

Software Management

8. Kontrolle: Metriken, Konfigurations- und Änderungsmanagement

Prof. Dr. K.-P. Fähnrich

17.06.2008

Übersicht der Vorlesung

1. Grundlagen
2. Planung
3. Organisation: Gestaltung
4. Organisation: Prozess-Modelle
5. Personal
6. Leitung
7. Innovationsmanagement
8. Kontrolle: Metriken, Konfigurations- und Änderungsmanagement
9. CASE
10. Wiederverwendung
11. Sanierung

Gliederung

1. Grundlagen

2. Metriken definieren, einführen und anwenden

3. Konfigurationsmanagement etablieren

3.1. Konfigurationen

3.2. Versionen und ihre Verwaltung

3.3. Varianten

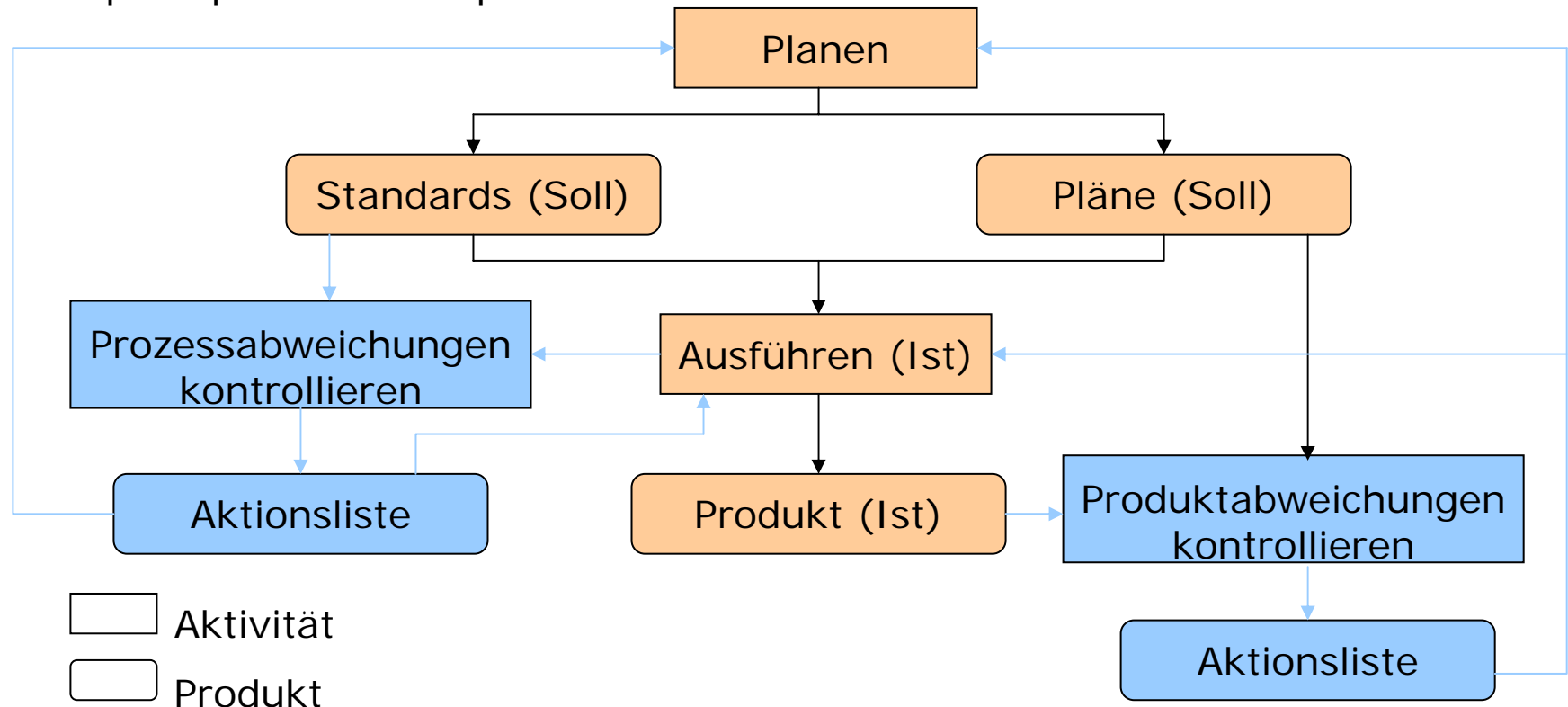
3.4. Konfigurations- und Änderungsmanagement

Begleitliteratur: Helmut Balzert, Lehrbuch der Software-Technik

Quelle der Grafiken und Tabellen: Helmut Balzert, Lehrbuch der Software-Technik,
wenn nicht anders angegeben

1. Grundlagen

- Zur **Kontrolle** gehören alle Management-Aktivitäten, die sicherstellen, dass die laufenden Tätigkeiten mit dem Plan übereinstimmen.
- **Pläne** legen Produkthanforderungen, Zeit und Kosten fest.
- Unter **Standards** werden für die Ausführung benötigte Prozessmodelle, Richtlinien, Methoden und Werkzeuge zusammengefasst.
- Der prinzipielle Kontrollprozess:



1. Grundlagen

- Methoden und Werkzeuge zur Kontrolle müssen:
 - objektiv,
 - anpassbar und
 - ökonomisch sein.
- Abweichungen vom Plan und den Standards müssen ohne Rücksicht auf die betroffenen Mitarbeiter und Stellen aufgezeigt werden.
- Kontrolle muss zu korrigierenden Aktionen führen, entweder um den Prozess zu den Standards und den Plan zurückzubringen, die Standards und den Plan zu ändern oder den Prozess zu beenden.
- Mit der Kontrolle sind folgende Probleme verbunden:
 - Viele Kontrollmethoden nehmen die bisher benötigte Zeit und die bisher verbrauchten Kosten als Maßstab für die Entwicklungsfortschritt.
 - Standards für Entwicklungsaktivitäten sind entweder nicht vorhanden oder nicht schriftlich fixiert.
 - Eine Software-Messtechnik, die Software-Maße über den Entwicklungsprozess und das Produkt bereitstellt, ist noch nicht voll entwickelt.

Aufgaben eines Software-Managers

1. Standards entwickeln und festlegen
2. Kontroll- und Berichtssystem etablieren
3. Prozess und Produkte vermessen
4. Korrigierende Aktionen initiieren
5. Loben und Tadeln

Aufgaben eines Software-Managers (1)

1. Standards entwickeln und festlegen

- Standards können für eine gesamte Firma oder für jeweils ein Produkt verbindlich vorgeschrieben werden.
- Werden selbst entwickelt oder von Organisationen oder anderen Firmen übernommen.
- Folgende Teilaktivitäten sind durchzuführen:
 - Entwicklung von Qualitäts- und Quantitätsstandards,
 - Festlegen des Prozessmodells,
 - Festlegen der QS-Methoden,
 - Entwicklung von Produktivitäts-, Qualitäts- und Prozessmetriken,
 - Standards sollten immer immer daraufhin überprüft werden, ob sie folgende Vorteile mit sich bringen:

Vorteile

- Senkung der Einarbeitungs- und Umschulungskosten.
- Verbesserung der Kommunikation zwischen Teammitarbeitern.
- Erleichterung des Personalaustauschs zwischen Projekten.
- Erfahrungen können besser weitergegeben werden.
- Einheitliche Anwendung der besten Erfahrungen erfolgreicher Projekte.
- Vereinfachung der Wartung und der Pflege.
- Kontrolle der Standards.

Aufgaben eines Software-Managers (2)

2. Kontroll- und Berichtssystem etablieren

- Kontroll- und Berichtssysteme müssen ausgesucht oder entwickelt werden, um den Entwicklungsprozess zu überwachen und jederzeit den Entwicklungsstatus bestimmen zu können.
- Für Entwicklungsbereiche ist der Typ, die Häufigkeit, der Ersteller und der Empfänger festzulegen.
- Die Art und der Umfang von Kontroll- und Berichtssystemen hängen von folgende Parametern ab:
 - Verwendeter Führungsstil (Management-by ...),
 - Verwendete Aufbau- und Ablauforganisation (Prozessmodelle),
 - Umfang der Entwicklungszeit (Zeit, Anzahl, Mitarbeiter),
 - Eingesetzte CASE-Umgebung.

Aufgaben eines Software-Managers (2)

Methode	Definition und Erläuterung
Kontrolle des Prozesses	
• Budget-überprüfung	Vergleich des geschätzten Budgets mit den aktuellen Ausgaben, um Übereinstimmungen mit oder Abweichungen von dem Plan festzustellen.
• Meilenstein-überprüfung	Überprüfung des Prozesses und des Projekts an den Meilensteinen, die in der Planung vorgesehen sind.
• Verfolgung der Top-ten Risiken	Risiko-Überwachung durch Konzentration auf die jeweiligen 10 kritischen Risiken
• Qualitäts-sicherung	Geplante und systematische Überprüfungen des Prozesses auf Einhaltung der vorgegebenen Prozessstandards.
Kontrolle des Produkts	
• Konfigurations-management	Methode zur Kontrolle eines Software-Status
• Qualitäts-sicherung	Geplante und systematische Überprüfung des Produkts auf Einhaltung der vorgegebenen Qualitätsziele

Kontroll- und Berichtssysteme

Aufgaben eines Software-Managers (2)

Berichtstyp

Definition oder Erläuterung

- | | |
|--|--|
| • Budgetbericht | Vergleicht das Budget mit den Ausgaben. |
| • Terminübersicht | Vergleicht den Terminstatus mit den fertiggestellten Meilensteinen. |
| • Mitarbeiterstunden/
Aktivitäten-Bericht | Zeigt die Anzahl der Stunden, die benötigt wurden, um eine Aktivität durchzuführen. |
| • Meilensteinfällig-
keitsbericht | Gibt einen Status über die erreichten und überfälligen Meilensteine. |
| • Projektfortschritts-
bericht | Ein nicht formalisierter Bericht über den Projektfortschritt. |
| • Aktivitätenbericht | Projektbezogener Bericht, der angibt, welche Aktivitäten in der Periode erledigt wurden. |
| • Trenddiagramm | Zeigt Ausnahmen vom Plan und signifikante positive und negative Veränderungen. |
| • Änderungsbericht | Periodischer Bericht, der pro Periode die 10 Risiko-
Elemente mit den größten Risiken angibt. |
| • Top-ten-Risiko-
elementenliste | |

Berichtstypen (nach [Thayer 90], [Boehm 91])

Aufgaben eines Software-Managers (2)

- Zur Aufwandsermittlung pro Phase kann eine einfache Strichliste verwendet werden, die ökonomisch ausgefüllt werden kann. Die Auswertung trägt wesentlich dazu bei, Aufwandsschätzungen für neuere Projekte genauer vorzunehmen.
- Hat man die Projektplanung mit Hilfe eines Planungssystem vorgenommen, kann dieses auch für die Projektkontrolle verwendet werden.
- Terminpläne können detailliert überwacht werden:
 - Anfangs- und Endtermine sowie Dauer von Vorgängen,
 - Prozentsatz, zu dem Vorgänge bereits abgeschlossen sind,
 - Kosten des Projekts, einzelner Vorgänge und Ressourcen,
 - Arbeitsstunden, die von jeder Ressource ausgeführt wurden.

Aufgaben eines Software-Managers (2)

Projektstatistik	Monat/Jahr:		
Projekt:	Mitarbeiter:		
Phase	Phasenplanung	Phasenrealisierung	Phasenüberprüfung
Planung			
Definition			
Entwurf			
Implementierung			
Abnahme & Einführung			
Wartung & Pflege			
Projektleitung			
Schulung, Einarbeitung			
Dienstreisen, Tagungen			

Strichliste zur Erfassung des Aufwands je Phase

Aufgaben eines Software-Managers (3, 4)

3. Prozess und Produkte vermessen

- Für die Qualitätsüberprüfung sowohl des Prozesses als auch des Produktes ist die Qualitätssicherung zuständig.
- Sind entsprechende Messverfahren etabliert, so muss das Software-Management sicherstellen, dass die Messungen konsequent durchgeführt werden.

4. Korrigierende Aktionen initiieren

- Plan- und Standard-Änderungen können zu zusätzlichen Budget-Anforderungen, zusätzlichen Mitarbeitern oder leistungstärkeren Arbeitsplatzrechnern führen.
- Manchmal ist es möglich einen Teil des Plans einzuhalten, wenn die Zeit für die Einhaltung eines anderen Plans vergrößert wird.
- Als letzte Maßnahme können auch Anforderungen an das Produkt geändert werden.

Aufgaben eines Software-Managers (5)

5. Loben und Tadeln

- Mitarbeiter, die den Plan und die Standards einhalten, sollten vom Manager gelobt, die anderen getadelt werden.
 - Lob sollte sich insbesondere durch nichtmonetäre Belohnungen ausdrücken, z.B. einen zusätzlichen freien Tag, eine Kongressreise u.ä..
-
- Die Aufgaben **1** und **2** sind in der Regel einmal für die gesamte Softwareentwicklung durchzuführen.
 - Die Aufgaben **3** bis **5** sind permanent für jede Softwareentwicklung durchzuführen.

Gliederung

1. Grundlagen
2. Metriken definieren, einführen und anwenden
3. Konfigurationsmanagement etablieren
 - 3.1. Konfigurationen
 - 3.2. Versionen und ihre Verwaltung
 - 3.3. Varianten
 - 3.4. Konfigurations- und Änderungsmanagement

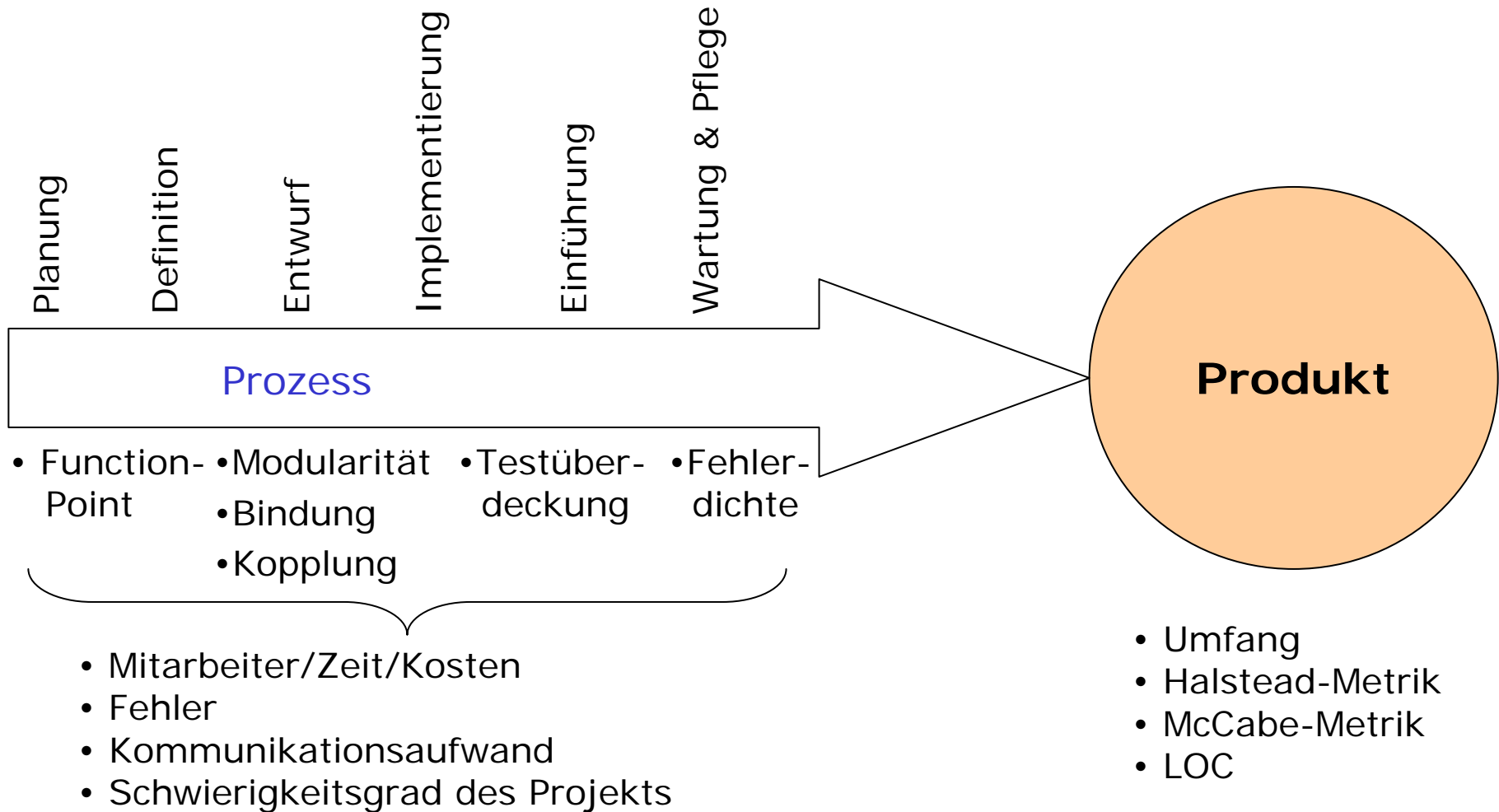
2. Metriken definieren, einführen und anwenden

Einleitung

Software-Metrik: Definiert, wie eine Kenngröße eines Software-Produkts oder eines Software-Prozesses gemessen wird.

- Motto: „You can't control what u can't measure“.
- Software-Metriken werden ermittelt, um den Software- und Qualitätsmanagement quantitative Angaben über den Softwareentwicklungsprozess und das Softwareprodukt zur Verfügung zu stellen.
- Vorgehensweise für das Messen [Wallmüller 90]:
 1. Definition der Messziele.
 2. Ableitung der Messaufgaben aus den Messzielen.
 3. Bestimmung der Messobjekte.
 4. Festlegen der Messgröße und Messeinheit.
 5. Zuordnung der Messmethoden und Messwerkzeuge zu den Messobjekten und Messgrößen.
 6. Ermittlung der Messwerte.
 7. Interpretation der Messwerte.

2. Metriken definieren, einführen und anwenden



Typische Kenngrößen von Software-Prozessen und Produkten

2. Metriken definieren, einführen und anwenden

- Als Beispiel für das Messen wird das Ermitteln von nicht-kommentierten Quellanweisungen betrachtet:
 - **Messziel:** Bestimmung der Anzahl nichtkommentierter Quellanweisungen NCSS (non-commented source statements).
 - **Messaufgabe:** Zählen der Anzahl NCSS eines Programms.
 - **Messobjekt:** Ausgewähltes Programm
 - **Kenngroße:** Anzahl der Quellanweisungen einschließlich Compiler-Anweisungen, Datendeklarationen, ausführbaren Anweisungen, aber ohne Leerzeilen oder Zeilen, die vollständig aus Kommentaren bestehen. Jede Codezeile wird einmal gezählt. Jede enthaltene Datei wird einmal gezählt.
 - **Messeinheit:** $KNCSS = 1000 \text{ NCSS}$
 - **Messmethoden/ Messwerkzeuge:** Automatischer Zeilenzähler
 - **Interpretation:** Repräsentiert den Umfang der produzierten Software.

Gütekriterien für Software-Metriken (1)

- Um von einem Software-Maß sprechen zu können, muss eine Software-Metrik bestimmte Gütekriterien erfüllen.
-
1. **Objektivität** (Intersubjektivität)
Ein Maß ist objektiv, wenn keine subjektiven Einflüsse des Messenden auf die Messung möglich sind.
 2. **Zuverlässigkeit** (Messgenauigkeit)
bei der Wiederholung der Messung unter denselben Messbedingungen werden die selben Messergebnisse erzielt.
 3. **Validität** (Gültigkeit, Messtauglichkeit)
Die Messergebnisse erlauben einen eindeutigen und unmittelbaren Rückschluss auf die Ausprägung der Kenngröße.
 4. **Normierung**
Es gibt eine Skala, auf der die Messergebnisse eindeutig abgebildet werden. Gibt es eine Vergleichbarkeitsskala, dann ist ein Maß normiert.
 5. **Vergleichbarkeit**
Kann ein Maß mit anderen in Relation gesetzt werden, dann heißt es vergleichbar.

Gütekriterien für Software-Metriken (2)

5. **Ökonomie**

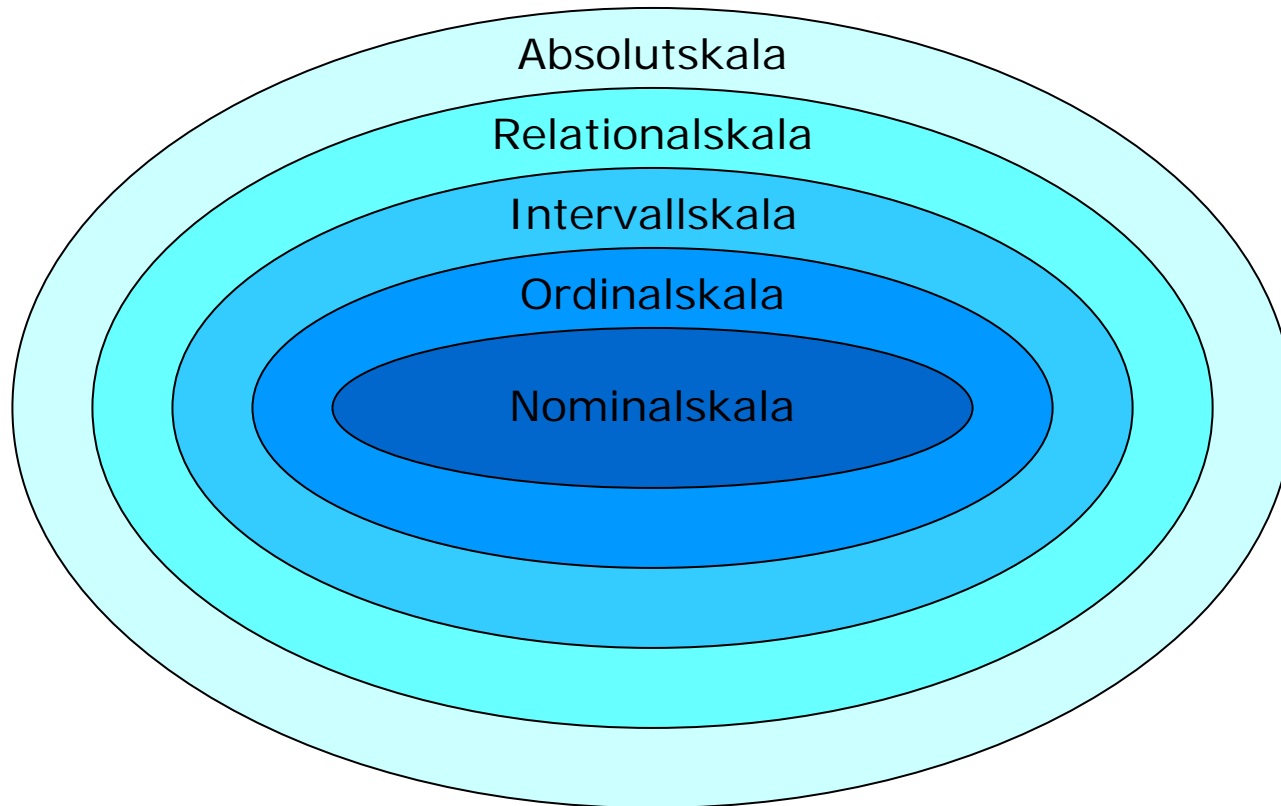
Die Messung muss mit geringen Kosten durchgeführt werden können. Die Ökonomie hängt vom Automatisierungsgrad, der Anzahl der Messgrößen und der Anzahl der Berechnungsschritte ab.

6. **Nützlichkeit**

Werden mit einer Messung praktische Bedürfnisse erfüllt, dann ist ein Maß nützlich.

-
- Das am schwierigsten nachzuweisende Kriterium ist die Validität, d. h. der Nachweis, dass das Maß tatsächlich das misst, was es vorgibt zu messen.
 - Ist eine Funktion von der realen Welt in eine formale numerische Welt vorhanden, dann gibt es auch eine Skala.
 - Skalen unterscheiden sich durch die zulässigen Transformationen, die auf ihnen durchgeführt werden können.

2. Metriken definieren, einführen und anwenden



Skalenhierarchie nach [Zuse 85]

2. Metriken definieren, einführen und anwenden

Metriken lassen sich klassifizieren.

1. **Objektiv/Subjektiv**
 - Objektiv: Programmumfang, verbrauchte Zeit für eine Phase, Fehleranzahl
 - Subjektiv: Kundenzufriedenheit (Daten aus Interviews u.ä.)
2. **Absolut/Relativ**
 - Absolut: invariant gegenüber dem Hinzufügen neuer Elemente
 - Relativ: ändern sich (z.B. der Durchschnitt, oder die Steigung einer Kurve)
3. **Explizit/Abgeleitet**
 - Explizit: (auch Basis-Metriken) z.B. „Entdeckte Fehler in einem Programm“
 - Abgeleitet: Anzahl der Fehler / Anzahl von tausend Quellzeilen
4. **Dynamisch/Statisch**
 - Dynamisch: Zeitdimension; z.B. gefundene Fehler pro Monat
 - Statisch: invariant; z.B. gefundene Fehler während der gesamten Entwicklungszeit
5. **Vorhersagend/Erklärend**
 - Vorhersagend: Können im voraus ermittelt oder generiert werden.
 - Erklärend: Werden hinterher ermittelt oder generiert.
6. **Prozessorientiert/Produktorientiert**
 - Prozessorientiert: z.B. Kosten der Entwicklung oder der Entwicklungsumgebung
 - Produktorientiert: Werden am produkt gemessen z.B. Umfang des Produkt.
7. **Global/Speziell**
 - Global: Umfassen mehrerer Phasen des Softwareentwicklungsprozesses
 - Speziell: Indikatoren für eine spezielle Phase im Entwicklungsprozess

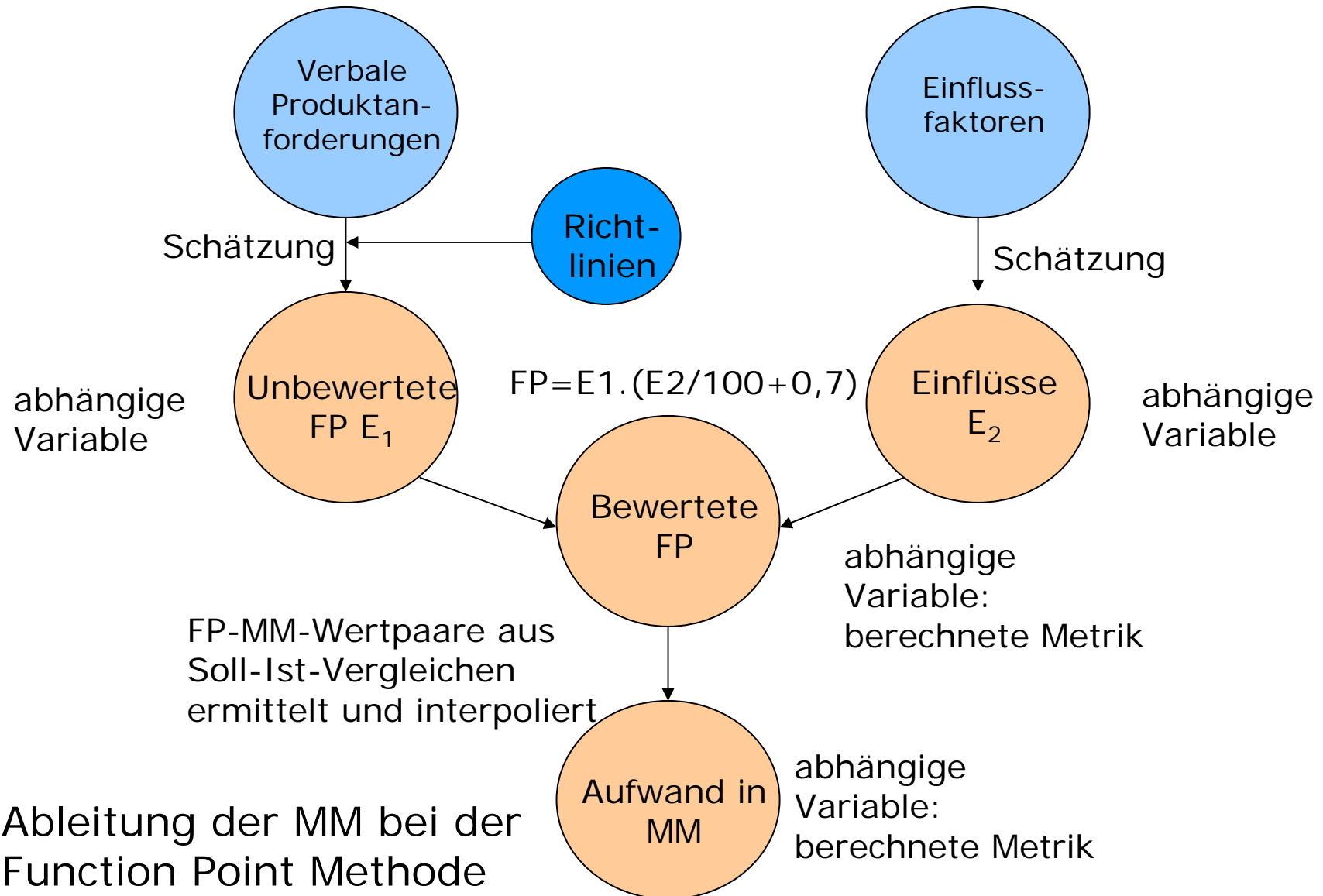
Function Point Methode (1)

- Mit Hilfe der **Function Point Methode** versucht man den Aufwand einer Software-Entwicklung in Abhängigkeit von Art und Umfang der Produkt-anforderungen sowie den Schwierigkeitsgrad des Produkts zu ermitteln.

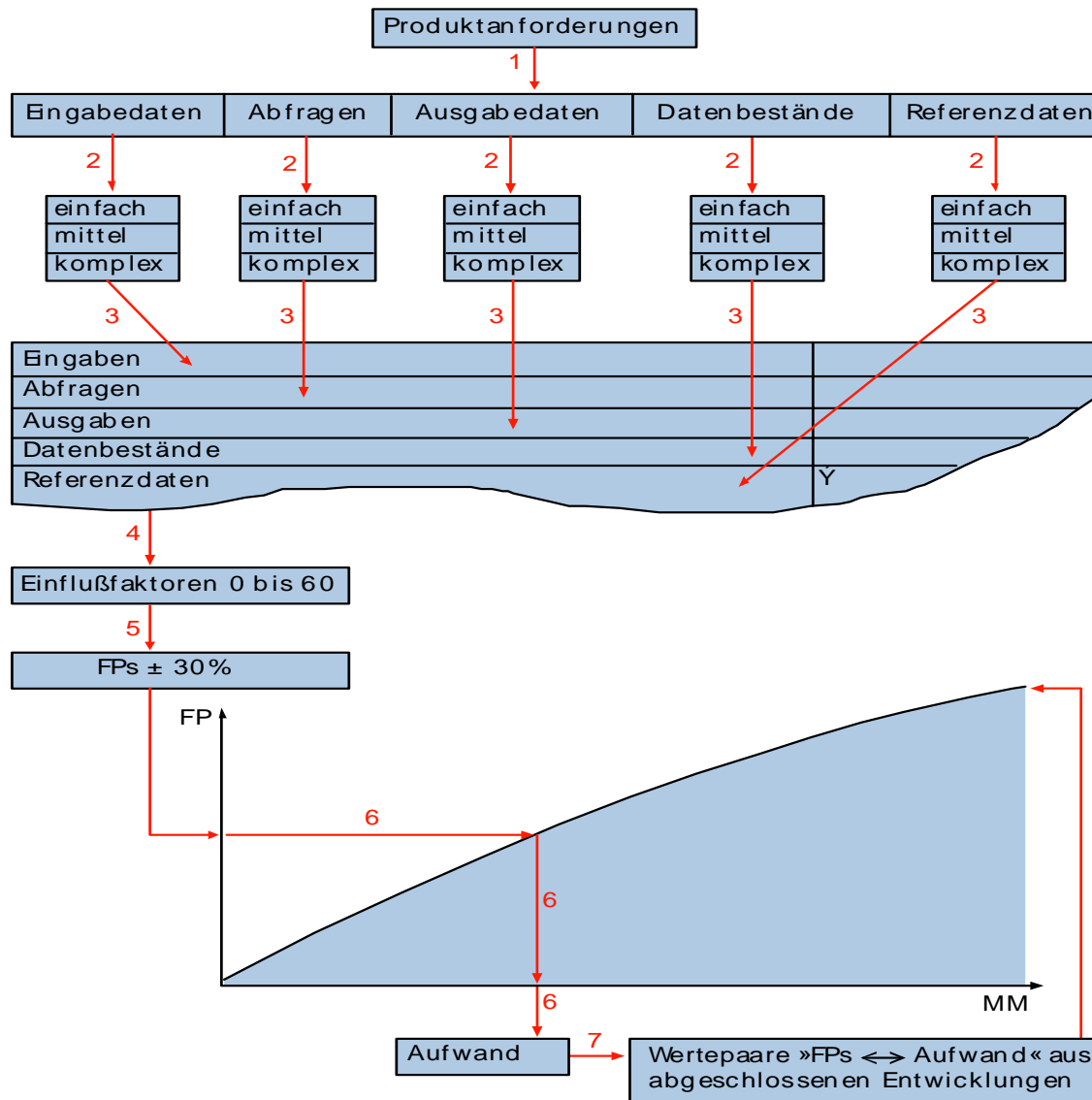
Aufwand = f (Art, Umfang, Schwierigkeitsgrad)

- Der grundsätzliche Zusammenhang zwischen den Kenngrößen Art, Umfang und Schwierigkeitsgrad und dem Aufwand wurde vom Erfinder der Function Point -Methode postuliert. Die konkrete Formel wurde dann mittels statistischer Analyse ermittelt.
- Im Falle der Function Point -Methode wurden Art und Umfang aus den verbalen Produktanforderungen ermittelt und der Schwierigkeitsgrad anhand eines Kriterienrasters geschätzt.

Function Point Methode (3)



Vorgehensweise bei der FP-Methode



- Schritt:** Jede Anforderung einer Kategorie zuordnen
- Schritt:** Klassifizierung
 - Jede Anforderung in eine der Klassen einfach, mittel oder komplex einordnen.
- Schritt:** Eintrag in Berechnungsformular
- Schritt:** Bewertung der Einflussfaktoren
- Schritt:** Berechnung der bewerteten Function Points
- Schritt:** Ablesen des Aufwands in MM
 - Voraussetzung: Empirische Ermittlung der Zuordnung FP ↔ MM
- Schritt:** Aktualisierung der empirischen Daten.

Function Point Methode (2)

- IBM-Tabelle

<i>Function P.</i>	IBM-MM	<i>Function P.</i>	IBM-MM	<i>Function P.</i>	IBM-MM
50	5	700	52	1700	142
100	8	75	56	1800	153
150	11	800	60	1900	164
200	14	850	64	2000	175
250	17	900	68	2100	188
300	20	950	72	2200	201
350	24	1000	76	2300	215
400	28	1100	85	2400	230
450	32	1200	94	2500	245
500	36	1300	103	2600	263
550	40	1400	112	2700	284
600	44	1500	122	2800	307
650	48	1600	132	2900	341.

Gliederung

1. Grundlagen
2. Metriken definieren, einführen und anwenden
3. Konfigurationsmanagement etablieren
 - 3.1. Konfigurationen
 - 3.2. Versionen und ihre Verwaltung
 - 3.3. Varianten
 - 3.4. Konfigurations- und Änderungsmanagement

3. Konfigurationsmanagement etablieren

Probleme bei der Software-Entwicklung

- Häufige Änderungen an Software-Elementen verursachen ein Chaos. Bereits korrigierte Fehler tauchen wieder auf, es ist unklar von wem welche Änderungen durchgeführt wurden.
→ *Reduktion des Problems durch Aufzeichnung der Historie.*
- Es ist unklar, ob ein Fehler bereits behoben wurde oder nicht.
→ *Verknüpfung von Änderungswünschen und vorgenommenen Änderungen.*
→ *Überwachung des Änderungsprozesses*
- Es ist schwierig, das System so zu konfigurieren, dass alle Fehlermeldungen bis vor zwei Wochen berücksichtigt sind. Die letzten „Verbesserungen“ waren fehlerhaft, sie müssen aus der Konfiguration entfernt werden.
→ *Automatische Versions- und Konfigurationsselektion.*
- Man ist unsicher, ob alles neu übersetzt wurde und ob die Kunden die neueste Freigabe haben.
→ *Software-Konfigurationsmanagement sorgt dafür, dass kein Arbeitsschritt bei der Vor- und Nacharbeitung von Software-Elementen vergessen wird.*

Software-Element: Jeder identifizierbarer und maschineller Bestandteil des entstehenden Produkts oder der entstehenden Produktlinie.

Software-Konfiguration: Benannte und formal freigegebene Menge von Software-Elementen, mit den jeweils gültigen Versionsangaben, die zu einem bestimmten Zeitpunkt im Produktlebenszyklus in ihrer Wirkungsweise und den Schnittstellen aufeinander abgestimmt sind und gemeinsam eine vorgesehene Aufgabe erfüllen sollen.

- Klassifikation von Software-Elementen nach ihrer Entstehungsart:
 - Quellelement (wird durch manuelle Eingaben erzeugt, z. B. unter Zuhilfenahme eines Editors)
 - Abgeleitetes Element (Software-Element, das vollautomatisch durch ein Programm erzeugt wurde, z. B. Objektcode)
- Klassifikation von Software-Elementen nach ihrer Struktur:
 - Atom (Software-Element, das für eine Softwarekonfiguration eine unteilbare Einheit bildet.)
 - Konfigurationen (Software-Element, das aus mehreren anderen Software-Elementen zusammengesetzt ist.)

Konfigurations-Identifikationsdokument

Konfigurations-Identifikationsdokument (KID):

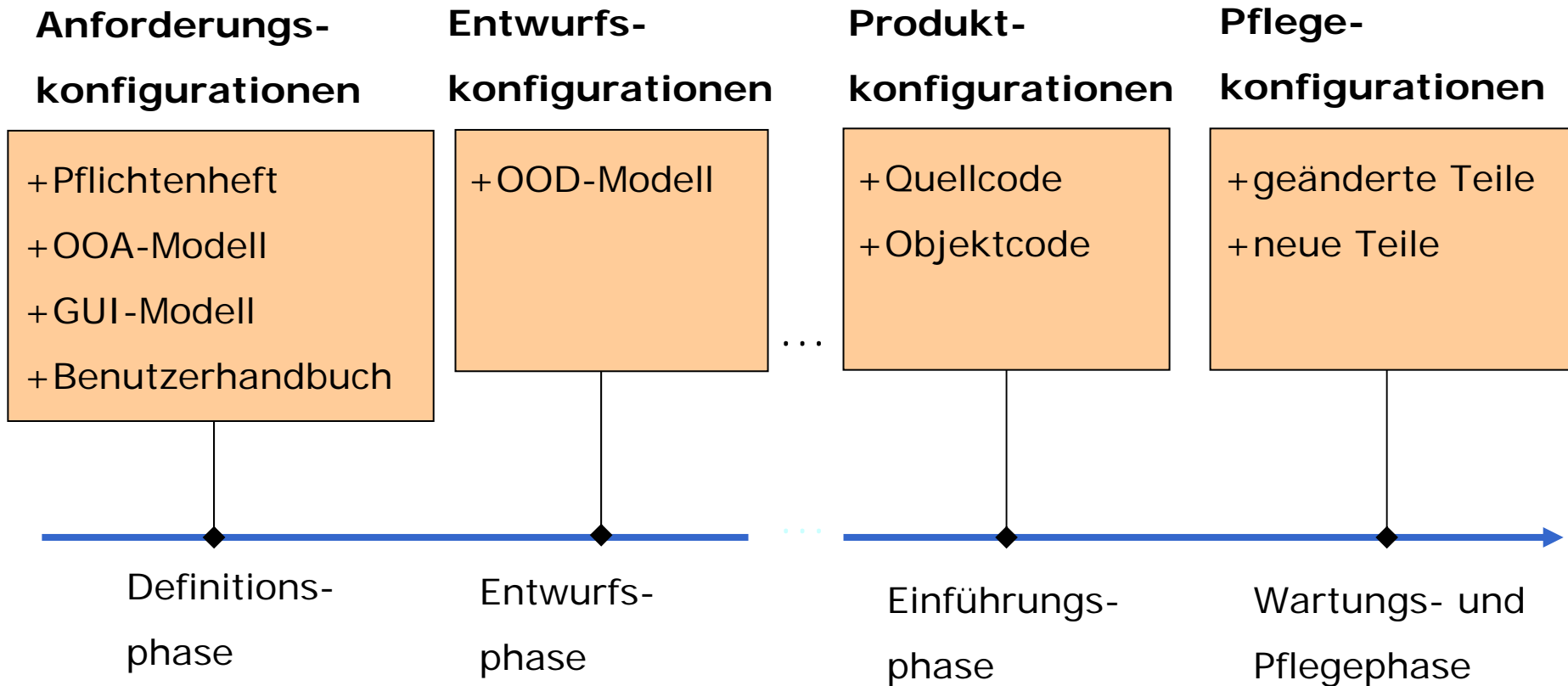
Hier wird für eine Konfiguration aufgeführt, welche Software-Elemente zu ihr gehören. Ein solches Dokument wird auch Konfigurationshierarchie oder Elementstrukturplan genannt.

- In der Regel werden auch solche Software-Elemente aufgeführt, die als Hilfsmittel und Werkzeuge zur Erstellung verwendet wurden, aber nicht an den Auftraggeber bzw. Käufer ausgeliefert werden.

Referenzkonfiguration (baseline): Ein zu einem bestimmten Zeitpunkt im Entwicklungsprozess ausgewähltes, gesichertes und freigegebenes Zwischenergebnis.

- Jede Änderung führt zu einer neuen Konfiguration, die in einem neuen Konfigurations-Identifikationsdokument festgehalten wird.

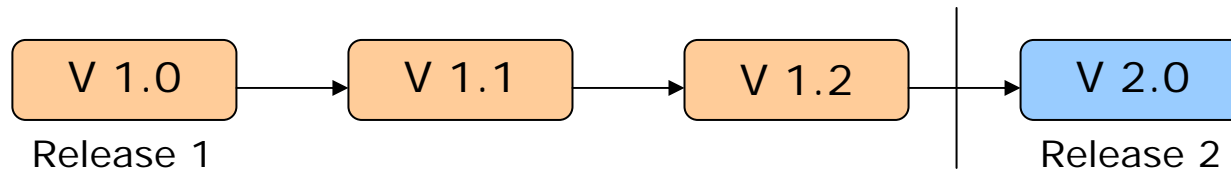
Konfigurationstypen



3.2 Versionen und ihre Verwaltung

Version: Ist gezeichnet durch die Ausprägung eines Software-Elements zu einem bestimmten Zeitpunkt. Unter Versionen werden zeitlich nacheinander liegende Ausprägungen eines Software-Elements verstanden und in der Regel durch eine Nummer beschrieben.

- Bestandteile einer Versionsnummer:
 - Release-Nummer
 - Level-Nummer



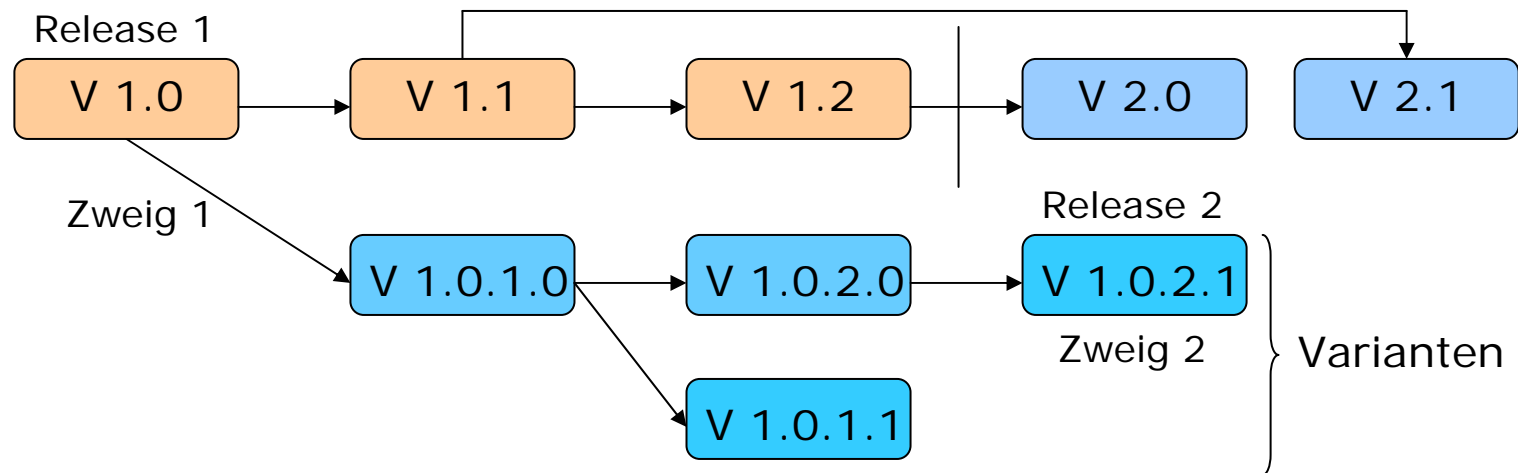
- Verbreitetstes Modell zur Versionsverwaltung: **Checkin/Checkout-Modell**. Software-Elemente werden in diesem Modell in Archiven gesammelt.
- Checkout-Operation: Holt eine Kopie des Elements aus dem Archiv und reserviert sie für den Ausbucher.
- Checkin-Operation: Befördert die geänderte Kopie ins Archiv und löscht die Reservierung und erledigt die Geschichtseinschreibungsoperationen (Autor, Zeit, ...).

3.3 Varianten

- Verschiedene Versionen eines Software-Elements führen zu einem sequentiellen Versions-Stamm. Für die Praxis reichen solche Versions-Stämme aber nicht aus.
- Durch Varianten werden zusätzliche Anforderungen abgedeckt.
- Varianten können:
 1. Zeitlich nebeneinander liegende Ausprägungen von SW-Elementen;
 2. Verwendet werden, um parallele Entwicklungslinien darzustellen;
 3. Unterschiedliche, d.h. variante, Implementierung derselben Schnittstelle sein;
 4. Sich durch bedingte Übersetzungen unterscheiden;
 5. Auf unterschiedliche Hardware- und/oder Software-Konfigurationen zugeschnitten sein;
 6. Ab einem Abstraktionsniveau nicht mehr unterschieden werden.

Varianten – Beispiel

- Beispiel:



- Ein Produkt befindet sich in zwei Versionen bei Kunden (V1.0 und V1.2). Die Entwicklung arbeitet an Version 1.2. Der Kunde mit der Version 1.0 findet einen gravierenden Fehler. Da er auf Grund seiner Hardwarevoraussetzungen nicht auf Version 1.1 wechseln kann, wird der Fehler behoben und es entsteht Version 1.0.1.0.
- Variantennummer = release.level.branch.sequence

Identifikation – Nummerierungsschema

- Damit eine Software-Konfiguration gebildet werden kann, muss ein Software-Element eindeutig identifizierbar sein. Daher ist es notwendig, ein Nummerierungsschema einzuführen.
- Ein Nummerierungsschema muss folgendes festlegen:
 - Struktur bzw. Aufbau des Identifikators,
 - Informationen, die der Identifikator enthalten soll,
 - Verfahren, wie sich der Identifikator bei Änderung des Elements (z.B. neue Version, neue Variante) verhält.
- Als Identifikator kann beispielsweise ein hierarchisches Namensschema verwendet werden, das mit Versions- Variantenkennung kombiniert ist.
- Der Identifikator kann auch durch ein Konfigurationsmanagement-werkzeug bestimmt werden.
- Beispiel: SemOrg/Def/pfV23a.doc

3.3 Varianten

Konfigurationen erfüllen also folgende Zwecke:

- Sie legen fest, welche Entwicklungsergebnisse zu welchen Produkten oder Teilprodukten gehören.
- Sie bringen Ordnung in die Vielfalt der Entwicklungs-, Wartungs- und Pflegeergebnisse.
- Sie bilden Bezugspunkte für bestimmte Entwicklungs-, Wartungs- und Pflegeschritte.
- Sie frieren Zwischenergebnisse eines Produkts über seine gesamte Lebensdauer ein.
- Sie ermöglichen eine Analyse des Entwicklungsprozesses und des Produkts.

3.4 Konfigurations- und Änderungsmanagement

Software-Konfigurationsmanagement: unterstützt die Identifikation, Initiierung und effiziente Verwaltung von Softwarekonfigurationen, durch geeignete Methoden, Werkzeuge und Hilfsmittel, um durch kontrollierte Änderungen die Entwicklung und Pflege eines Softwareprodukts zu erleichtern und transparent zu machen.

- Ziele des Software-Konfigurationsmanagement:
 - Sicherstellen der Sichtbarkeit, Verfolgbarkeit und Kontrollierbarkeit eines Produkts und seiner Teile im Lebenszyklus.
 - Überwachung der Konfigurationen während des Lebenszyklus.
 - Sicherstellung, dass jederzeit auf vorangegangene Versionen zurückgegriffen werden.

3.4 Konfigurations- und Änderungsmanagement

Aufgaben

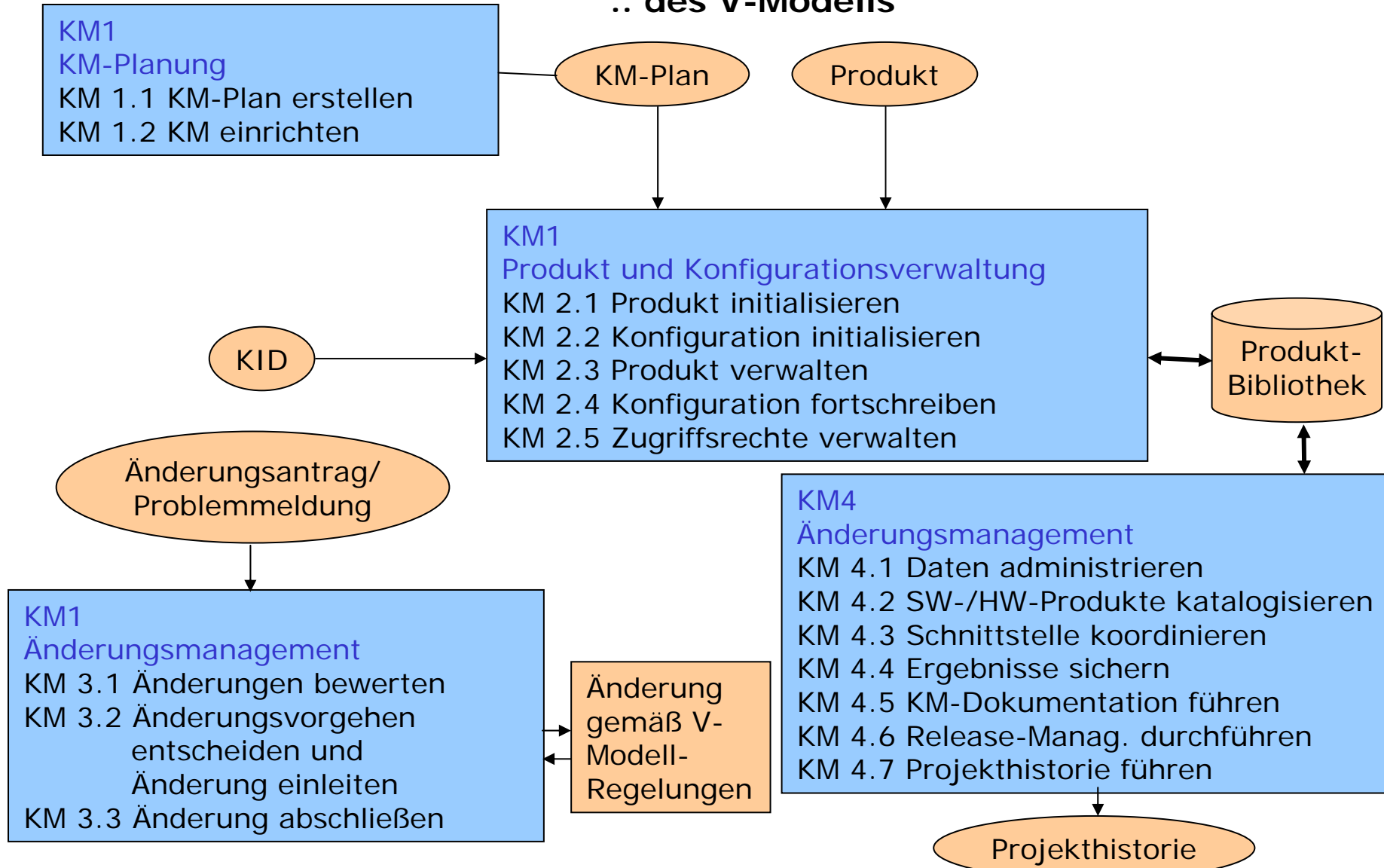
1. Planung des Konfigurationsmanagement
2. Produkt- und Konfigurationsverwaltung
3. Änderungsmanagement
 - a. Erfassung und Verwaltung eingehender Fehlermeldungen
 - b. Entscheidung über die Bearbeitung von Änderungsanträgen
 - c. Abschluss der Änderungen und Information aller Betroffenen
4. Konfigurationsmanagement-Dienste
 - a. Daten Verwalten
 - b. Soft-/Hardware-Produkte katalogisieren
 - c. Schnittstelle koordinieren
 - d. Ergebnisse sichern
 - e. KM-Dokumentation führen
 - f. Release-Management durchführen
 - g. Projekthistorie führen

Konfigurationsverwaltungswerkzeuge

- Konfigurationsverwaltungswerkzeuge müssen in der Lage sein, typische Fragen zu beantworten [Sommerville 89]:
 - Welcher Kunde hat eine spezielle Version des Produkts erhalten?
 - Welche HW- und System-SW-Konfigurationen wird für eine geplante SW-Version benötigt?
 - Wie viele Versionen des Produkts wurden wann erzeugt?
 - Welche Versionen des Produkts sind betroffen, wenn eine spezielle Komponente geändert wird?
 - Wie viele Änderungsanträge oder Problemmeldungen sind für eine spezielle Person noch unerledigt?
 - Wie viele Fehlermeldungen liegen für eine spezielle Version vor?

Das Submodell Konfigurationsmanagement ..

.. des V-Modells



- [Boehm 91]
Boehm B.W., Software Risk Management: Principles and Practices, IEEE Software
- [Itzfeld 83]
Itzfeld W., Methodische Anforderungen an Software-Kennzahlen in Angewandte Information
- [Itzfeld, Schmidt, Timm 84]
Itzfeld W., Schmidt M., Timm M.. Spezifikation von Verfahren zur Validierung von SW-Qualitätsmaßnahmen in Angewandte Informatik
- [Sommerville 89]
Sommerville I., Software-Engineering –3rd ed. Addison-Wesley
- [Thayer 90]
Thayer R.H., Tutorial Software Engineering Project Management, IEEE Society Press
- [Wallmüller 90]
Wallmüller E., Software-Qualitätssicherung in der Praxis, Carl Hanser Verlag
- [Zuse 85]
Zuse H., Messtheoretische Analyse von statischen Softwarekomplexitätsmaßen, Tu Berlin