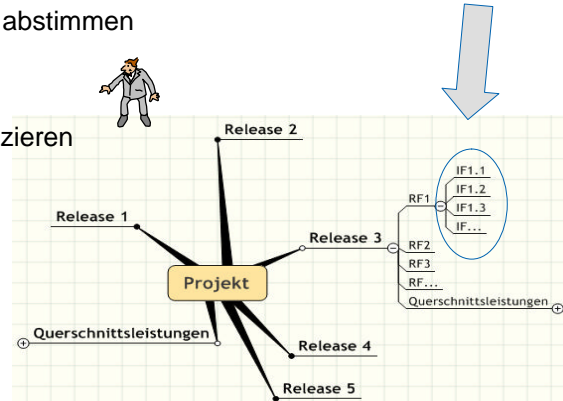
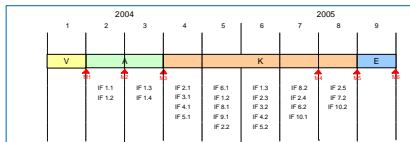
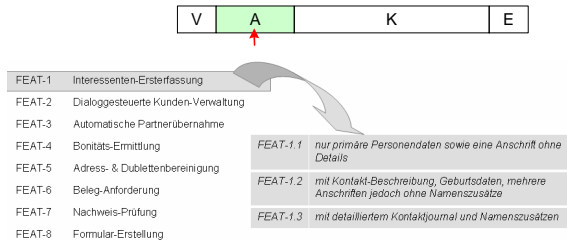
© 2006 by oose GmbH

Der **Richtwert** ist nur ein Richtwert! Das heißt, er hat u.U. große Schwankungsbreiten, da er von der Release-Komplexität abhängt. Wir haben auch schon Releases mit 60 Release-Features gesehen. Aber wenn es nahe 100 sind oder weniger als 3, sollten Sie stutzig werden...

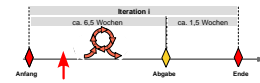
4. Iterations-Features definieren

Je Release-Feature:

- Iterations-Features bilden (ca. 7-10 je Release-Feature)
- Beschreibungen anfertigen
- Plan-Iterationen mit Nachbarteams abstimmen
- Aufwände schätzen
- Projektstrukturplan ergänzen
- Iterationsplan ergänzen + kommunizieren



5. Arbeitsaufträge definieren, abarbeiten, kontrollieren



vor Folge-Iteration:

- Arbeitsaufträge bilden (ca. 7-30 je Iterations-Feature)
- Aufträge beschreiben: Wer macht was bis wann?
- Zuliefertermine mit Nachbarteams abstimmen (sog. *Kieselsteine*)



während Fortschrittsabschnitt:

- Arbeitsaufträge abarbeiten
- Fortschrittskontrolle: wöchentliche Wahrscheinlichkeitsabfrage
- Probleme an Release-Koordination eskalieren

im Orientierungsabschnitt:

- Erreichte Ergebnisse abziehen
- Restaufwände schätzen, Fertigstellungsgrad errechnen
- Reviews + Lessons Learned durchführen
- Planungen verbessern



© 2006 by oose GmbH

- Jeder Arbeitsauftrag muss genau einem Iterations-Feature dienen (oder einen Querschnittsleistung)
Je nach Anzahl und Umfang der Arbeitsaufträge lassen sich diese auch als **letzte Gliederungsebene** unter den Iterations-Features ansiedeln
- Erst mit den Arbeitsaufträgen (also nur 1 bis 2 Iterationen im voraus) findet eine Zuordnung von Personen (Wer?) statt. Somit sind Arbeitsaufträge **konkrete Maßnahmen**, die notwendig sind, um ein Iterations-Feature zum Laufen zu kiregen.
- Jetzt dürfen endlich auch die Erarbeitung von Modellen, Konzepten usw. eingeplant werden...
- **Kieselsteine**: entweder der Abgabetermin einer Iteration oder auch wichtige Zwischentermine innerhalb einer Iteration.
- Im Jour Fix: **Wahrscheinlichkeitsabfrage** für Fertigstellung (nicht fragen: „Wie weit bist Du?“ – Antwort lautet immer: „fast fertig“)
Liegt die Fertigstellungswahrscheinlichkeit < 75% wird das Ziel überprüft und der Aufgabeninhalt ggf. angepasst

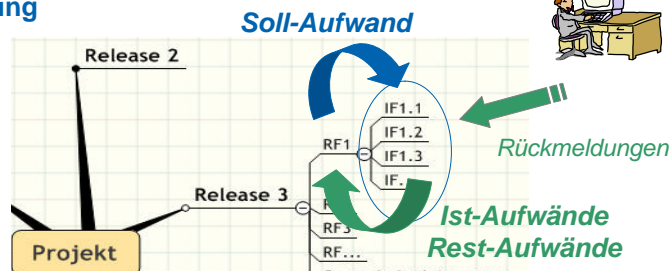
Featurebasiertes Controlling

Fertigstellungsgrad:

$$FG = \frac{Ist}{Ist + Rest}$$

korrt. Plan

$$\frac{90}{90+20} = 81\%$$

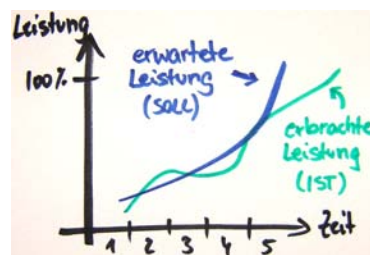


→ Was hat das Projekt bisher geleistet?

Erbrachter Wert (Earned Value):

$$EV = FG \cdot Soll$$

*81% · 50PT
≈ 40PT*



© 2006 by oose GmbH

Was ein laufendes Projekt bislang gekostet hat, ist meist klar (genauso wie die verstrichene Zeit). Aber was hat es bisher geleistet?

Soll-, Ist- und Rest-Größen in PT oder in EUR möglich

EV-Einheit ebenfalls in PT oder in EUR möglich

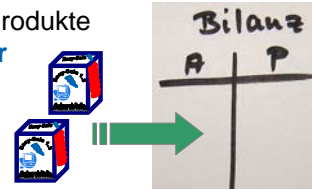
Weitere ableitbare Größen:

- Bisherige Produktivität (erbrachter Wert / bisherige Aufwände bzw. Kosten)
- Notwendige Produktivität (noch zu schaffender Wert / Restaufwand bzw. -budget)
- Bisherige Arbeitsgeschwindigkeit (erbrachter Wert / verstrichene Zeit)
- Notwendige Arbeitsgeschwindigkeit (noch zu schaffender Wert / Restzeit)
- Endtermin-Prognose (noch zu schaffender Wert * bisherige Arbeitsgeschwindigkeit)
- Gesamtaufwand- bzw. -budget-Prognose (noch zu schaffender Wert * bisherige Produktivität)

Vertragsaspekte



Releases, Features = echte Produkte
→ bilanztechnisch **aktivierbar**



Auftragsvergabe an Dritte:

- Höhere Transparenz: worin besteht die Leistung?
- Bessere Kontrolle: was wurde geleistet?
- Flexiblere Preisgestaltung für beide Seiten

Beispiel

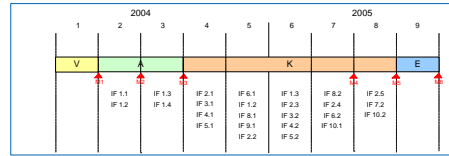
Agiler Festpreis:



Verbindlicher Gesamtpreis für gegebene Release-Features. Nicht realisierte Features jederzeit durch **gleichzeitige** Features **ersetzbar**.

- AG: absolute Budgetsicherheit.
- AG: Anforderungen inhaltlich flexibel änderbar
- Features können bei Vertragsschluss ganz oder teilweise offen bleiben.
- AN: höhere Gewinnchance, wenn Feature günstiger realisiert
- AN: Umsatzsicherheit.

Vorteile



Featurebasiertes Controlling:

- Echtes Leistungscontrolling
- Bilanztechnische Aktivierbarkeit

Release-orientierte Projektorganisation:

- Verantwortung für Projekterfolg
- Pragmatismus bei Zusammenarbeit

Iterations-Features:

- Teilprodukt am Iterationsende
→ Flexible Auslieferung
- Transparenz nach innen/außen

Release-Features als Gliederung:

- Produktgedanke:
→ Was ist dem Kunden wichtig?
- Vertragliche Vereinfachungen
- Laufende Software vor Dokumentation

Einheitliches Iterationsraster:

- Bessere Zuliefer-Synchronisation
- Verbesserung der Schätzgenauigkeit
- Macht des Gleichtakts nutzbar



Bernd Oestereich
- Geschäftsführer -



Herzlichen Dank!

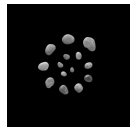


Christian Weiss
- Senior Consultant -

Steine...



Ein **Meilenstein** (*Milestone*) definiert einen für das Gesamtprojekt äußerst wichtigen Termin, zu dem eine Menge von Ergebnissen in vorgegebener Detaillierung und Vollständigkeit nachprüfbar und formal dokumentiert vorliegen muss. Liegen die Ergebnisse zum geplanten Termin nicht vor, wird der Meilenstein verschoben.



Ein **Kieselstein** (*Inchpebble*) definiert einen für mehrere Teams wichtigen Termin, zu dem eine Menge von Ergebnissen in vorgegebener Detaillierung und Vollständigkeit nachprüfbar vorliegen soll, damit die Zusammenarbeit der Teams reibungsarm verläuft (bspw. Zulieferleistungen). Liegen die Ergebnisse zum geplanten Termin nicht vor, wird ein neuer Kieselstein definiert.

© 2006 by oose GmbH

Ein **Meilenstein** ist ein Hilfsmittel zur Planung und Überwachung eines Entwicklungsprozesses.

Meilensteine sollten nur wenige existieren (max. 7 je Release) und allen jederzeit bekannt sein, um eine Zielfokussierung herbeizuführen. Je mehr Meilensteine existieren, desto stärker wird die Aufmerksamkeit gestreut.

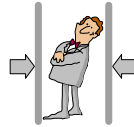
Ein **Kieselstein** ist ein Hilfsmittel zur Planung von Abhängigkeiten.

Kieselsteine existieren viele. Jedes Team muss nur diejenigen Kieselsteine kennen, von denen es selbst direkt betroffen ist.

Grundbegriffe



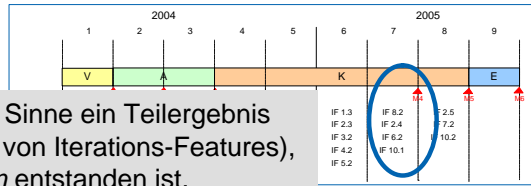
Eine **Phase** ist ein zeitlich bzw. sachlogischer Gliederungsabschnitt eines Projekts. Eine Phase fasst eine Menge von Aktivitäten und Ergebnissen zu einer Planungs- und Kontrolleinheit zusammen. Am Ende jeder Phase steht gewöhnlich ein → *Meilenstein*, der die in der Phase zu erzielende Inhalte definiert.



Eine **Iteration** ist ein in ähnlicher Weise mehrfach vorkommender Zeitabschnitt in einem Prozess. Iterationen sind in der Regel → *Timeboxen*. Am Ende einer Iteration steht gewöhnlich ein → *Kieselstein*.

Eine **Timebox** definiert einen unverrückbaren Zeitrahmen, in dem eine Menge von Ergebnissen in vorgegebener Detaillierung und Detaillierung vorliegen soll. Liegen die Ergebnisse nicht wie geplant vor, werden die offenen Teile in eine nachfolgende Timebox verschoben. Hierzu werden zum geplanten Endtermin die tatsächlich erreichten Ergebnisse bestimmt.

Grundbegriffe



Ein **Inkrement** ist im weiteren Sinne ein Teilergebnis (im engeren Sinne die Menge von Iterations-Features), das am Ende einer → *Iteration* entstanden ist.



Ein **Build** ist eine gewöhnlich unvollständige und temporäre Version eines in Entwicklung befindlichen Systems.



Ein **Gewerk** ist die werksvertragliche Definition eines bestimmten Umfangs an Software-Funktionalität, die zu einem vereinbarten Termin in evtl. bereits bestehende Systeme integriert und benutzbar sein soll. Der Begriff wird vor allem in der Kommunikation zum Auftraggeber verwendet.

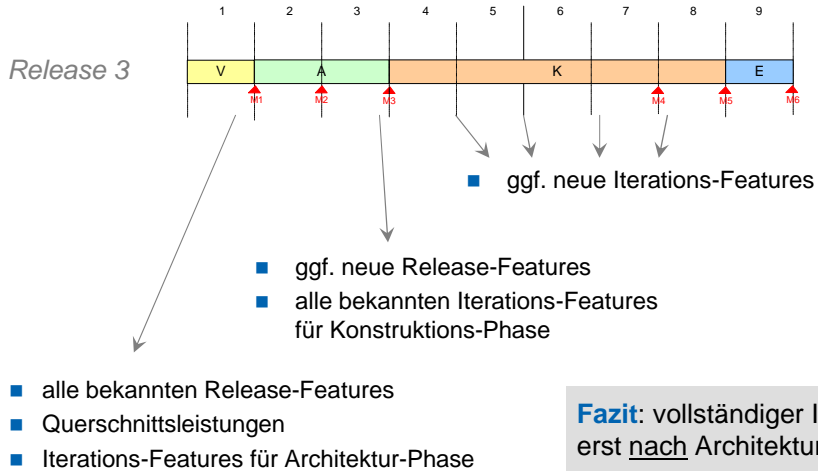


Ein **Release** ist ein spezielles → *Build*, das an den Auftraggeber ausgeliefert werden kann, d.h. es ist produktreif. Geht es in Produktion, entspricht es einem anfassbaren → *Gewerk* und heißt **Produktions-Release**. Der Begriff wird vor allem in der IT benutzt.

Das Produktions-Release ist gewissermaßen die technische Umsetzung des Gewerkvertrags

Zeitliche Entwicklung des Iterationsplans

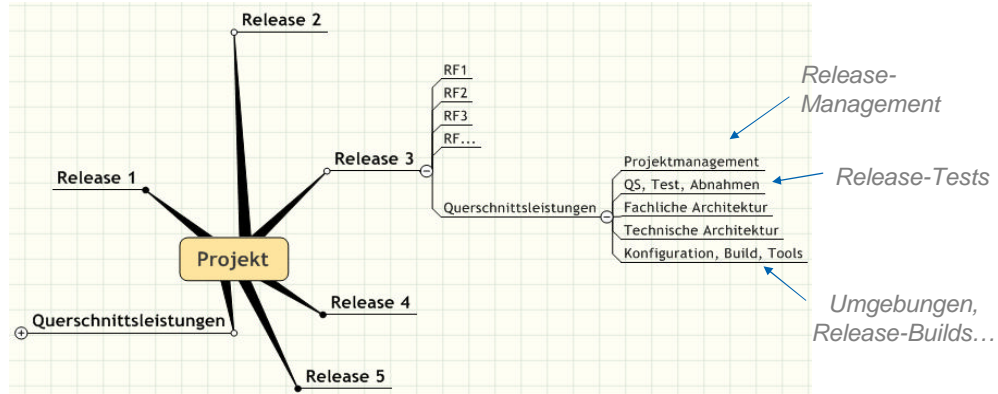
Wann entsteht welcher **Detaillierungsgrad** des Iterationsplan?



Fazit: vollständiger Iterationsplan erst nach Architektur-Phase!

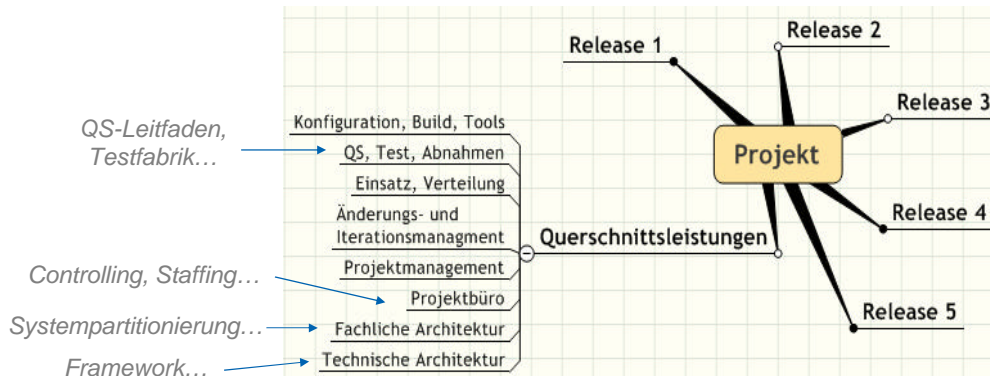
Release-Querschnittsleistungen

querschnittliche Release-Aufwände: direkt dem Release zurechenbar



Projekt-Querschnittsleistungen

querschnittliche Projekt-Aufwände: keinem Release zurechenbar

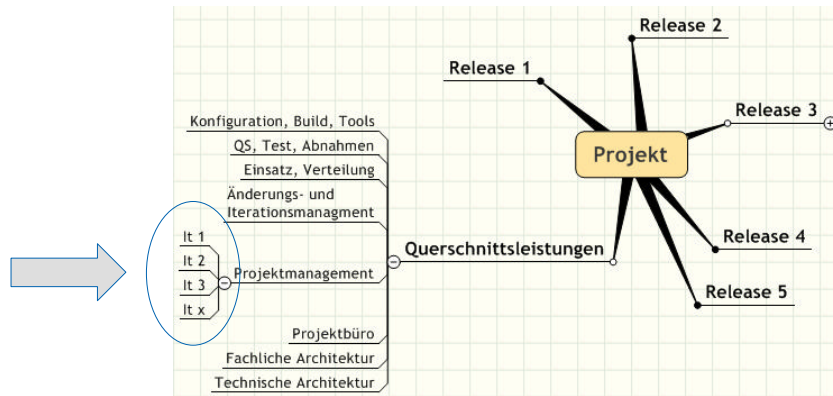


- **Disziplinen** für Gliederung wählen
- typisch: Management-Organisationsstrukturen = Disziplinen
- %-Verteilung von Ist-Aufwänden auf Releases

Querschnittsleistungen als Buchungskonten

Controlling je Iteration?

→ auch Querschnittsleistungen nach Iterationen aufteilen



Iterations-Aufwände für Querschnittsleistungen meist **leicht zu ermitteln**
(Personenanzahl, Arbeitstage bekannt)

© 2006 by oose GmbH

Meist ist die Personenanzahl bekannt (bspw. 2 Gesamt-Projektleiter, 4 Vollzeitkräfte im Projektbüro usw.)

Auch die Arbeitstage sind im voraus bekannt, da die Iterationen bereits feststehen.

Zieht man die geplanten Urlaubstage ab (die meist auch länger bekannt sind), lässt sich der geplante Iterationsaufwand für Projekt-Querschnittsleistungen leicht ausmultiplizieren.