Karlsruher Institut für Technologie

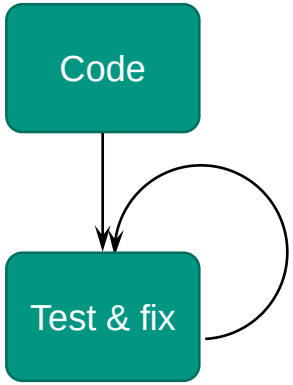
Kapitel 8.1 – Prozessmodelle

SWT I – Sommersemester 2021
Walter F. Tichy, Christopher Gerking, Tobias Hey

Prozessmodelle

- Programmieren durch Probieren
- Wasserfallmodell
- V-Modell
- Prototypenmodell
- Iterative Modelle
- Synchronisiere und Stabilisiere
- Agile Methoden
  - Extreme Programming
  - Scrum

Programmieren durch Probieren



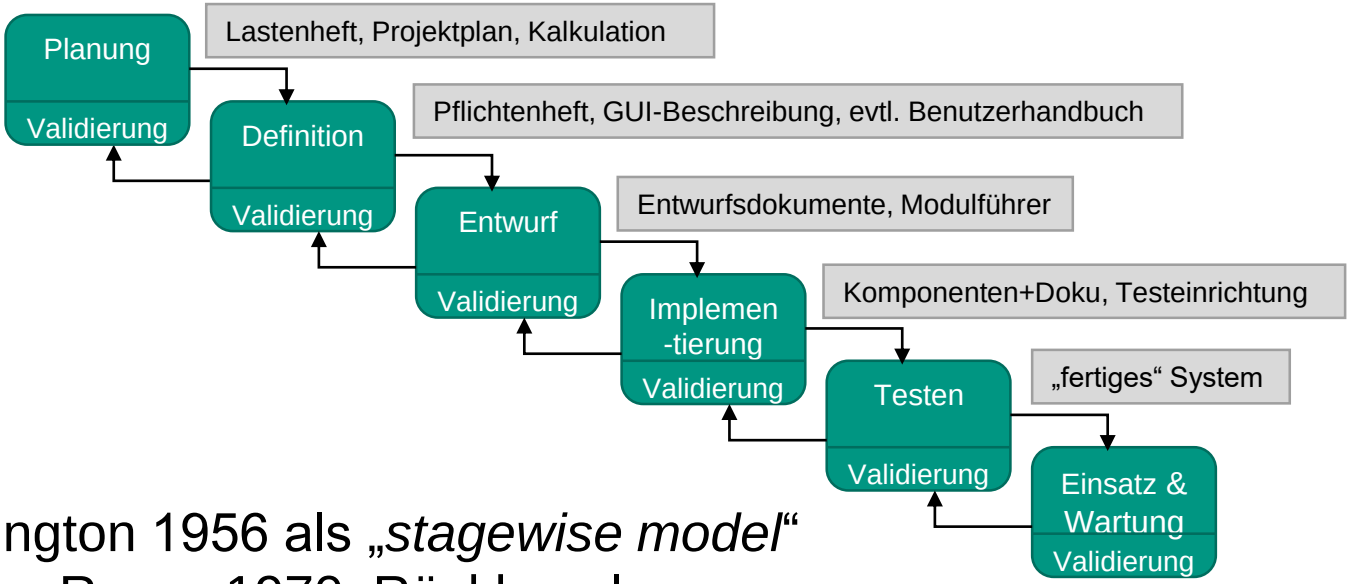
- Auch „code & fix“ oder „trial & error“
- Vorgehen
  - Vorläufiges Programm erstellen
  - Anforderung, Entwurf, Testen, Wartung überlegen
  - Programm entsprechend „verbessern“
- Eigenschaften
  - Schnell (?), Code ohne „nutzlosen“ Zusatzaufwand
  - Erzeugt schlecht strukturierten Code wegen unsystematischer Verbesserungen und fehlender Entwurfsphase

Programmieren durch Probieren

- Probleme
  - Mangelhafte Aufgabenerfüllung wegen Fehlens der Anforderungsanalyse
  - Wartung/Pflege kostspielig, da Programm nicht darauf vorbereitet
  - Dokumentation nicht vorhanden
  - Für Teamarbeit vollständig ungeeignet, da keine Aufgabenaufteilung vorgesehen.

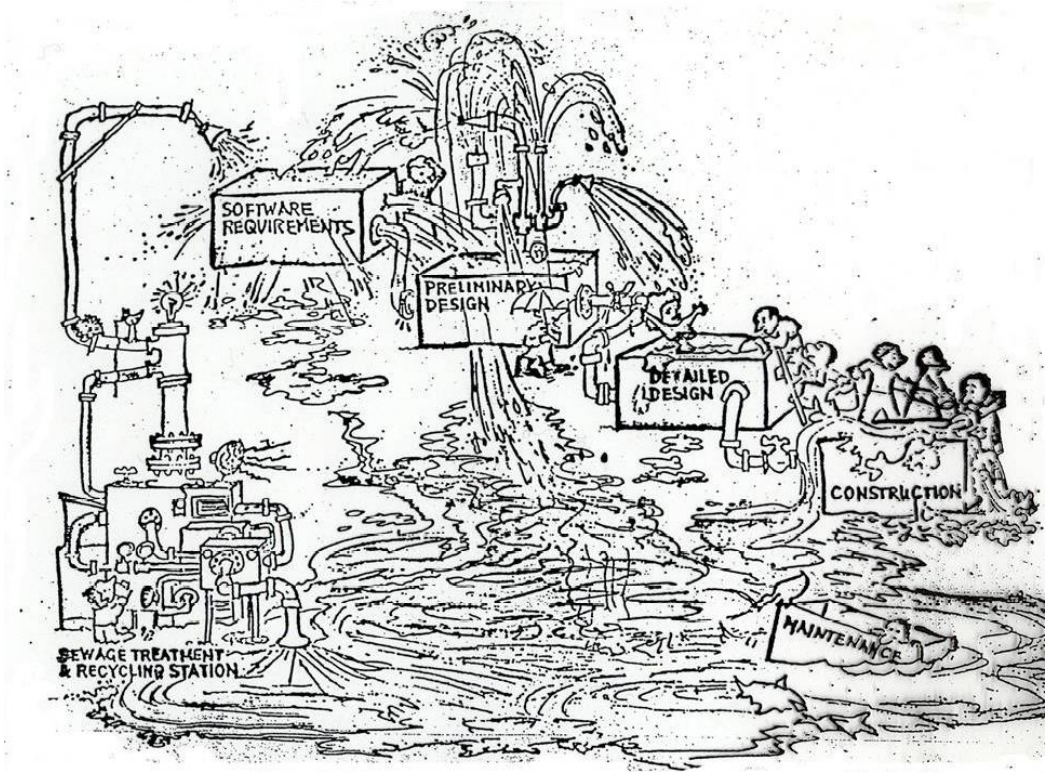
Wasserfallmodell

- Auch „Phasenmodell“



- Erstmals: Benington 1956 als „*stagewise model*“
- Erweiterung von Royce 1970: Rückkopplung

Wasserfallmodell



Wasserfallmodell

- Vorgehen
  - Jede Aktivität
  - in der angegebenen Reihenfolge
  - vollständig durchführen

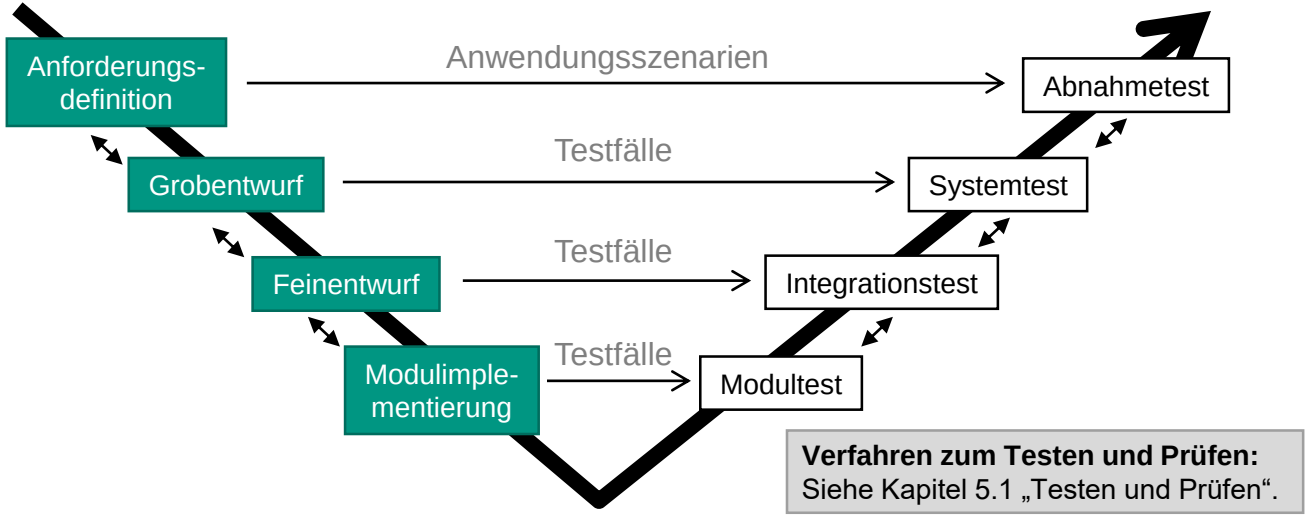
  } streng sequenzielles Vorgehen
- Am Ende jeder Aktivität steht ein fertiges Dokument
  → „dokumentgetriebenes“ Modell
- Einfach, verständlich
- Benutzerbeteiligung nur in der Definitionsphase vorgesehen

Wasserfallmodell

- Probleme
  - Keine phasenübergreifende Rückkopplung vorgesehen
    → Fehlersuche und Korrektur problematisch
  - Parallelisierungspotential möglicherweise nicht richtig ausgeschöpft
    → Markteinführung verzögert sich unnötig
  - Zwingt zur genauen Spezifikation auch schlecht verstandener Benutzerschnittstellen und Funktionen
    → Entwurf, Implementierung und Testen von später nutzlosem Code
- Daher: Wasserfallmodell ist eher ein pädagogisches Modell, bei dem die einzelnen Aktivitäten klar getrennt sind und daher in „Reinform“ studiert und erlernt werden können. Insbesondere zeigt es auf, dass SW-Entwicklung wesentlich mehr ist als nur Codieren.
- Viele Praktiker benutzen leider noch Code&Fix.

„V-Modell 97“ – das „handelsübliche“

- V wie Vorgehensmodell
- Jede Aktivität hat einen eigenen Prüfungsschritt



V-Modell XT® (Vorgehensmodell)

- Entwicklungsstandard für IT-Systeme der öffentlichen Hand in Deutschland [BMI-KBSt] (535 Seiten)
- Aktivitäten, Produkte und Verantwortlichkeiten werden festgelegt, jedoch keine Reihenfolge/Phasengrenzen
  - Aktivität – Tätigkeit, die im Bezug auf ihr Ergebnis und ihre Durchführung genau beschrieben werden kann
  - Produkt – Ergebnis einer Aktivität

  → Traditionelles Wasserfallmodell ist eine mögliche Ausprägung!
- Projekt wird aus vielen möglichen Perspektiven (Rollen) betrachtet
- Weiterentwicklung des V-Modells 97

V-Modell XT: Rollen (siehe [BMI-KBSt], Teil 4.2)

- Definierte Rollen (30 Stück): Akquisiteur, Anwender, etc.
- Beispiel „Rolle SW-Architekt“:
  - **Beschreibung**: Der SW-Architekt ist der Verantwortliche für Entwurf und Entwicklung aller SW-Einheiten des Systems.
  - **Aufgaben und Befugnisse**
    - Entwurf der SW-Architektur
    - Umsetzung der Anforderungen an die Software-Einheiten
    - Verantwortlichkeit für Implementierungs-, Integrations- und Prüfkonzept SW
    - …
  - **Fähigkeitsprofil**
    - Kennt Anwendung, Rahmenbedingungen und Einsatzgebiete des Systems
    - Kennt Architekturprinzipien und verschiedene SW-Architekturen
    - Kennt Methoden und Werkzeuge
    - …

V-Modell XT: Rollen

- Beispiel SW-Architekt (Forts.):
  - **Verantwortlich für**
    - Datenbankentwurf
    - Implementierungs-, Integrations-und Prüfkonzept SW
    - SW-Architektur
    - SW-Spezifikation
    - …
  - **Mitwirkend an**
    - Änderungsentscheidung
    - Ausbildungsunterlagen
    - Implementierungs-, Integrations- und Prüfkonzept
    - Instandhaltungsdokumentation
    - …

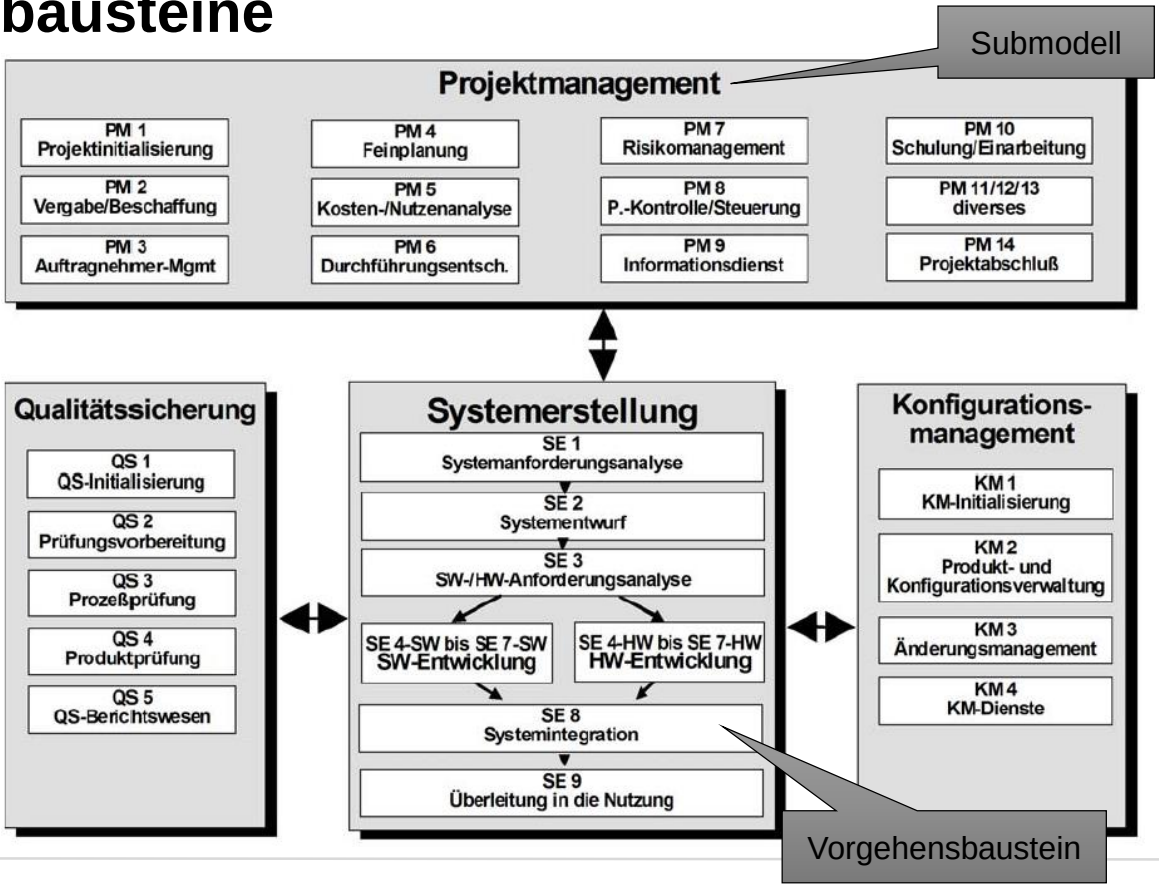
Im Spezifikationsdokument: Hyperlink zum Vorgehensbaustein 3.10.6 Datenbankentwurf

V-Modell XT: Submodelle/Vorgehensbausteine

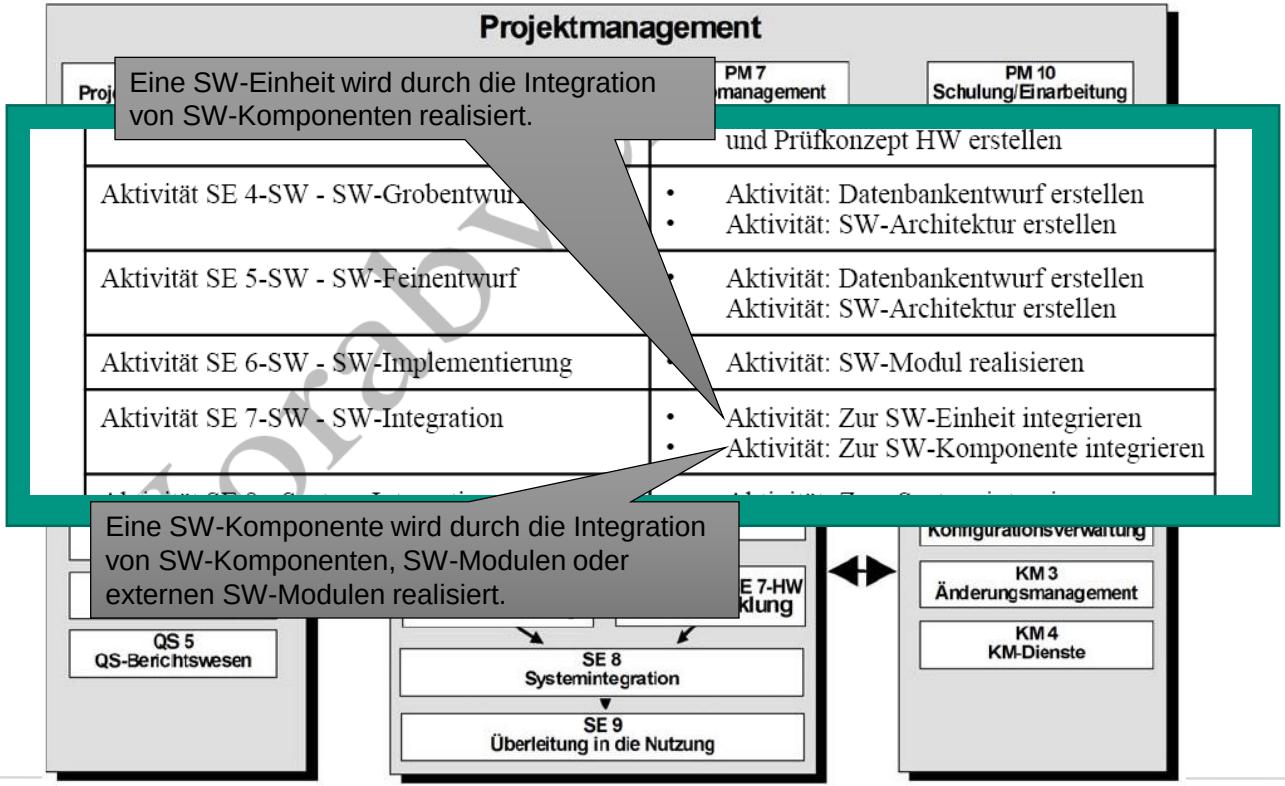
- Im „alten“ V-Modell 97 existierten 4 Submodelle, die so zugeschnitten waren, dass es hinsichtlich der dort auftretenden Rollen keine Überschneidungen gab
  - Submodell Projektmanagement (PM)
  - Submodell Qualitätssicherung (QS)
  - Submodell Konfigurationsmanagement (KM)
  - Submodell Systemerstellung (SE)

- Das aktuelle V-Modell XT gliedert diese Submodelle in sog. „Vorgehensbausteine“.

V-Modell XT: Zuordnung Submodelle zu Vorgehensbausteine

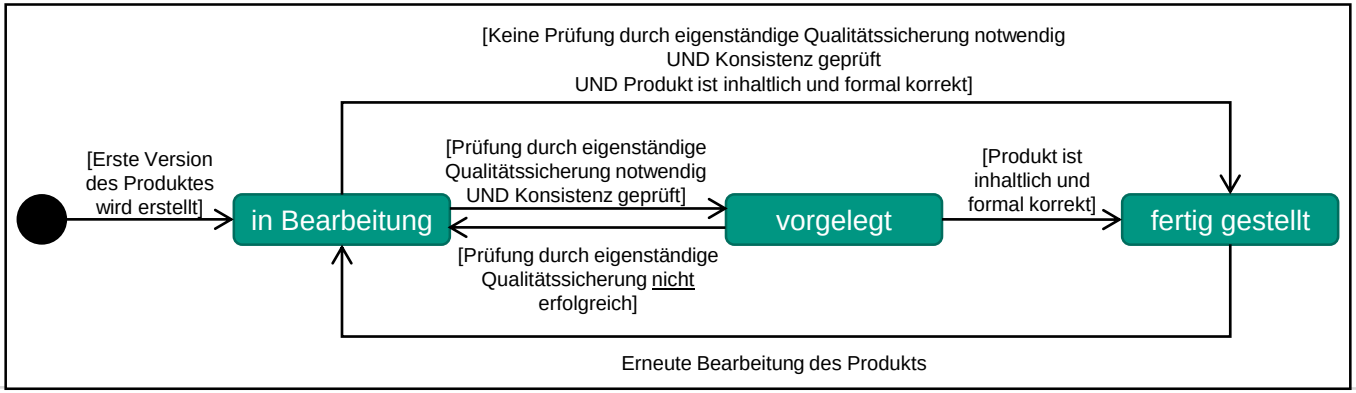


V-Modell XT: Abbildung Submodelle/Vorgehensbausteine



V-Modell XT: Produktzustände

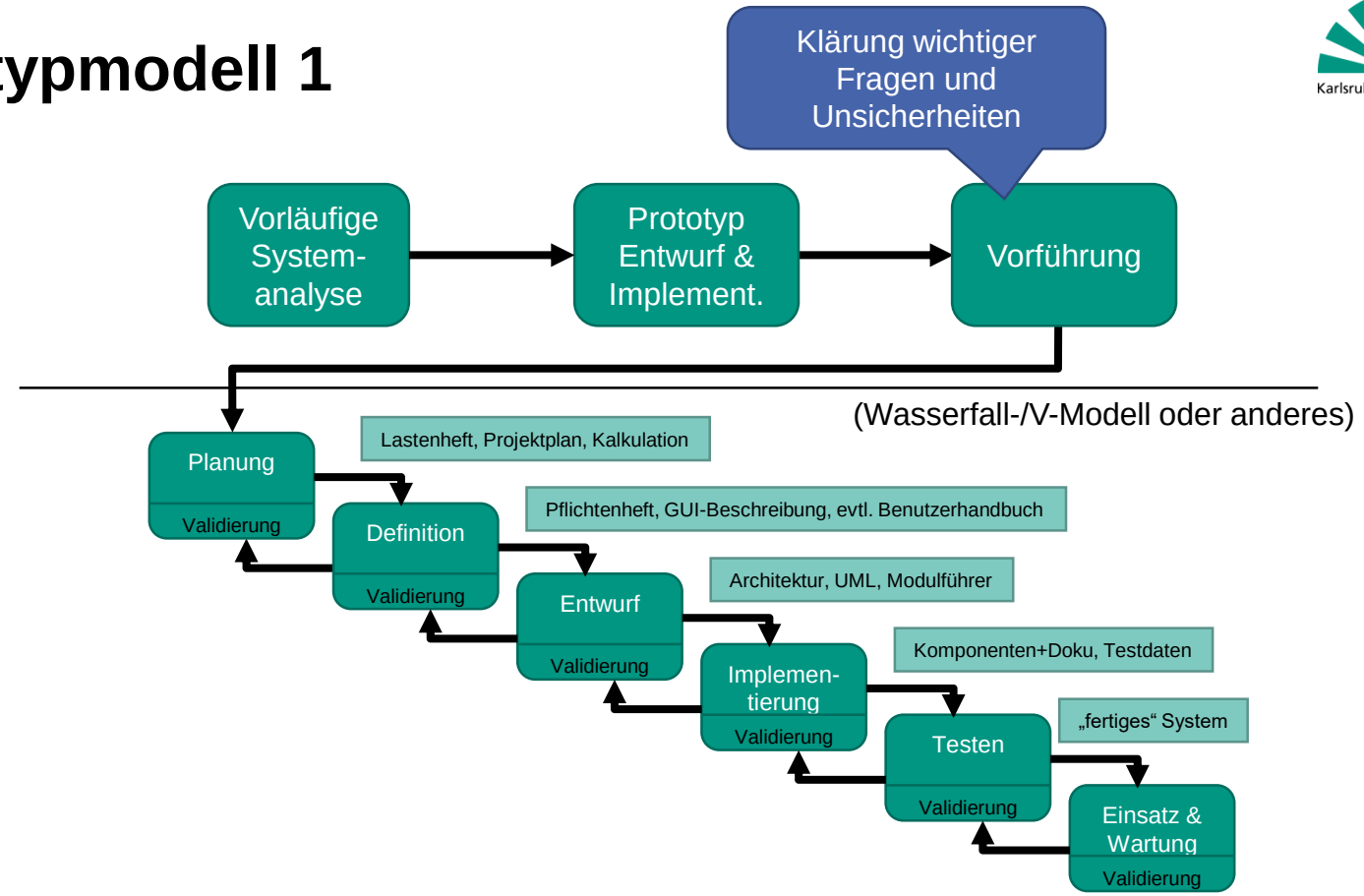
- Jedes definierte Produkt durchläuft vier Zustände:
  - in Planung
  - in Bearbeitung
  - vorgelegt
  - fertig gestellt
- wobei folgende Übergänge möglich sind:



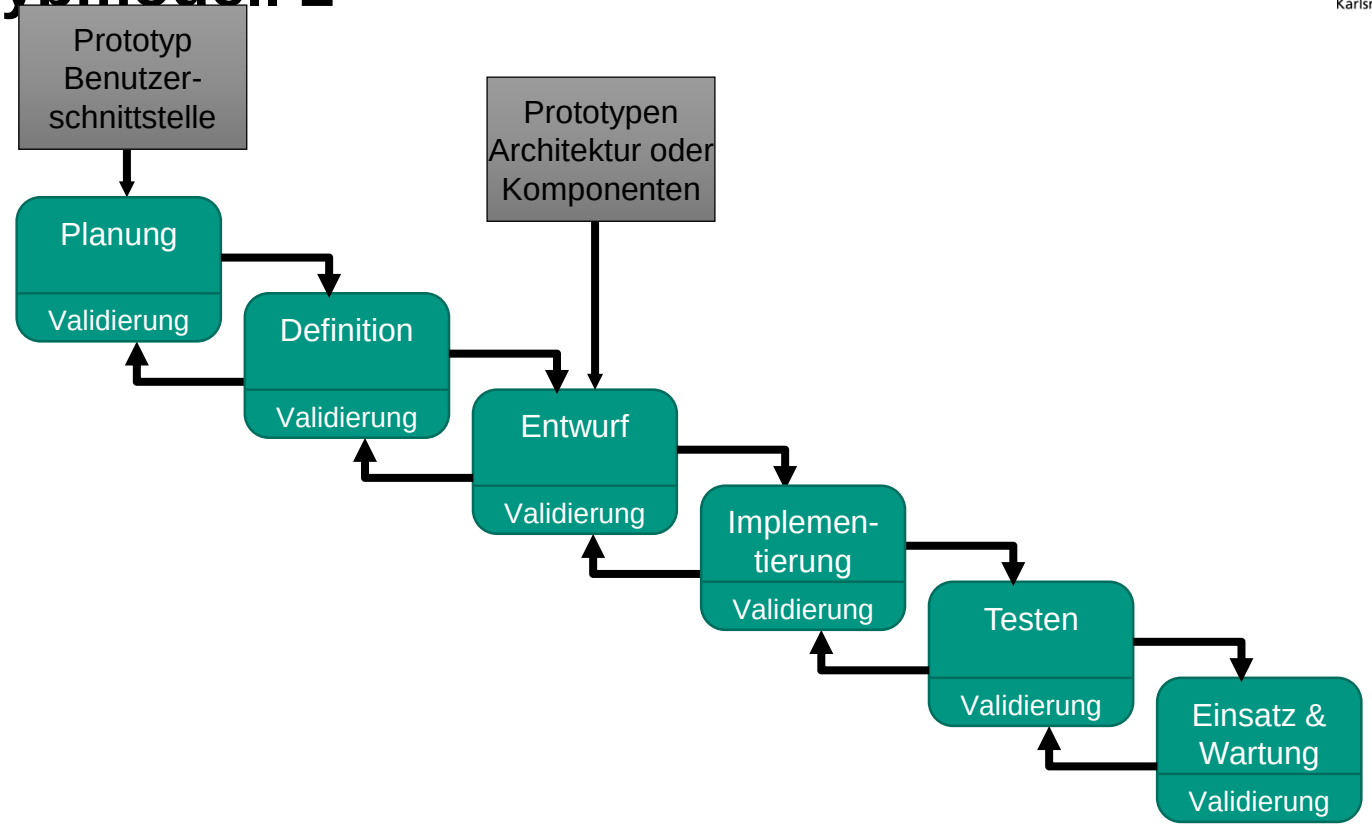
Prototypmodell

- Geeignet für Systeme, für die keine vollständige Spezifikation ohne explorative Entwicklung oder Experimentation erstellt werden kann
- Der Prototyp (eingeschränkt funktionsfähiges System) kann Arbeitsmoral und Vertrauen zwischen Anbieter und Kunden stärken
- Frederick P. Brooks in „The Mythical Man-Month“ (Ch. 11, S. 116): *Plan to throw one away; you will, anyhow.*
- Wichtig: **PROTOTYP WEGWERFEN!**

Prototypmodell 1



Prototypmodell 2



Iteratives Modell

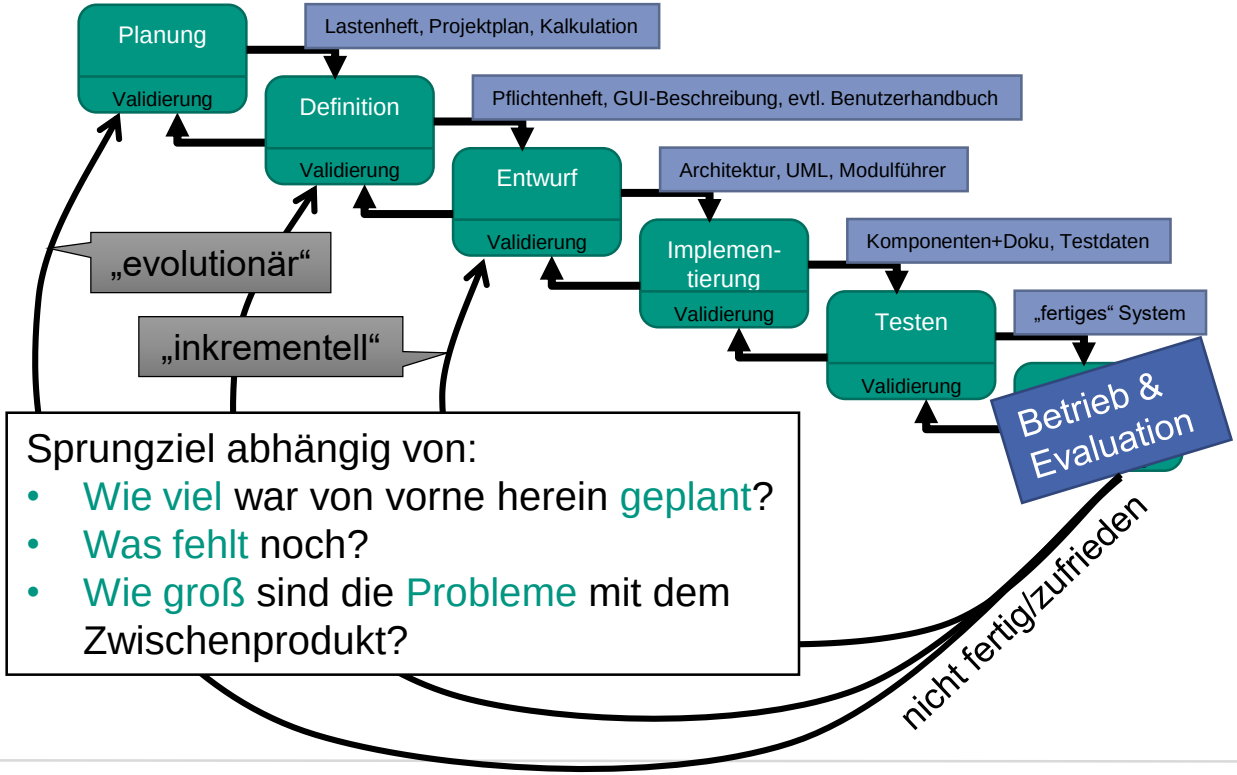
- Auch „*successive versions*“ – Erweiterung der Prototypen-Idee
- Idee: Zumindest Teile der Funktionalität lassen sich klar definieren und realisieren
- Daher: Funktionalität wird Schritt für Schritt erstellt und dem Produkt „hinzugefügt“
- Gleiche Vorteile und Einsatzgebiete wie Prototypmodell
- Versuch, mehr weiter zu verwenden als beim Prototypmodell

Iteratives Modell

- In der Literatur unterschiedliche Ansätze für Planungs- und Analysephase
  - Evolutionär: Plane und analysiere nur den Teil, der als nächstes hinzugefügt wird (x-faches Wasserfall-Modell)
    - Risiko, dass sich der nächste Teil aufgrund struktureller Schwierigkeiten nicht integrieren lässt und deshalb Teile noch mal gemacht werden müssen
  - Inkrementell: Plane und analysiere alles und iteriere dann n-mal über Entwurfs-, Implementierungs- und Testphase
    - Erfordert vollständige Planung und Analyse, was ja eigentlich mit diesem Modell umgangen werden soll…

→ Mischformen und Flexibilität angebracht

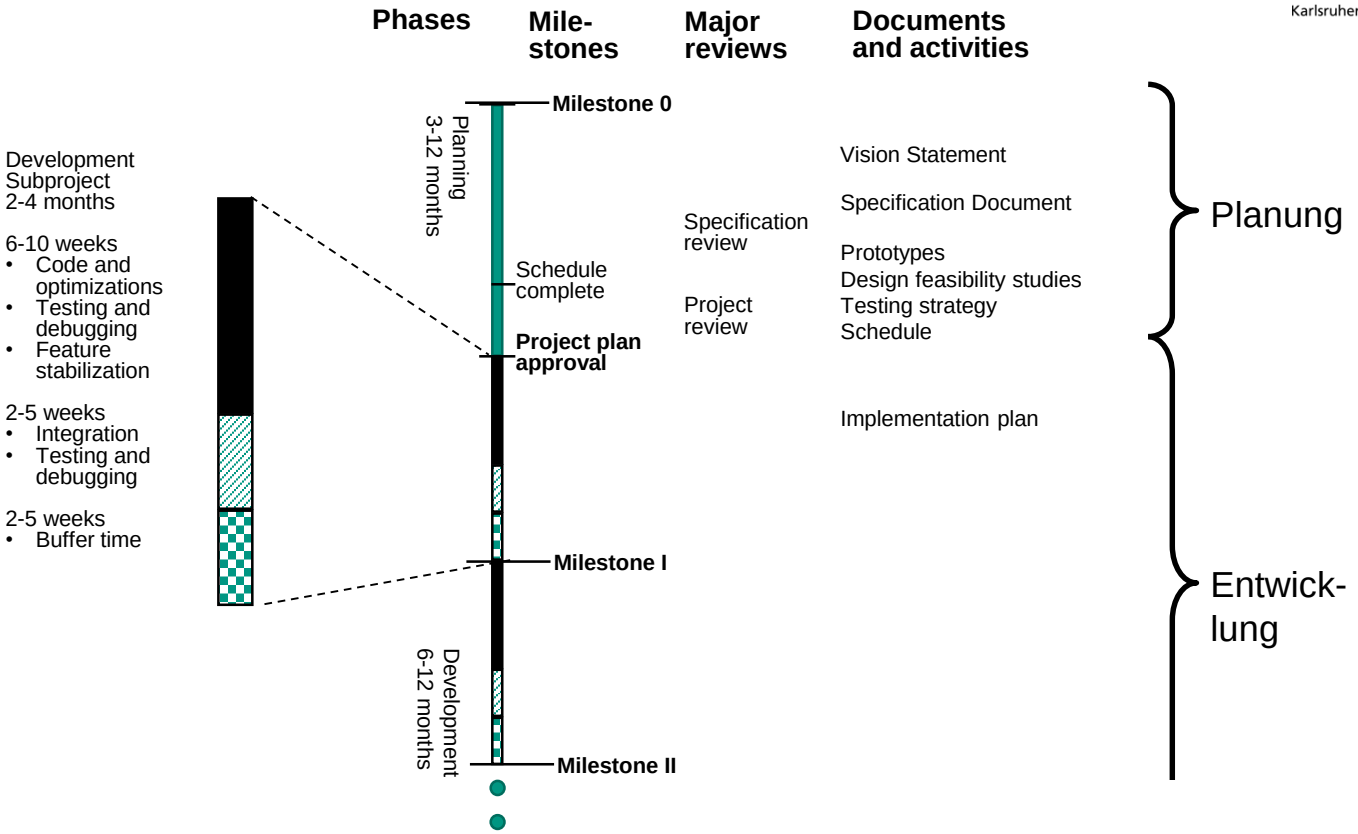
Iteratives Modell



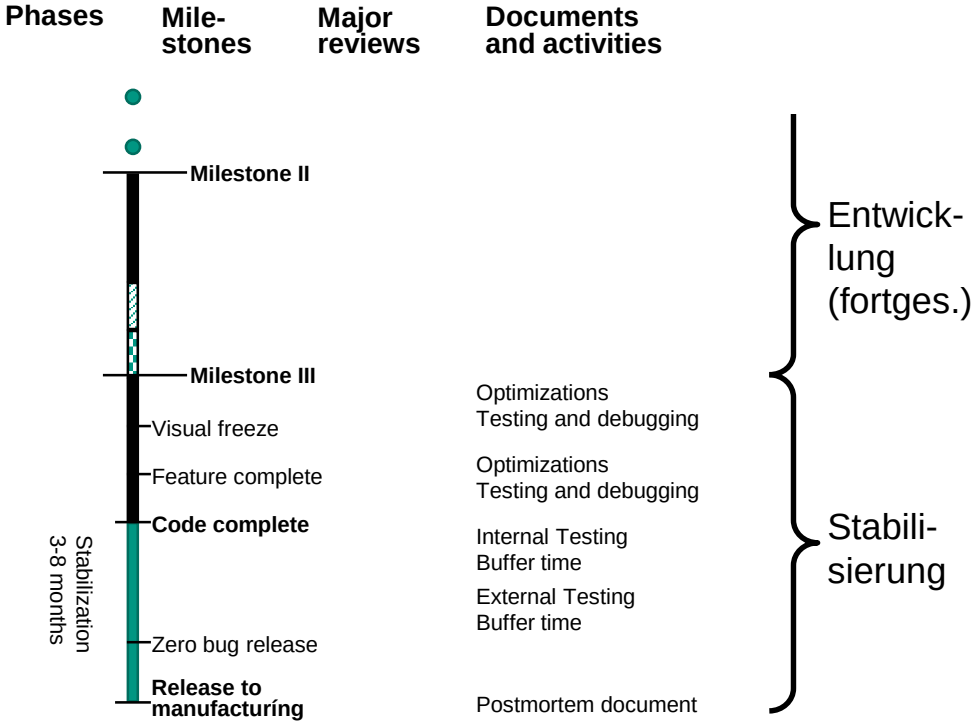
Synchronisiere und Stabilisiere

- Auch „Microsoft-Modell“ (siehe [CusSel95])
- Ansatz
  - Organisiere die 200 Programmierer eines Projektes (z.B. Windows 95) in „kleinen Hacker-Teams“
    → Freiheit für eigene Ideen/Entwürfe
  - Aber: Synchronisiere regelmäßig (nächtlich)
  - Und Stabilisiere regelmäßig (Meilensteine, 3 Mon.)
- Drei Phasen
  - Planungsphase
  - Entwicklungsphase in 3 Subprojekten
  - Stabilisierungsphase

Synchronisiere und Stabilisiere



Synchronisiere und Stabilisiere



Synchronisiere und Stabilisiere Planungsphase

≥ 30% Änderungen während der Projektlaufzeit

- **Wunschbild** (*vision statement*): Manager (Vermarktungsfachleute) identifizieren und priorisieren Produkteigenschaften aufgrund umfangreicher Sammlungen von Kundenwünschen
- **Spezifikation**: Manager und Entwickler definieren Funktionen, Architektur und Komponenten-Abhängigkeiten anhand des Wunschbildes
- **Zeitplan und Teamstruktur**: Aufteilung der Aufgaben auf „Produktfunktionsgruppen“ mit je
  - 1 Manager
  - 3-8 Entwickler
  - Genau so viele Tester (arbeiten 1:1 parallel zu Entwicklern)
- **Dauer**: 3 -12 Monate (je nach Komplexität)

Synchronisiere und Stabilisiere Entwicklungsphase

- Aufgaben
  - Manager koordinieren Weiterentwicklung der Spezifikation
  - Entwickler entwerfen, codieren und entfernen Fehler
  - Tester arbeiten parallel zu ihrem Entwickler
- Drei Teilprojekte → Drei Meilensteine
  - Erstes Drittel der geplanten Funktionalität, wichtigste Funktionen
  - Zweites Drittel
  - Letztes Drittel: unwichtigste Funktionen

Synchronisiere und Stabilisiere Entwicklungsphase

- Ausbuchen, Bearbeiten, Übersetzen & Testen
- Einbuchen (bei Bedarf, i.d.R 2x pro Woche)
- Nächtliches, vollständiges (Neu-)Übersetzen aller Quellen
- Automatische Regressionstests
- Sanktionen (Ausprägung je nach Team) für das Verursachen von Fehlern bei der Integration. Diese Fehler müssen sofort behoben werden, da sie andere Entwickler aufhalten!
- Zum Ende jedes Subprojektes werden alle entdeckten Fehler beseitigt (Subprojekt-Stabilisierung)

Synchronisiere und Stabilisiere
Stabilisierungsphase

- Aufgaben
  - Manager koordinieren Beta-Tester und sammeln Rückmeldungen
  - Entwickler stabilisieren Code
  - Tester isolieren Fehler
- Tests
  - Interne Tests (innerhalb von Microsoft)
  - Externe Tests (Tests bei „Beta-Testern“)
- Vorbereitung der Auslieferung
  - Fertiges Produkt auf den „Master“-Rohling brennen
  - Dokumentation für den Druck aufbereiten
- Dauer: 3 - 8 Monate

Synchronisiere und Stabilisiere
Zeitplan

- Planung: 3 - 12 Mon.
- Jedes der 3 Teilprojekte: 2 - 4 Mon., wobei
  - 6 - 10 Wochen Codieren, Optimieren, Testen, Fehlersuche und Stabilisieren der Funktionalität
  - 2 - 5 Wochen Integration, Test und Fehlersuche
  - 2 - 5 Wochen Pufferzeit
- Stabilisierung: 3 - 8 Mon.

Entwicklung: 6 - 12 Mon.

12 - 32 Monate gesamt

Synchronisiere und Stabilisiere

- Pro
  - Effektiv durch kurze Produktzyklen
  - Priorisierung nach Funktionen
  - Natürliche Modularisierung nach Funktionen
  - Fortschritt auch ohne vollständige Spezifikation möglich
  - Viele Entwickler arbeiten in kleinen Teams und damit genau so effektiv wie wenige
  - Rückmeldungen können frühzeitig einfließen

Synchronisiere und Stabilisiere

- Kontra
  - Ungeeignet für manche Art von Software – Architekturprobleme, mangelhafte Fehlertoleranz, Echtzeitfähigkeit
  - Mündliche Arbeitsweise: ad-hoc-Prozesse in jedem Team, kein Lernen über Teamgrenzen
  - Alle 18 Monate sind 50% des Codes überarbeitet worden (Code- Instabilität)

Synchronisiere und Stabilisiere
Vergleich mit Phasenmodell

Sync-and-Stabilize	Sequential Development
Product development and testing done in parallel	Separate phases done in sequence
Vision statement and evolving specification	Complete „frozen“ specification and detailed design before building the product
Features prioritized and built in 3 or 4 milestone subprojects	Trying to build all pieces of a product simultaneously
Frequent synchronizations (daily builds) and intermediate stabilizations (milestones)	One late and large integration and system test phase at project´s end
„Fixed“ release and ship dates and multiple release cycles	Aiming for feature and product „perfection“ in each project cycle
Customer feedback continuous in the development process	Feedback primarily after development as inputs for future projects
Product and process design so large teams work like small teams	Working primarily as a large group of individuals in a separate functional department

Literatur

[Wiki] http://de.wikipedia.org/wiki/Wasserfallmodell
http://de.wikipedia.org/wiki/V-Modell

[FHDarm04] Andelfinger et al., Script zur Vorlesung „Softwaretechnik“, 2004, Kap. 4.3, zum Thema Iteratives Modell

[CusSel95] M. A. Cusumano, R. W. Selby, 1997, ACM, „How Microsoft builds software“, unter http://portal.acm.org/citation.cfm?id=255698

[MalPal99] Malik, S., Palencia, J., „Synchronize and Stabilize vs. Open-Source“unter http://www.cs.toronto.edu/~smalik/downloads/paper_314.pdf

[BMI-KBSt] „V-Modell XT“ unter http://www.v-modell-xt.de/