Fachhochschule Frankfurt am Main
Fachbereich 2: Informatik
SS 2008

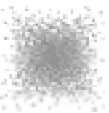
IT Projekt Management

Vorlesung 9:

Software Life Cycle Modelle und Modellierung

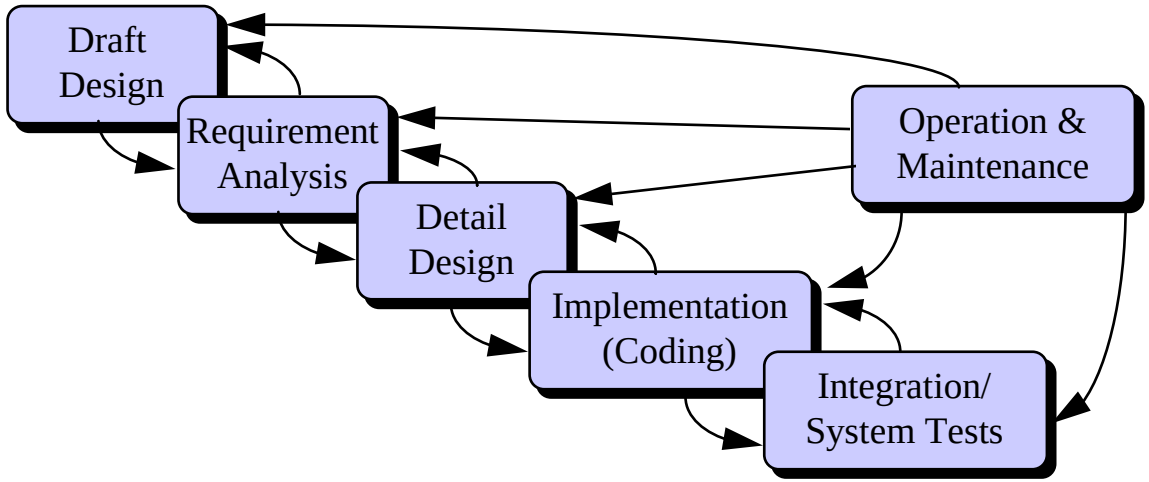
Dr. Erwin Hoffmann

E-Mail: it-pm@fehcom.de



Lebenszyklus-Modelle von Software

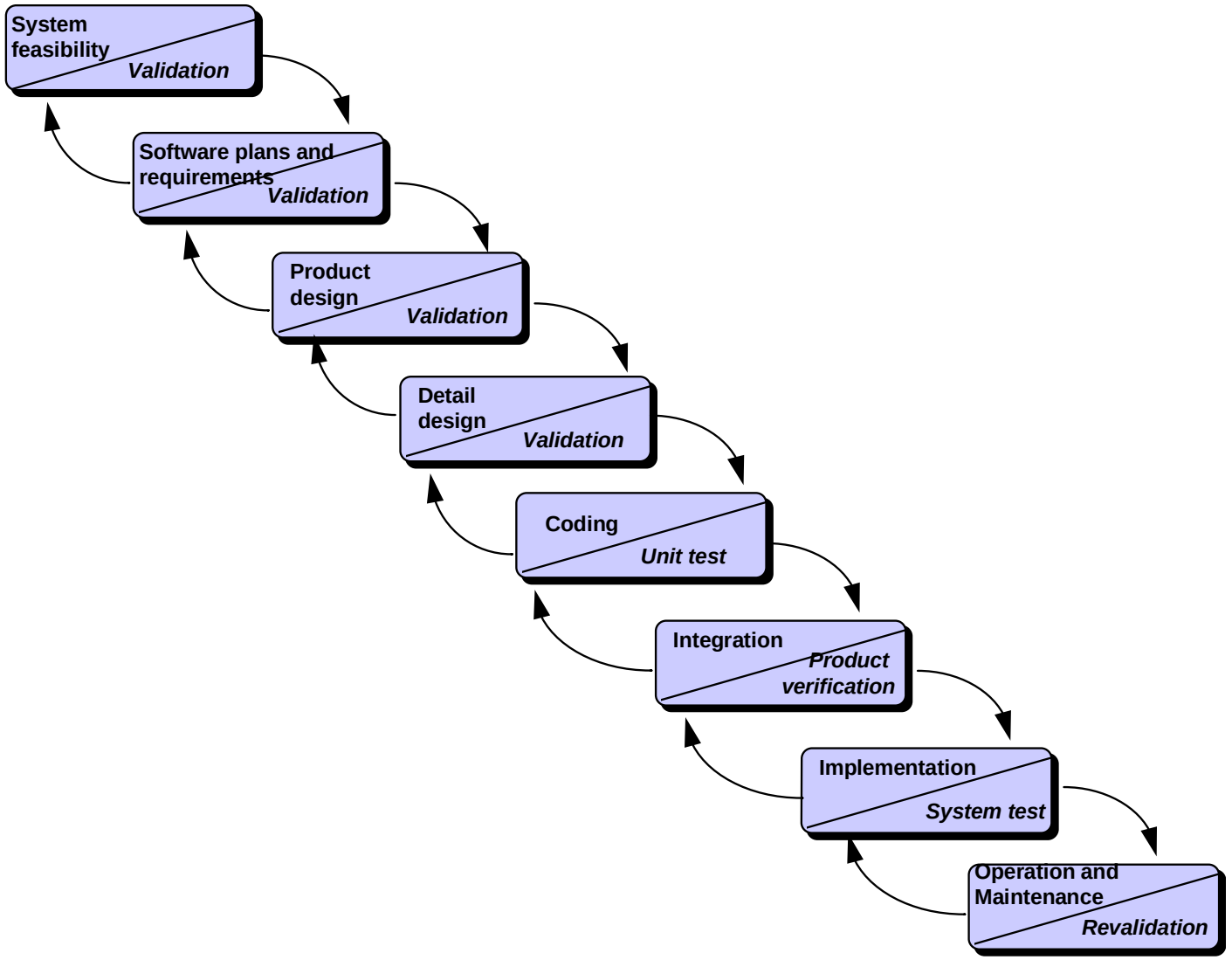
- Es hat sich als allgemeines Wissen etabliert, dass Computer-Software eine eigenen „Lebenszyklus“ aufweist.
  - Der Lebenszyklus startet mit der ersten Idee zu der betreffenden Software und und mit der Inbetriebnahme noch lange nicht abgeschlossen.
- Mit der Realisierung der ersten Software-Projekte wurden unterschiedlich Lebenszyklus-Modelle diskutiert und realisiert.



Das Wasserfall Modell

- Das Wasserfall Modell ist wahrscheinlich der früheste Ansatz zur Qualitätsverbesserung der Software-Entwicklung und entstand 1956 im Rahmen des Projekts 'Semi-Automated Ground Environment (SAGE)' zunächst als "Phasen-Modell".
  - Die endgültige Ausprägung erhielt das Wasserfall Modell in 1970.
- Die wesentlichen Eigenschaften des Wasserfall Modells sind:
  - Erkennung von *Rückkopplungs-Kreisläufen* zwischen den einzelnen Phasen, sodass die Auswirkungen von Fehler und Probleme in einer Phase auf die benachbarte Phase beschränkt bleiben und sich nicht auf das endgültige Produkt durchschlagen
  - Das Erstellen von *Prototypen* im Lebenszyklus als sogenannter "build-twice" Schritt.
- Hieraus resultiert, dass das Wasserfall Modell zwingend den Einsatz eines Verifikation/Validierungs- Prozesses vorschreibt, um zur nächsten Phase zu gelangen.
  - Hieraus folgt, dass prinzipiell im Software-Entwicklungsprozess "offene Probleme" nicht akzeptiert werden, was insbesondre die Entwicklung nicht-vorhersagbarer Produkte, wie z.B. interaktive Systeme, die die Mitwirkung von Menschen bedingt, nicht effizient abgewickelt werden können.
  - Zudem war das Konzept der "Use Cases" noch nicht entwickelt.

Das Wasserfall Modell (2)



Das V-Modell

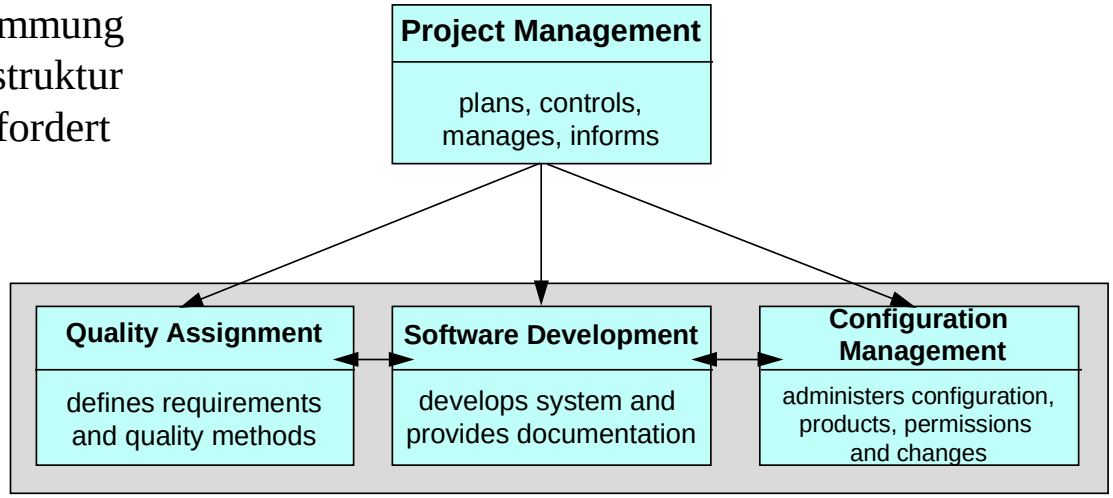
- Das V-Modell stellt einen prozess-orientierten Ansatz zur Software-Entwicklung als Bestandteil des IT Projekt Managements dar.
  - Es wurde Barry Boehm vorgeschlagen und anschliessend in Deutschland im Rahmen öffentlicher und Militärprojekten weiterentwickelt.
  - Das zunächst 1986 publizierte V-Modell ist durch das aktuelle V-Model XT abgelöst.
- Die Anwendung des V-Modell für die Software-Entwicklung ist ein wichtiger Baustein zur Realisierung der ISO 9000 Konformität:
  - Das V-Modell verlangt in Übereinstimmung mit ISO 9000 eine qualifizierte Infrastruktur für die Software-Entwicklung und erfordert Verantwortlichkeiten für das

    Projekt-Management,

    Qualitäts-Management,

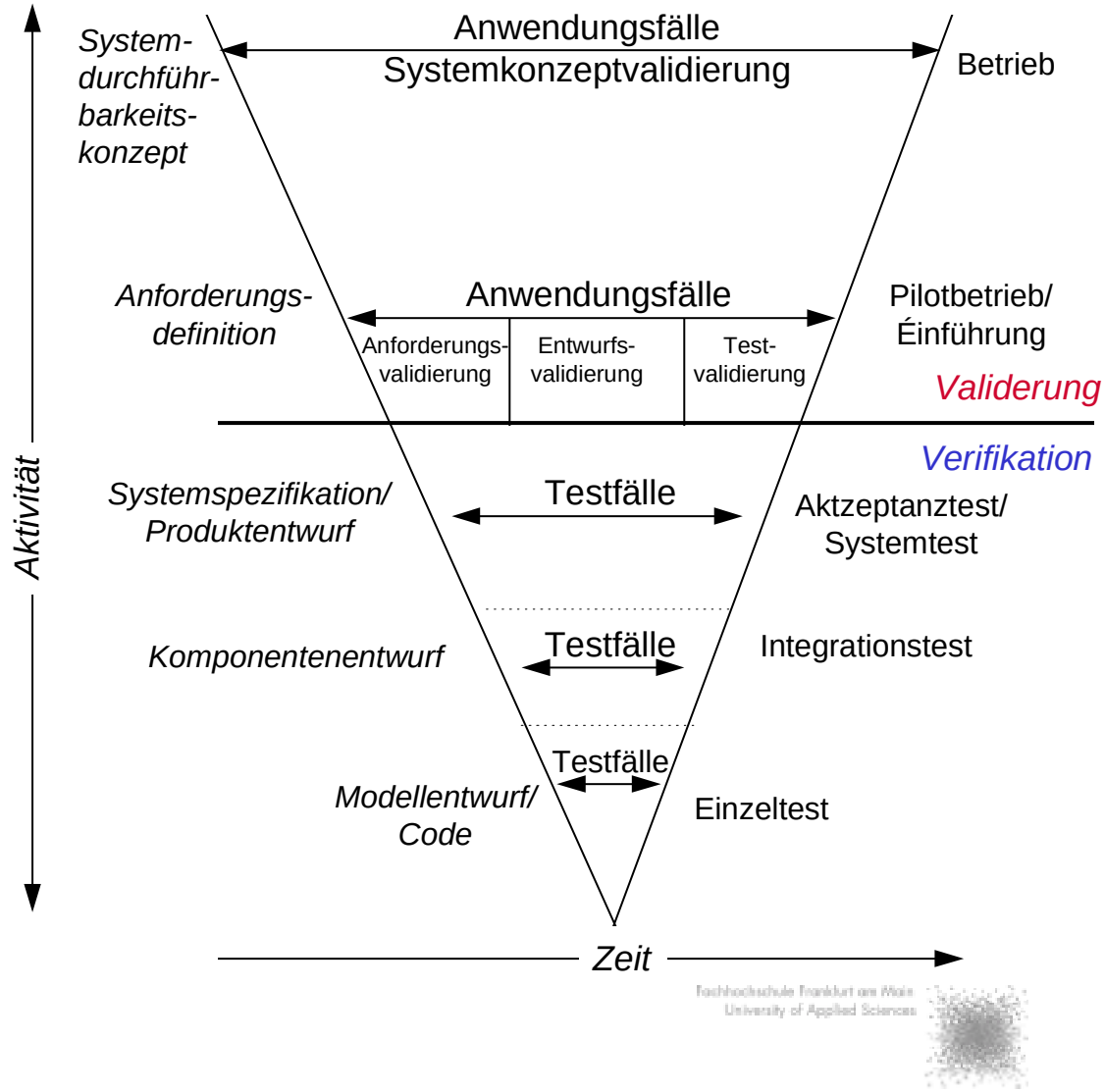
    Konfigurations-Management,

    und die Software-Entwicklung.



Das V-Modell (2)

- Abweichend von klassischen Projekt-Management Methoden, werden die Phasen nicht auf einer Zeitlinie definiert, sondern *Prozedur-abhängig*:
  - Das V-Modell legt eine Reihe von Ereignissen und Aktivitäten im Hinblick auf die Planung, das Design und die Entwicklung, sowie die Tests und den Roll-Out Prozess fest.
- Das V-Modell ist *Dokumenten-getrieben:*
  - Im Rahmend der Projekt-Durchführung werden die Aktivitäten auf Management-Ebene herunter-gebrochen, mit der Massgabe, dass alle Aktivitäten und deren Resultate dokumentiert sein müssen.

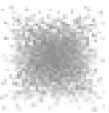


Das V-Modell (3)

- Die Nachteile des V-Modells lassen sich wie folgt zusammenfassen:
  - Das V-Modell 'formalisiert' das Projekt-Management und die Software-Entwicklung durch die Reduzierung auf die entsprechenden Dokumente, deren Erstellung umfangreiche administrative Prozesse beinhaltet
  - Die Software-Entwicklung muss strikt vorbestimmt ablaufen; was teilweise unrealistisch ist und jede Änderung muss über einen Approvement-Prozess eingestellt werden.
  - Zwischen den Team-Mitgliedern wird wenig Interaktion erwartet bzw. verlangt, was der Kreativität des Teams nicht unbedingt zuträglich ist.

Das V-Modell XT

- Sowohl das V-Modell 1997 und das neue V-Modell XT wird in Deutschland vom BMI-KBSt (Koordinierungs- und Beratungsstelle der Bundesregierung) gewartete und öffentlich verfügbar gemacht (als Dokumente und SW-Downloads).
- Das neue V-Modell XT beinhaltet eine neue Ansätze, die der allgemeinen Entwicklung in der IT Welt entsprechen:
- Auf der untersten Ebene wird als Merkmal das *Project Subject* eingeführt:
  - Dies kann eine Hardware-Lösung sein,
  - oder eine Software-Lösung,
  - ein komplexes System, das sich aus beiden Komponenten ergibt,
  - eine 'embedded' Lösung,
  - oder ggf. eine System-Integration;
  - jedoch sind Dienstleistungs-Lösungen hiervon nicht beinhaltet.



Das V-Modell XT (2)

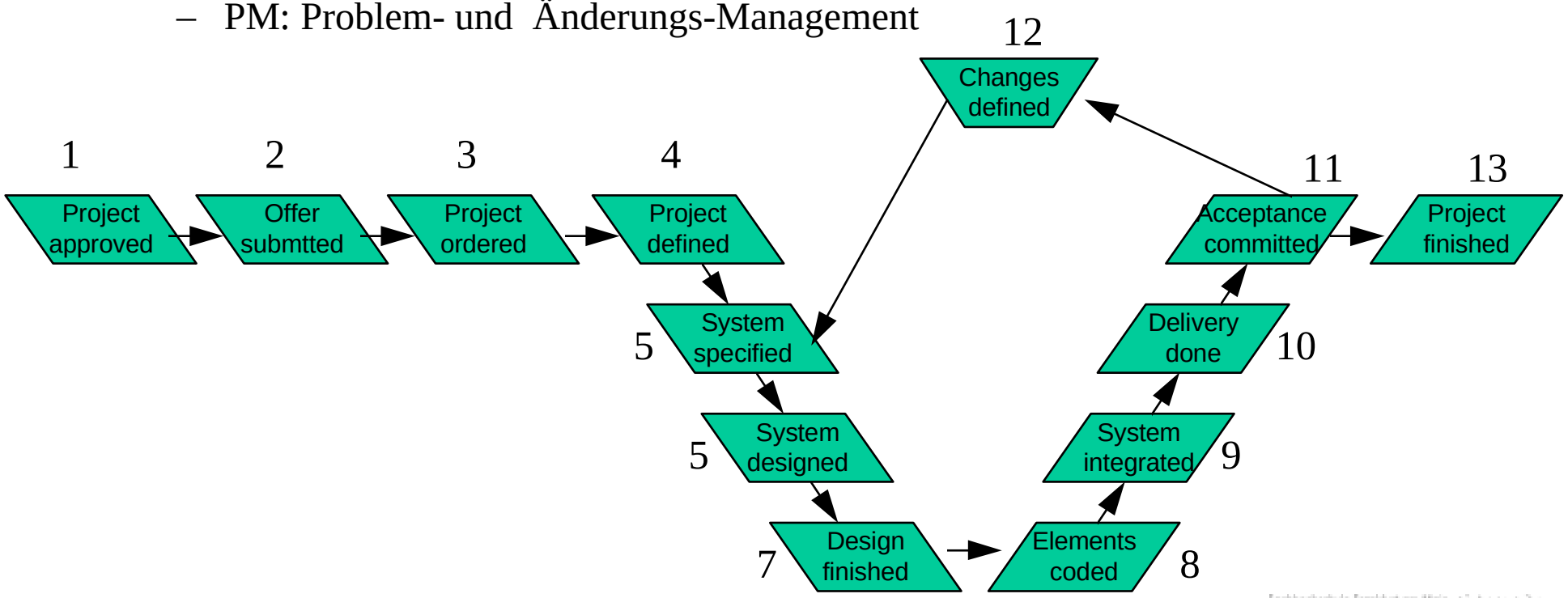
- Die Definition einer *Project Rôle*, die mitteilt, wer das V-Model XT verantwortlich einsetzt:
  - der Kunde (Auftraggeber), der mehrere Beteiligte (Auftragnehmer) managt,
  - der Lieferant (Auftragnehmer), der das Verfahren für sich selbst und weitere Unter-Lieferanten einsetzt.
  - gemeinsam, sowohl Kunde und die Lieferanten setzen das V-Model XT ein.
- Das V-Modell XT entspricht damit dem typischen Fall eines öffentlichen Auftraggebers (GO) bei dem ein Regierungs-Organisation als Kunde fungiert und der Lieferant ein Privat-Unternehmen ist, das nach einem Ausschreibungsverfahren ausgewählt wurde.

  ⇨ *Das V-Modell XT kann als inkrementelles, anpassbares Prozess-Modellverfahren betrachtet werden.*

  ⇨ *Im Unterschied zum ursprünglichen V-Model, werden nun keine Dokumente ausgetauscht, sondern der Projektfortschritt in einem CASE-Tools dokumentiert.*

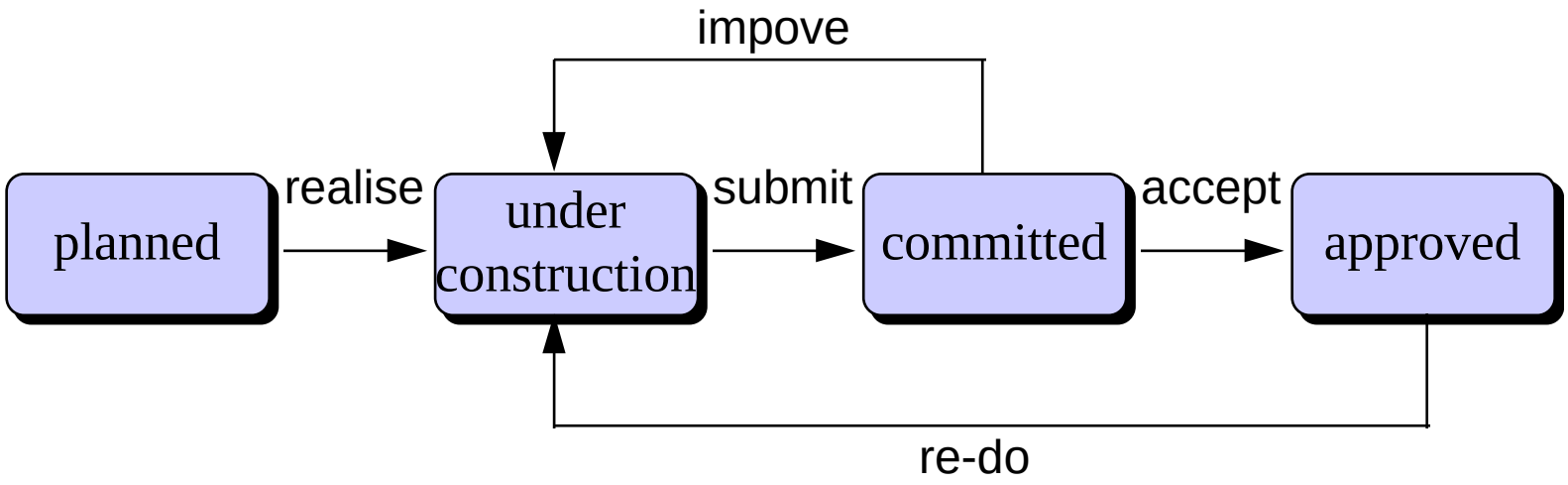
Das V-Modell XT (3)

- Das V-Modell XT beinhaltet folgende Haupt-Module:
  - PM: Projekt-Management
  - QA: Qualitäts-Sicherung
  - CM: Konfigurations-Management
  - PM: Problem- und Änderungs-Management



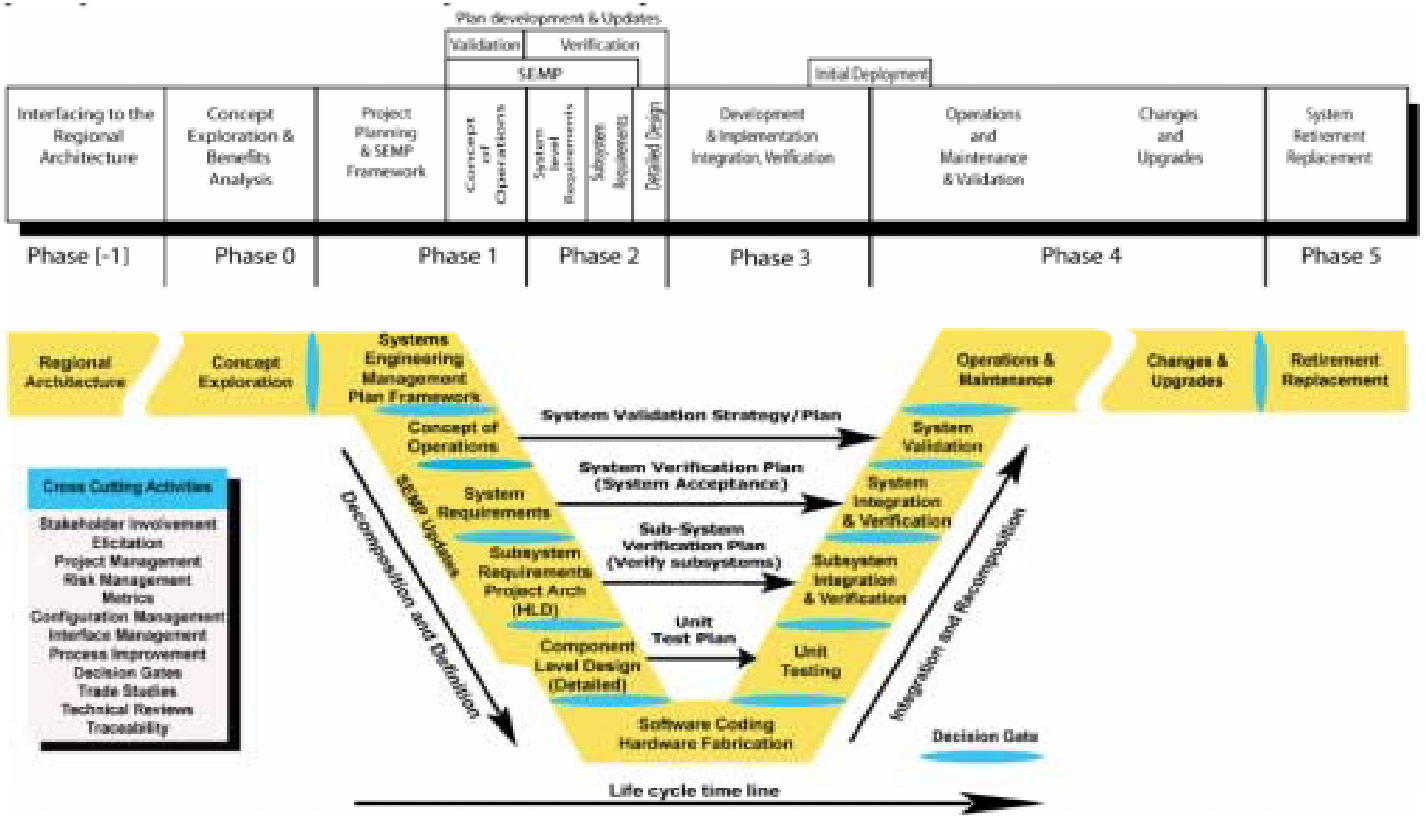
Das V-Modell XT (4)

- Jede Änderung am *Produkt* (oder einem Teil-Produkt) wird *Aktivität* genannt und muss die folgenden Phasen durchlaufen:



Das V-Model XT (5)

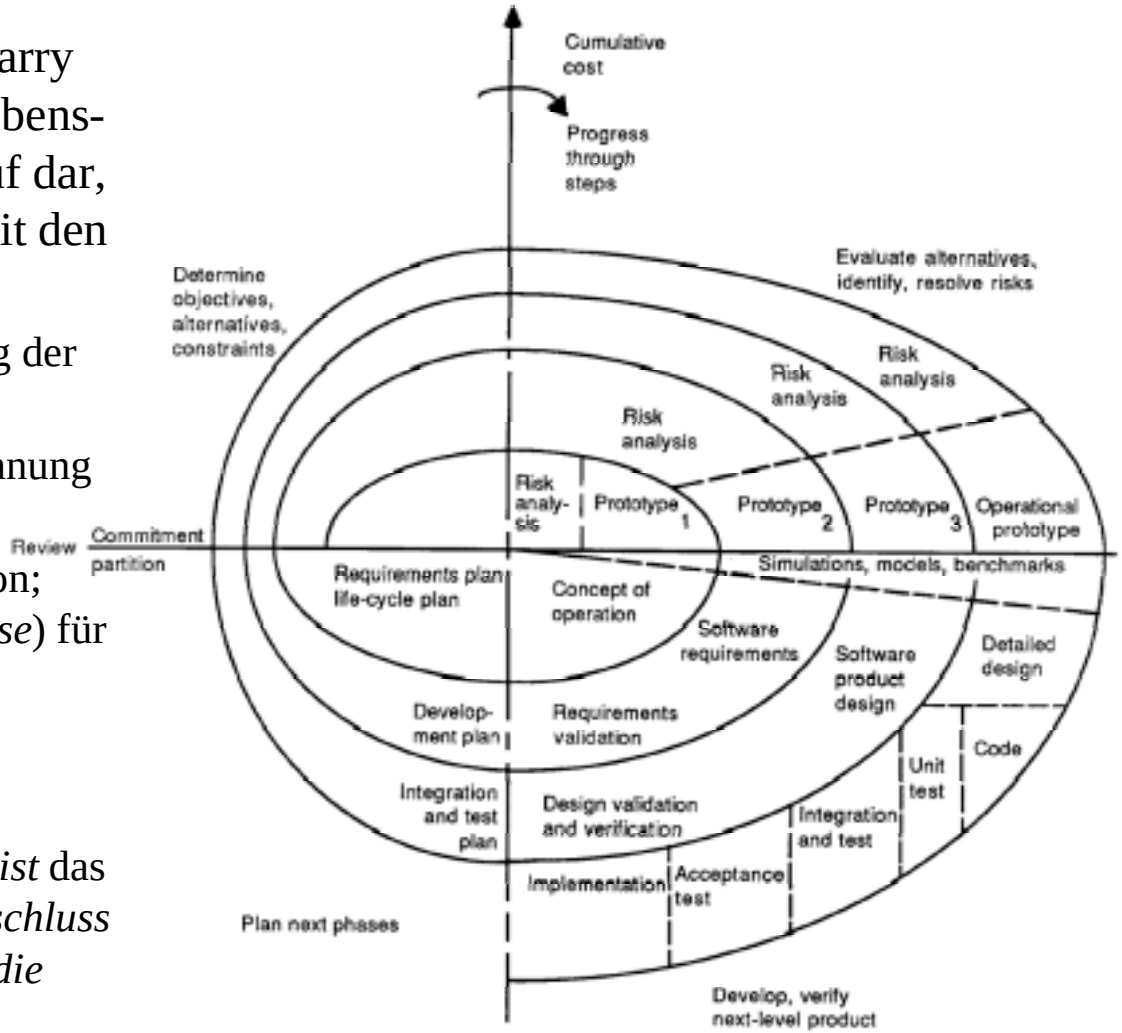
- Obwohl das V-Modell XT vor allem in Deutschland angewendet und genutzt wird, hat es dennoch internationale Bedeutung.
  - Das US amerikanische ITS (Intelligent Transport System) wurde entsprechend entwickelt und aufgebaut (vgl. "System Engineering Guidebook For ITS"):



Das Spiral-Model

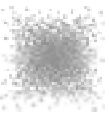
- Das originale Spiral-Model wurde von Barry W. Boehm 1988 entwickelt, stellt den Lebenszyklus eines Produktes in einem Kreislauf dar, bzw. projeziert diesen auf eine *Spirale* mit den folgenden vier Quadranten (Q1 - Q4):
  - Q1: Formulierung der Ziele, Feststellung der Alternativen und Randbedingungen
  - Q2: Evaluierung der Alternativen, Erkennung von Risken und ihrem Entgegentreten
  - Q3: Produktentwicklung und -verifikation; Übergang in den nächste Kreislauf (*Phase*) für das Produkt
  - Q4: Planung der nächsten Phase

➡ *Im Gegensatz zu den anderen Modellen ist* das *Spiral-Model <u>risiko-orientiert;</u> nach Abschluss eines Kreislaufs werden die Risiken für die nächste Runde bewertet.*



Das Spiral-Model

- Die Risikobewertung erfolgt durch das Stellen entsprechender Schlüsselfragen statt, die anschliessend beantwortet und dokumentiert werden müssen:
  - *Welche Ziele liegen vor ?*
  - *Welche Rahmenbedingungen sind zu berücksichtigen ?*
  - *Welche Alternativen existieren ?*
  - *Welche Risken sind zu berücksichtigen?*
  - *Wie kann diesen Risiken begegnet werden ?*
  - *Was wäre das Resultat der Risiko-Lösung ?*
  - *Welche Pläne gibt es für die nächste Phase?*
  - *Welche Verpflichtungen gibt es ?*
- In diesem Kontext wird ein Umlauf ("Runde") als Phase betrachtet:
  - Runde 0: *Machbarkeitsstudien*
  - Runde 1: *Betriebskonzept*
  - Runde 2: *Spezifikation der Top-Level Anforderungen für das Produkt*



GQM: Goal/Question/Metric

- GQM ist das Akronym für

  Goal/Question/Metric

  und wurde von Victory Basili an der University of Maryland in 1983 entwickelt und von Dieter Rombach 1988 weitergeführt.
- GQM ist eine Weiterentwicklung und Zusammenführung der Methoden
  - ***QFD*** Quality Function Deployment (entwickelt von Yoji Akao 1966 an der Tamagawa University in Tokyo)
  - ***SQM*** Software Quality Metrics (entwickelt 1980 von Marine für Metrics Incorporated)

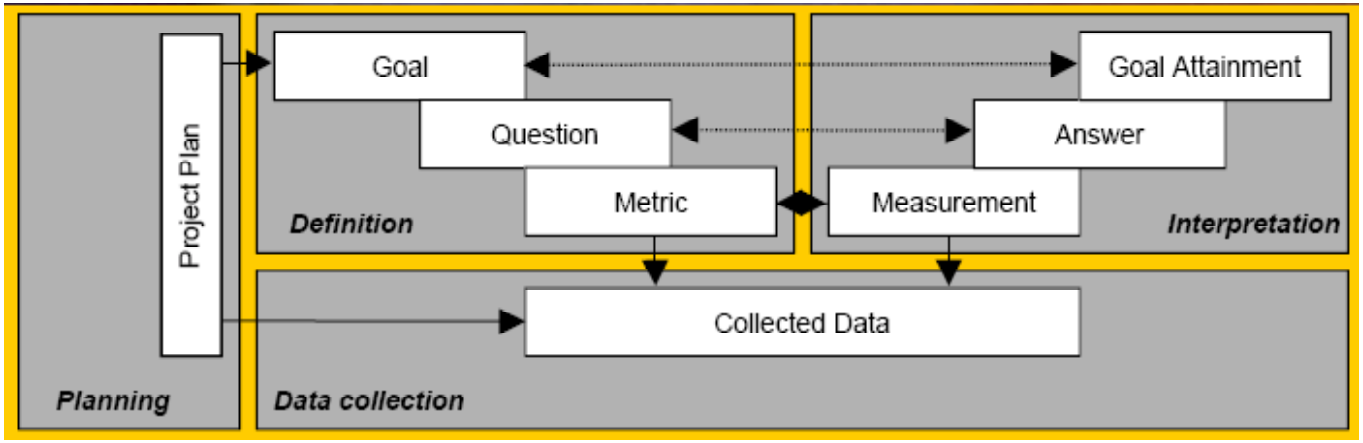
  und kann im Rahmen des
  - Qualitäts-Managements und
  - Projekt-Managements eingesetzt werden.

  ⇨ *GQM ein Paradigma-basiertes Verfahren; das Resultat jeder Messung wird ausschliesslich in einem bestimmten Kontext betrachte, der im GQM Plan beschrieben ist.*

GQM: Goal/Question/Metric (2)

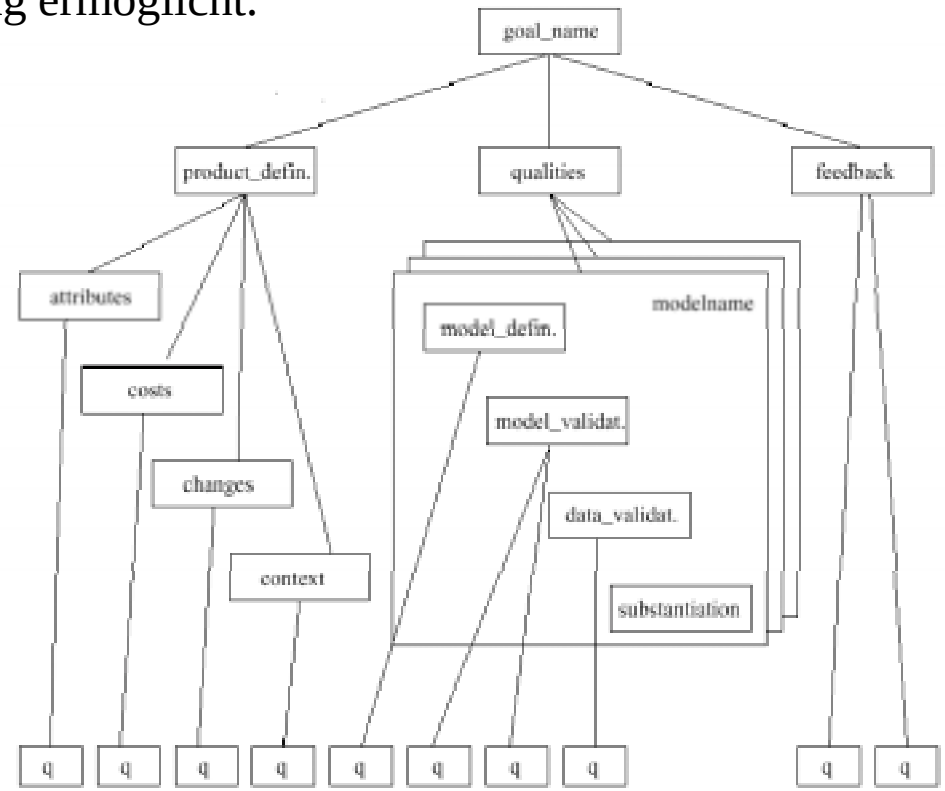
- Im Rahmen des GQM Ansatzes werden die folgenden Phasen berücksichtigt:
  - Die Entwicklung des GQM Plans
  - Die Definition der Fragen und der Metriken für die Messungen
  - Die Bereitstellung der Messdaten
  - Die Interpretation der Messungen im bezüglichen Kontext

    Im Gegensatz zu anderen Modelen, berücksichtigt GQM die Herkunft der Messdaten und kann sie gegen den Zugriff Dritter schützen.



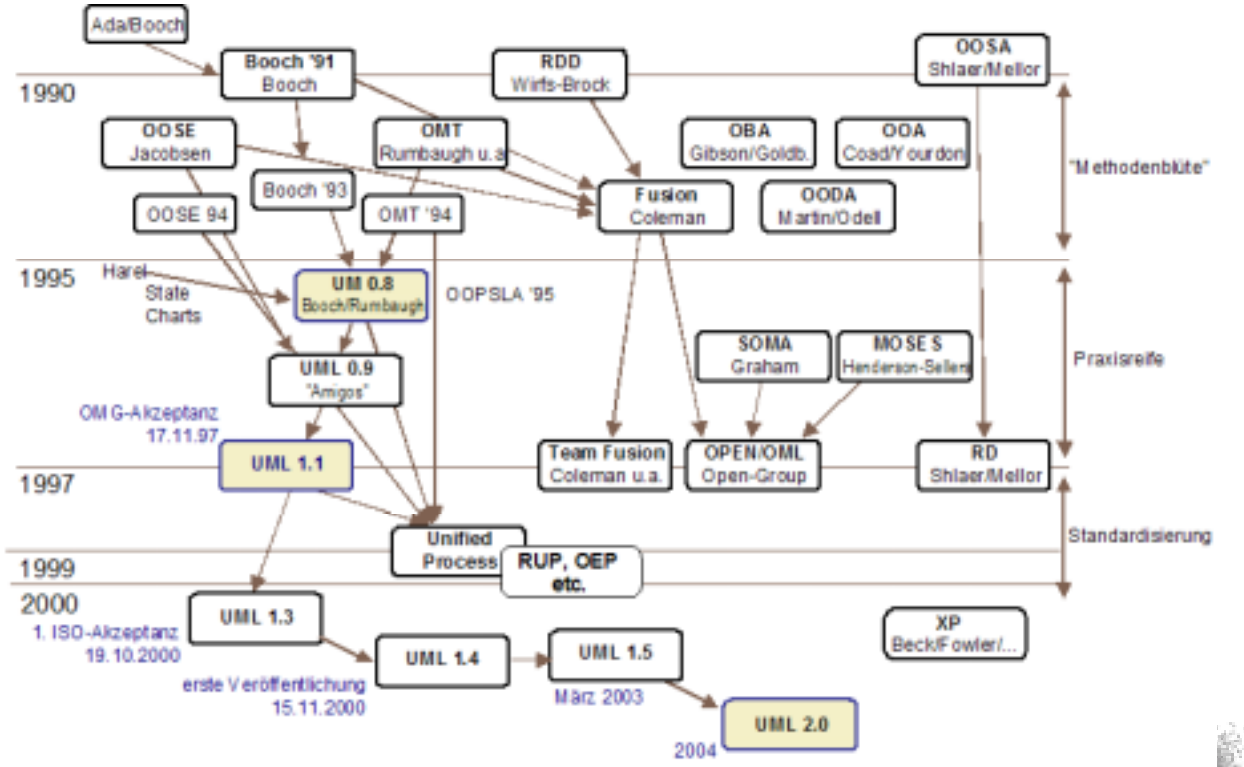
GQM: Goal/Question/Metric (3)

- Herausragend für die Software Entwicklung ist Bereitstellung einer *Prozedursprache*, mit der die Abhängigkeiten im Rahmen des GQM Modells beschrieben werden können.
  - Für das *Eclipse* IDE steht ein plug-in 'FOCUS' bereit, das eine umfangreiche Modellierung ermöglicht.



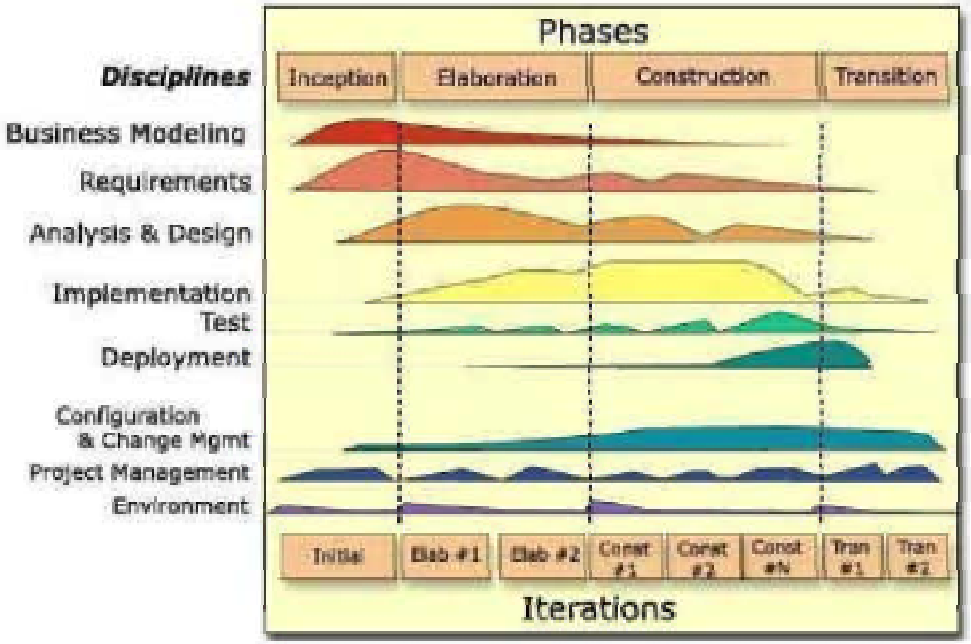
Das RUP Modell

- Das *Rational Unified Prozess* (*RUP*) Modell wurde von der Firma *Rational* in 1996 entwickelt (wie auch Rational Rose, Clear Case und andere Produkte) und wird von IBM weitergeführt.
  - RUP ist ein *prozedurales Modell* für die Software-Entwicklung und liegt z.Z. in der Version 9 vor.



Das RUP Modell (2)

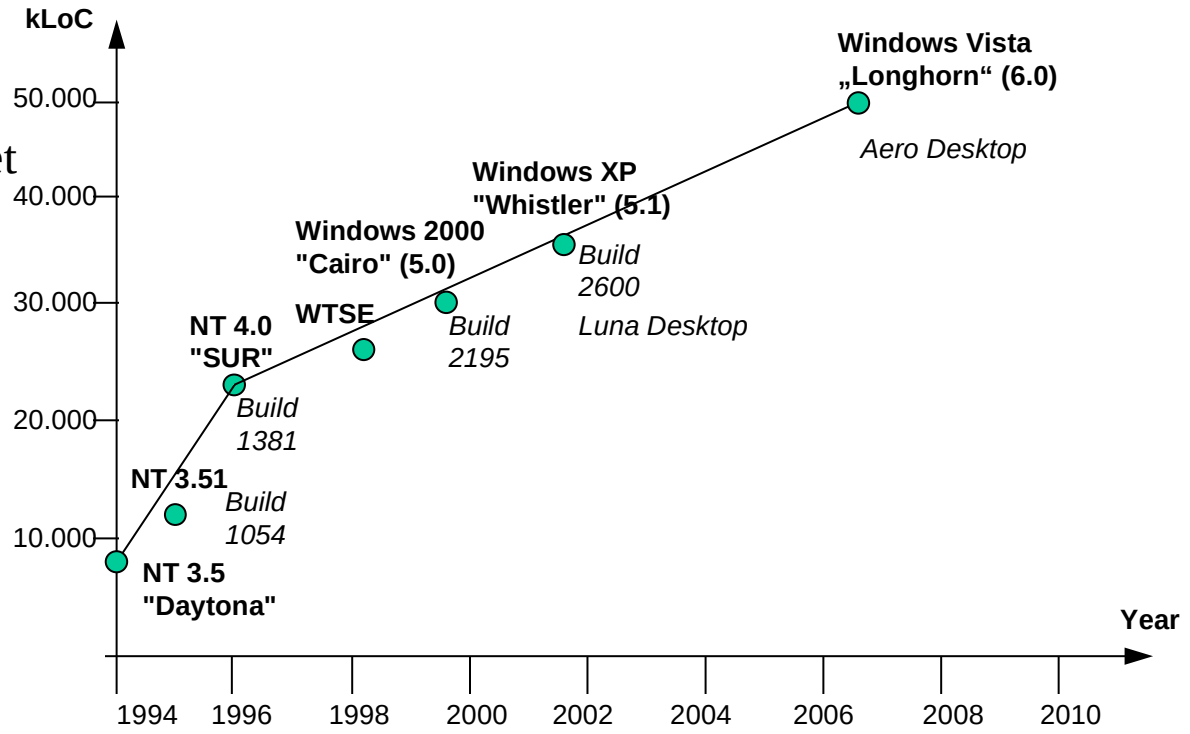
- Wie auch Rational Rose benutzt RUP einen objekt-orientierten (OO) Ansatz für Beschreibung der Software-Entwicklung und unterteilt diese in zwei voneinander unabhängige Betrachtungsweisen:
  - Disziplinen:
    - *Geschäftsmodellierung*
    - *Anforderungsanalyse*
    - *Analyse & Design*
    - *Implementierung*
    - *Tests*
    - *Auslieferung -- und zusätzlich*
    - *Konfiguration- & Änderungs-Management*
    - *Projekt-Management*
    - *Beschreibung des Umfelds (Environment)*



  - Phasen:
    - *Inception* (High-Level Design zur Bestimmung der *Lifecycle Objectives LCO*)
    - *Elaboration* (Detail-Design liefert die *Lifecycle Architecture LCA*)
    - *Construction* (Programmierung führt die *Operational Capabilities IOC* ein)
    - *Transition* (Phasen-Übergange verantwortlich für das *Product Release RP*)

Software Metriken

- Software Metriken stellen den Versuch dar, den Umfang des erstellten Programm-Codes zu bestimmen.
- Der Messwert ist typischerweise ein ein-dimensionaler Wert, der nach einer bestimmten Methode berechnet wird. Folgende Methoden sind üblich:
  - Anzahl der Programmzeilen (Number of Lines of Code LoC) ausgedrückt in kLoC oder MLoC.
  - Anzahl der Programm-Wörter, -Zeichen oder -Dateien.
  - Anzahl der Funktionsaufrufe bzw. Unterroutinen.
  - Anzahl der Klassen (bei OO Sprachen).



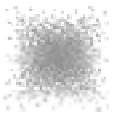
Software Metriken - Gesetzmässigkeiten

- Die vorgenommene Messmethode ist von der Programmiersprache abhängig.
  - Zum Vergleich einer unterschiedlichen Code-Basis muss immer dieselbe Methode herangezogen werden.
- Hintergrund ist die Korrelierung des gefunden Masses mit den 'Gewohnheiten' der Software-Entwicklung:
  - *Gesetz der kontinuierlichen Änderung*: Wird ein Use Case geändert zieht dies Anpassungen in allen relevanten Software-Modulen nach sich.
  - *Gesetz der wachsenden Entropie*: Mit wachsender Entwicklungsdauer, wird der Code immer weniger wartbar.
  - *Gesetz des statistisch langsam anwachsenden Programmcodes*: Ein typisches Software Projekt produziert einen kontinuierlich anwachsende Codebasis.
  - Das *'Pareto' Gesetz* sagt aus, dass 20% des Codes für 80% der Resultate wesentlich ist.
  - Das *Gesetz der Wiederverwertbarkeit* fordert:

    "Kein Problem nochmals zu lösen, für das es bereits eine Lösung gibt.
    Zunächst ist das Software-Repository auf eine bereits existierende Lösungen hin zu überprüfen."
  - Das *Parkinson'sche Gesetz* beinhaltet:

    "Es wird immer soviel gearbeitet, wie Zeit zur Verfügung steht."

    "Es gibt immer den richtigen Zeitpunkt für ein Projekt; aber niemals den, dieses nochmal zu tun." "Der Einsatz zusätzlicher Mitarbeiter in einer späten Projektphase verzögert das Projekt."



Software Metriken - bei Produkten

- Die Anzahl des Codes korreliert nicht mit der Qualität eines Produktes:

Mail Transfer Agent	Lines	Words	Characters	Files
qmail-1.03	16617	44780	395243	279
sendmail-8.9.1	55059	179376	1229121	54
zmailer-2.2e10	57595	205524	1423624	227
smail-3.2	62331	246140	1701112	151
exim-2.02	70102	283295	2172786	128

Software Modellierung und CASE Werkzeuge

- Beim Software-Design sind die folgenden Dokumente von Bedeutung:
  - *High Level Design Dokument* (***HLD***), das die Produktfunktionalität umfassen aber grob beschreibt und die
  - ***Low Level Design Dokumente*** (*LLD*), die für jede Komponente eine detaillierte Beschreibung beinhalten und daher auch *Detailed Design Dokumente* genannt werden.
- Bei Software-Projekten in Deutschland wird das HLD von den folgenden Input-Dokumenten bestimmt:
  - *Pflichtenheft* - der abgestimmte Spezifikationsumfang zwischen Kunde und Lieferant/Entwickler
  - *Lastenheft* - die angeforderten Spezifikation und Produktrandbedingungen des Kunden.
- Die vorgenommene Beschreibung in diesen Dokumenten folgt in der Regel einer formalen Modellierungs-Sprache und wird in Diagrammform dargestellt.
  - Zur Erstellung dieser Diagramme und zur Sicherstellung der Architektur-Konsistenz werden ***CASE*** (*Computer Aided Software Engineering*) Werkzeuge im Design-Prozess eingesetzt.

Software Modellierung und CASE Werkzeuge (2)

- Heutzutage finden besonders drei unterschiedliche Modellierungs-werkzeuge Einsatz:
  - UML - *Unified Modelling Language*
  - ERM - *Entity Relationship Model*
  - SA/SD - *Structured Analysis/Structured Design*
- Die CASE Werkzeug erlauben während des Modellierns die Bestimmung der Abhängigkeiten zwischen den Software-Komponenten.
- Daher unterstützen die CASE Werkzeuge das Ermitteln der geeigneten PSP bei Software-Projekten sehr effizient.