

Vorgehensmodelle (Prozessmodelle)

Problemstellung

- **Ziel:** Die Erstellung von Software-Systemen in ökonomischer Art und Weise, die bestimmten Qualitätsanforderungen genügt.
- Ein **Vorgehensmodell** entspricht einer Strategie für die Durchführung eines (SW-)Projekts.

Wozu Vorgehensmodelle?

- um bei der Produktentwicklung systematisch vorzugehen
- um den Entwicklungsprozess zu strukturieren, Tätigkeiten, Zwischenergebnisse und QS-Maßnahmen zu definieren,
- um den Entwicklern eine Orientierungshilfe zu geben,
- um Meilensteine, Zwischenziele, Termine planen, setzen und überprüfen zu können,
- um Projekte vergleichen, bewerten und aus ihnen lernen zu können

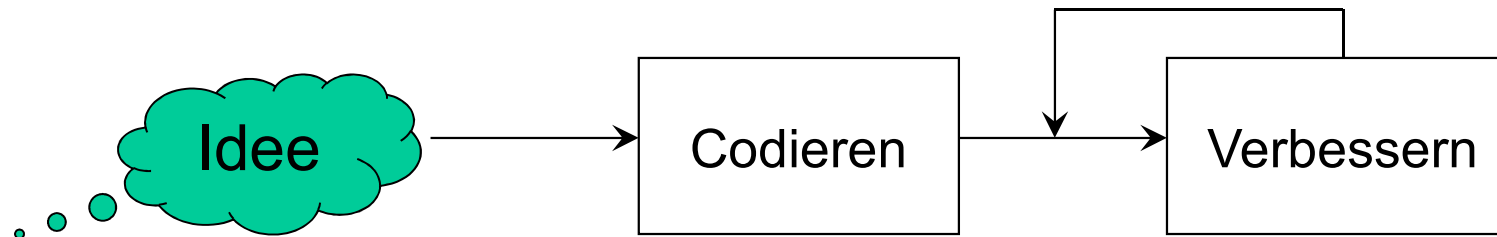
Welche Vorgehensmodelle werden behandelt?

- ✓ Build-and-Fix-Cycle
- ✓ Wasserfallmodell
- ✓ V-Modell
- ✓ Spiralmodell
- ✓ Das Prototypen-Modell
- ✓ Das nebenläufige Modell
- ✓ Das objektorientierte Modell
- ✓ Agile SW-Entwicklung (allgemein)
- ✓ Scrum

Build-and-Fix-Cycle

Build-and-Fix-Cycle (Codieren und verbessern)

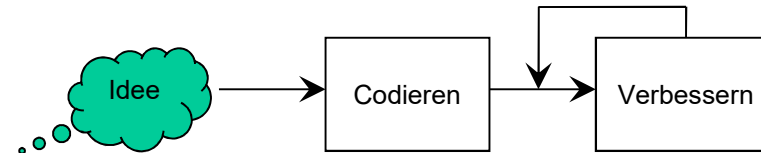
- Einfachstes Vorgehen (von jedem Programmierer bereits angewandt):



- Es wird so lange verbessert, bis es den Qualitätsansprüchen des Programmierers genügt.
- Treten während des Betriebs Änderungswünsche auf, werden diese vom Programmierer in das System eingebracht.

Build-and-Fix-Cycle (Codieren und verbessern)

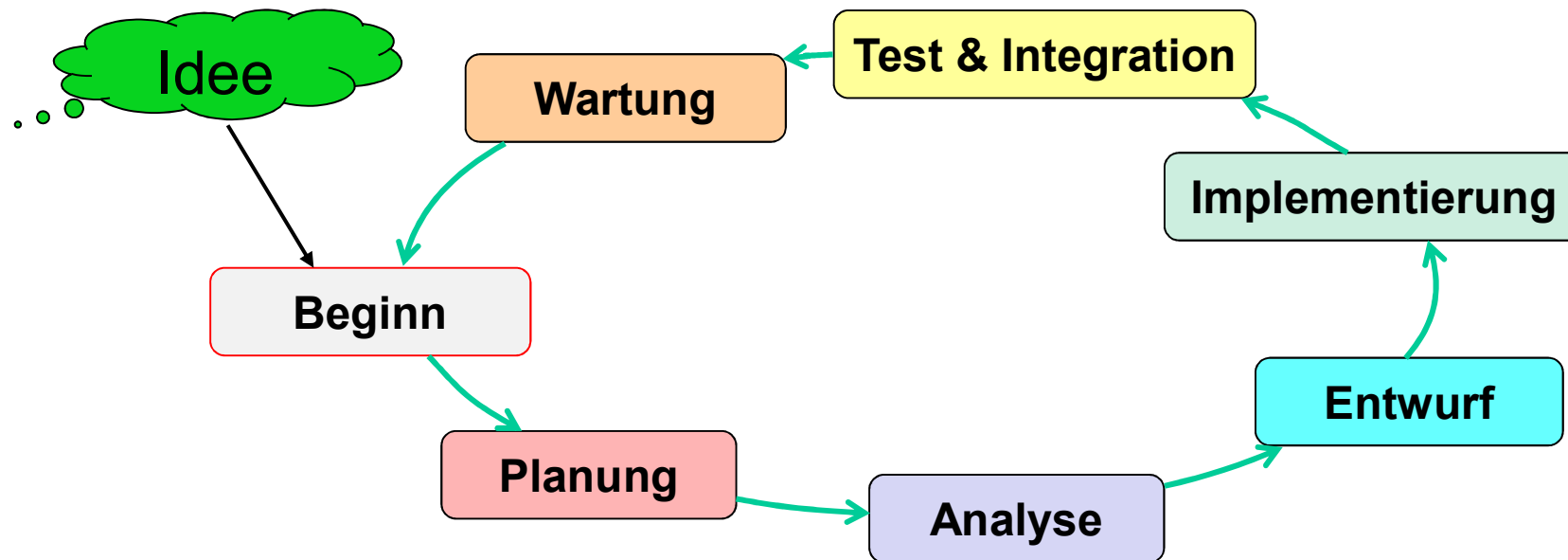
Probleme:



- Bei zu vielen Änderungen kann der Code unbrauchbar werden.
- Meist kann nur der Programmierer selbst den Code verstehen und warten, da selten bis gar nicht kommentiert wird.
- Es wird kaum strukturiert vorgegangen (keine Analyse, kein Entwurf, ...)
- Programme können von hoher Qualität sein, sind jedoch auf kleine, „überschaubare“ Programme begrenzt.

Software-Life-Cycle

Software Development Life Cycle (SDLC)



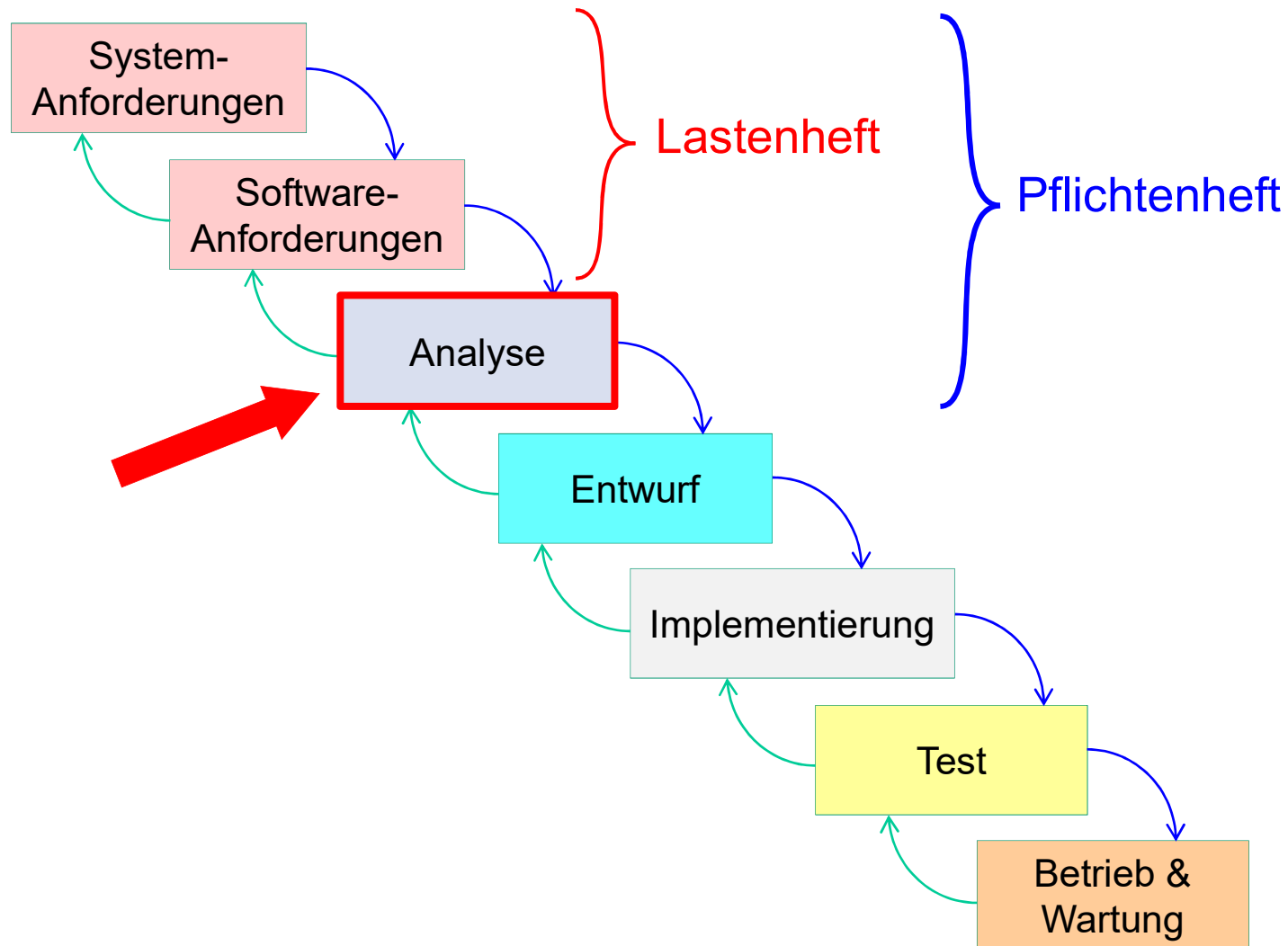
Grundidee:

- Sieht ein **strukturiertes Vorgehen** vor bei der Erstellung von Software
- Wird mindestens einmal durchlaufen
- Sieht keine „Schritte zurück“ vor
- Änderungen erfolgen in neuem Projekt
- Bildet die **Grundlage aller Vorgehensmodelle**

Das Wasserfallmodell

W.Royce & B. Boehm

Das Wasserfallmodell (Schema)



Das Wasserfallmodell (Eigenschaften)

- Besitzt die Möglichkeit von Rückschritten
- Jede Aktivität ist in der richtigen Reihenfolge und in der vollen Breite vollständig durchzuführen.
- Am Ende jeder Aktivität steht ein fertiggestelltes Dokument, d.h. das Wasserfallmodell ist ein dokumentengetriebenes Modell.
- Zwischenergebnisse aus den einzelnen Phasen sind Dokumente, Datenbankschemata und Programme.
- Der Entwicklungsablauf ist **sequentiell**, d.h. jede Aktivität muss beendet sein, bevor die nächste beginnt.

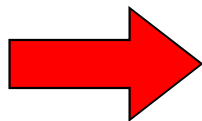
Das Wasserfallmodell (Eigenschaften (2))

- Es orientiert sich am top-down-Vorgehen.
- Es ist einfach, verständlich und benötigt nur wenig Managementaufwand.
- Eine **Benutzerbeteiligung ist nur in der Definitionsphase** vorgesehen, anschließend erfolgen der Entwurf und die Implementierung ohne Beteiligung des Benutzers bzw. Auftraggebers.

Das Wasserfallmodell (Eigenschaften (3))

Beim Wasserfallmodell werden implizite Annahmen gemacht:

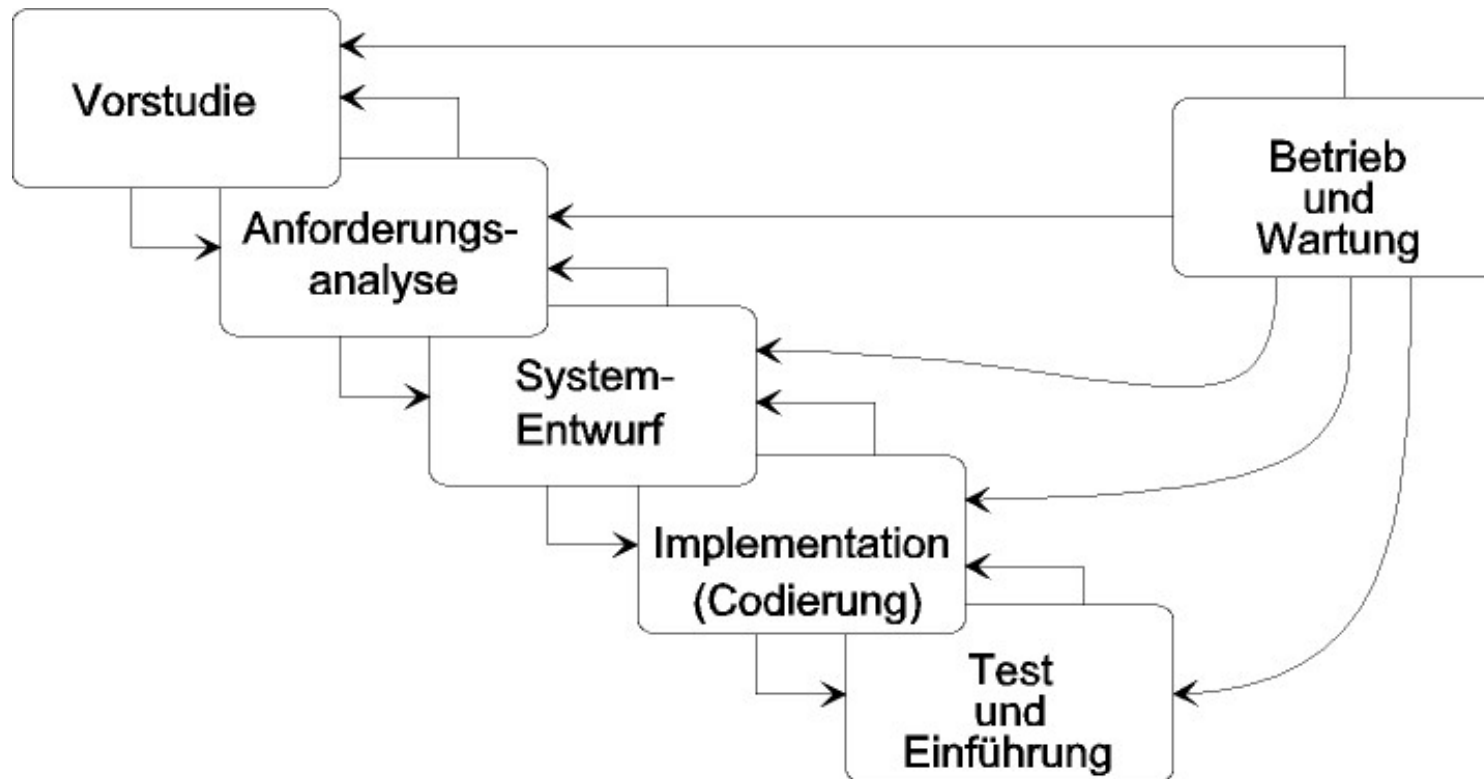
- Anforderungen lassen sich **à priori vollständig feststellen** und **verändern sich nicht oder nur sehr langfristig**.



Pflichtenhefte und Dokumente müssen vollständig ausgearbeitet werden

- Benutzer werden nur am Anfang und Ende der Entwicklung (Produktabnahme) einbezogen.
- Die den Phasen entsprechenden Zwischenergebnisse bilden die Dokumentation des Software-Produkts

Das „erweiterte“ Wasserfallmodell



- Hier wirken sich die Erfahrungen aus Betrieb und Wartung direkt auf den Entwicklungsprozess aus. Neuentwicklungen können durch Erfahrungen vorhergegangener Entwicklungen profitieren.

Voraussetzungen, Inhalte und Ergebnisse der Phasen des Wasserfallmodells

Das Wasserfallmodell (Vorstudie)

Voraussetzungen :

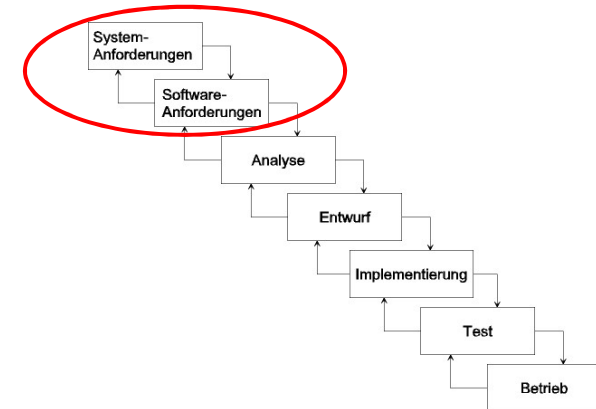
- erkannter Bedarf
- Zielvorstellungen

Inhalt :

- Grobe organisatorische, technische, und fachliche Vorgaben
- terminliche, personelle und wirtschaftliche Anforderungen

Ergebnisse :

- Machbarkeitsstudie (Vorstudie)
- Projektantrag für Analysephase



Das Wasserfallmodell (Analyse)

Voraussetzungen :

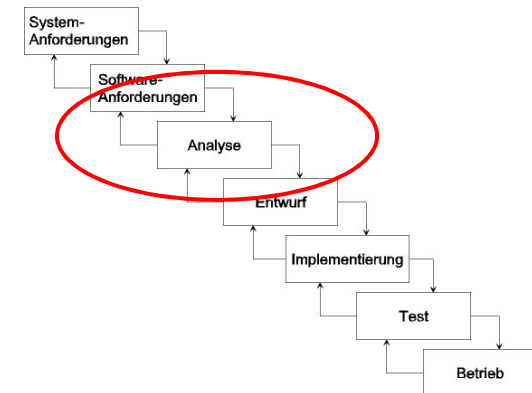
- Machbarkeitsstudie
- akzeptierter Projektantrag für Analysephase

Inhalt :

- Ist- und Sollaufnahme bzw. Ist- und Sollanalyse
- Modellbildung (Lösungsmöglichkeiten, incl. Marktübersicht)
- grobe Beschreibung der Informationsstrukturen und der Funktionen

Ergebnisse :

- Grobkonzept
- Qualitätssicherungssystem
- Projektantrag für die Phase “Fachliches Design”



Das Wasserfallmodell (Fachliches Design - Grobentwurf)

Voraussetzungen :

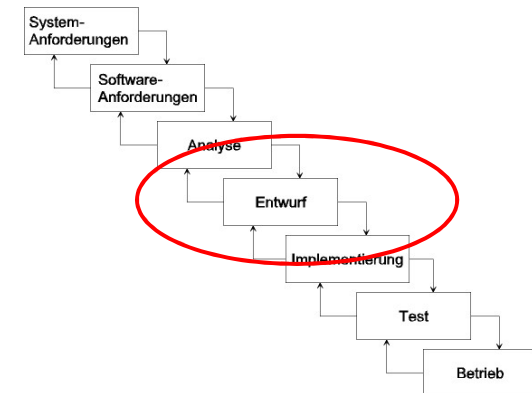
- Freigabe der Phase “Fachliches Design”
- Grobkonzept

Inhalt :

- detaillierte und vollständige Klärung der Leistungen des Anwendungssystems
- vollständige Beschreibung des Funktionsmodells, der Masken- und Listenlayouts, des (logischen) Datenmodells und der Schnittstellen zu anderen Anwendungen
- Testfallermittlung für den Abnahmetest

Ergebnisse :

- Fachkonzept (Pflichtenheft)
- Projektantrag für die Phase “DV-technisches Design”



Das Wasserfallmodell (DV-Techn. Design - Feinentwurf)

Voraussetzungen :

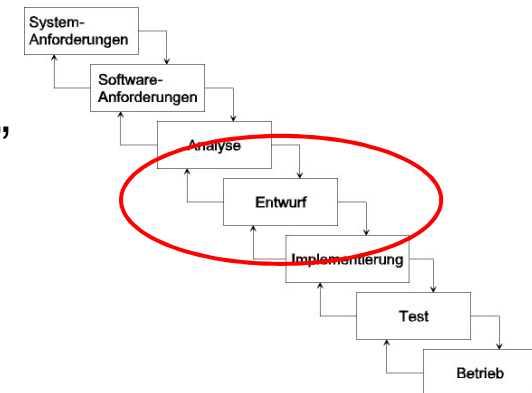
- Freigabe der Phase “DV-technisches Design”
- Fachkonzept

Inhalt :

- physische Strukturierung der Daten
- Beschreibung der Datenbanken und Dateien
- Modularisierungskonzept - Beschreibung der Schnittstellen der Module

Ergebnisse :

- Produktarchitektur (DV-Konzept)
- Testplanung
- Arbeitsaufträge (für Programme, Datenbanken, ...)



Das Wasserfallmodell (Realisierung (Implementierung))

Voraussetzungen :

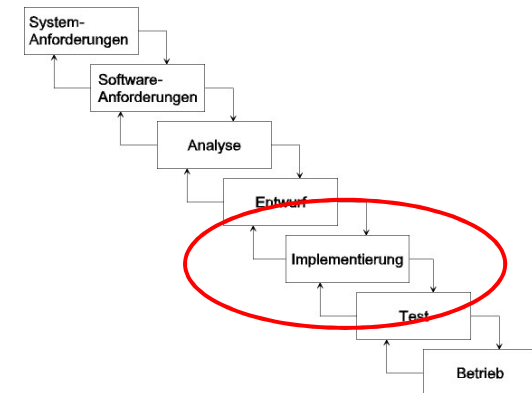
- Produktarchitektur

Inhalt :

- Erstellung der Datenbanken
- Programmdetaillierung
- Codierung
- Komponententest

Ergebnisse :

- Software (ausgetestete Komponenten)
- Modul- und Datenbeschreibung (Programmdokumentation)
- Testumgebung



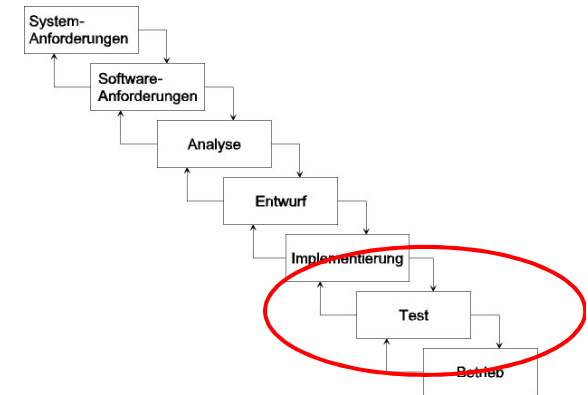
Das Wasserfallmodell (Integration)

Voraussetzungen :

- Software (ausgetestete Komponenten)
- Testumgebung mit Testdaten
- Anforderungsspezifikation (aus Fachkonzept)

Inhalt :

- Integrationstest, d.h. die schrittweise Zusammenfügung der ausgetesteten Einzelkomponenten zum DV-Anwendungssystem
- Systemtest: das Anwendungssystem wird mit den vorgegebenen Testdaten getestet und damit auf den Abnahmetest vorbereitet
- Zusammenstellung der Testdokumentation
- Validierung der Anforderungsspezifikation
- Abnahmetest (unter Beteiligung der Fachabteilung)



Ergebnisse:

- getestetes Anwendungssystem
- Testdokumentation

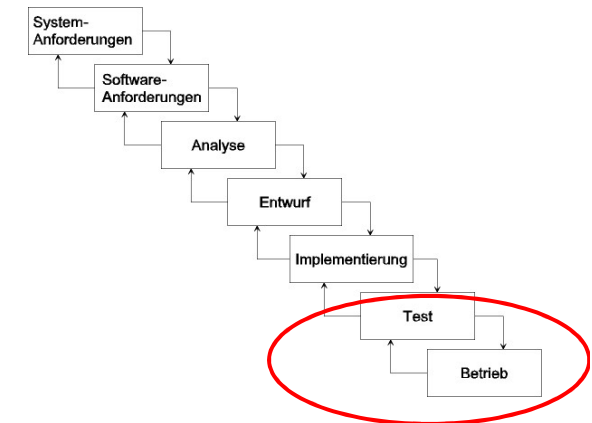
Das Wasserfallmodell (Übergabe und Einführung)

Voraussetzungen :

- getestetes Anwendungssystem
- Testdokumentation
- Fachkonzept

Inhalt :

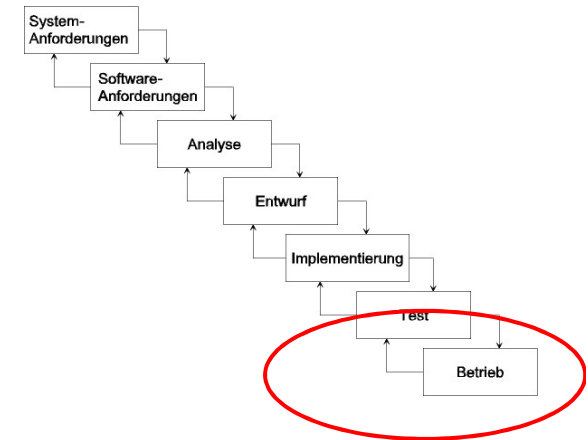
- Zusammenstellen der Schulungsunterlagen und der Systemdokumentation unter Verwendung der Phasenergebnisse
- Benennung einer verantwortlichen Stelle für die Betreuung des Anwendungssystems
- Schulung der Pilotanwender
- Übergabe des Anwendungssystems einschließlich der zugehörigen Dokumentation an Anwender, Rechenzentrum und Betreuungsstelle
- evtl. protokollarische Systemabnahme oder Erprobungsphase
- Integration des Anwendungssystems in das organisatorische Umfeld



Das Wasserfallmodell (Übergabe und Einführung)

Ergebnisse der Übergabe und Einführung:

- ein produktives DV-Anwendungsprogramm
- Schulungsunterlagen - Systemdokumentation
- evtl. die Abnahmeprotokolle



Das V-Modell

Das V-Modell (Eigenschaften (1))

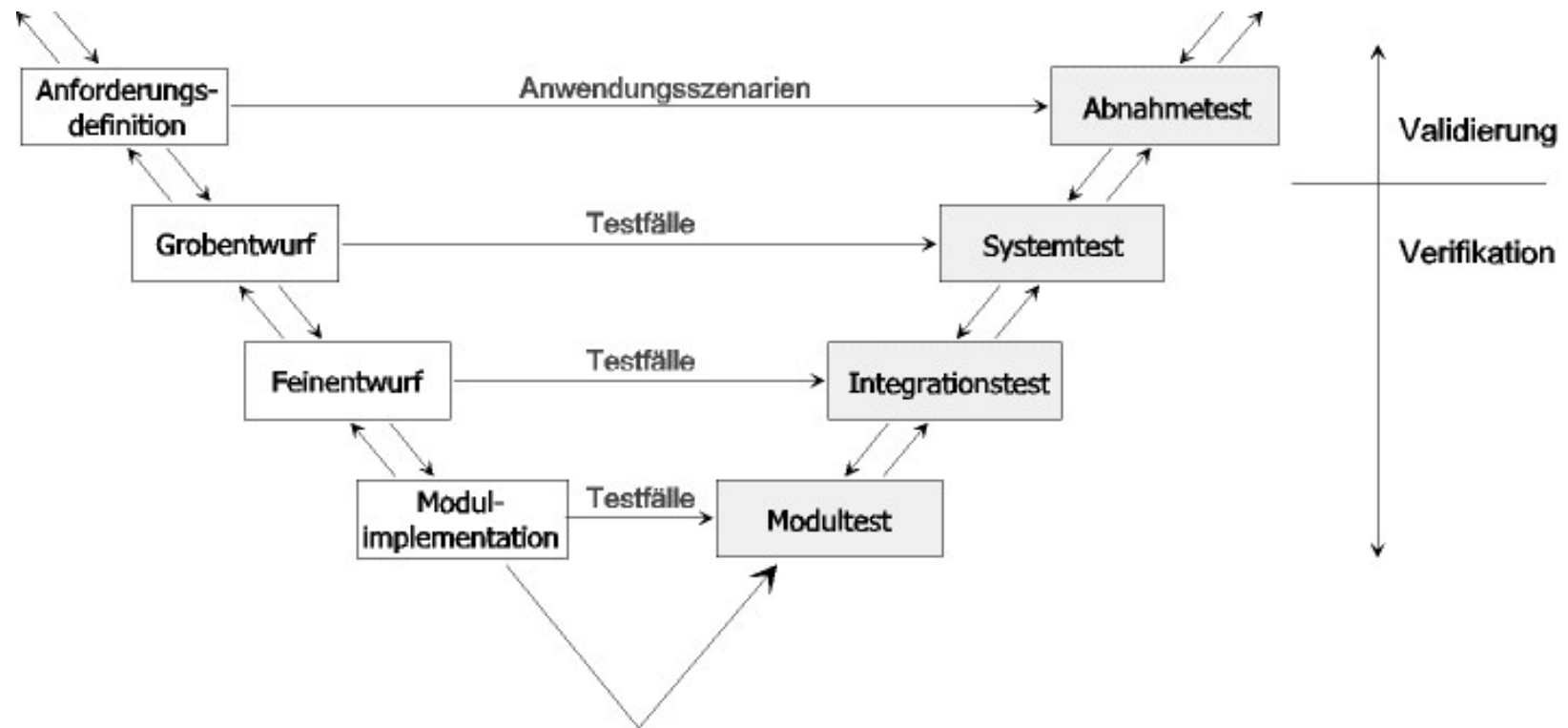
- Vorgehensmodell zur Planung und Durchführung von IT-Vorhaben (seit 1997)
- **Entwicklungsstandard für IT-Systeme des Bundes**
- V-Modell XT - Erweiterung des V-Modells 97 (seit 2005)
- Komplexes Vorgehensmodell für große Projekte

Das V-Modell (Schema)

Gliederung des SW-Entwicklungsprozesses in **zwei korrelierende Phasen**:

Erste Hälfte der Phasen: „herkömmliches“ Wasserfallmodell,

Zweite Hälfte der Phasen: bildet die **Tests** für die Produkte der 1. Phase



Das V-Modell (Eigenschaften (2))

Erste V-Modelle: B. Boehm 1979, W. Hesse 1981

Warum V-artige Darstellung des SW-Entwicklungsprozesses?

- Vorteile des Wasserfall-Modells bleiben erhalten
- Symmetrie
- Querbezüge zwischen "frühen" und "späten" Phasen
- Möglichkeit, zwischen Fach-/Anwendungswelt (oben) und DV-Welt (unten) zu unterscheiden.
- Einpassung in "Software-Entwicklungs-Landschaft"


Das V-Modell (Aufteilung in Submodelle)

Aufteilung in Submodelle für

- **Software-Erstellung (SE)**
 - Systementwicklung, Dokumentation
- **Qualitätssicherung (QS)**
 - Anforderungen, Methoden
- **Konfigurationsmanagement (KM)**
 - Verwaltung von Konfigurationen, Produkten, Rechten, Änderungen
- **Projektmanagement (PM)**
 - Planung, Kontrolle, Steuerung, Information

Das V-Modell (positive Eigenschaften)

Positive Eigenschaften:

- Ausnutzung der Vorteile von V-Modellen (Systematik, Symmetrie, Querbezüge)
- Sorgfältige Ausarbeitung, konsistente Begriffsbildung
- Wichtige Rolle der Submodelle Softwareerstellung (SWE), Qualitätssicherung (QS), Konfigurationsmanagement (KM), Projektmanagement (PM)
- Detaillierter und sorgfältiger ausgearbeitet als die meisten der traditionellen Phasenmodelle
- spiegelt "Stand der Technik" Ende der 80er Jahre wider
  **V-Modell XT**

Das V-Modell (Kritik)

Verbesserungsmöglichkeiten:

- Symmetrie ist nicht konsequent durchgehalten
- Traditionelle Phaseneinteilung dominiert
- Keine zyklische Entwicklung
- **besser:** Produkt-orientierte Anforderungen
- Produktbegriff ist sehr allgemein (z.B. nicht passend für Dokumente)
- Systemzerlegung: Braucht es so viele Hierarchiestufen ?
(Subsystem, Segment, Konfigurationseinheit, Komponente, Modul)
- Bezüge zwischen QS- und SWE -Submodell sind wenig explizit

Das V-Modell (Kritik (2))

Was stärker reflektiert werden könnte:

- Objektorientierte Entwicklung
- Evolutionärer Charakter vieler Entwicklungen
- Eigenständigkeit von SW-Bausteinen (Wiederverwendung)
- Gefahr von "Software-Bürokratie"
- Eignung nicht nur für große Projekte

Das Spiral-Modell

Das Spiral-Modell

1988 von Barry W. Boehm beschrieben

- Weiterentwicklung des Wasserfallmodells
- Freie problembezogene Kombination aller bereits existierender Ansätze unter ständiger Kontrolle des Managements.
- Entwicklungsprozess als iterativer Prozess, wobei jeder Zyklus folgende Aktivitäten enthält:
 1. Festlegung von Zielen, Alternativen und Rahmenbedingungen
 2. Evaluierung der Alternativen, erkennen und reduzieren von Risiken
 3. Realisierung und Überprüfung des Zwischenprodukts
 4. Planung der Projektfortsetzung.



Das Spiral-Modell (Eigenschaften)

- Meta-Modell
- Risikominimierung als oberstes Ziel
- Keine Trennung von Entwicklung und Wartung
- Durchlaufen von vier zyklischen Schritten für jede Verfeinerungsebene und jedes Teilprodukt
- Ergebnisse des letzten Zyklus → Ziele des nächsten Zyklus
- Bei Bedarf separate Spiralzyklen für verschiedene Komponenten

Das Spiral-Modell (Bewertung)

Vorteile:

- + Flexibles Modell und leichte Umdirigierung
- + Regelmäßige Risikoüberprüfung des Prozessablaufs
- + Keine Festlegung auf ein Prozessmodell
- + Frühzeitige Eliminierung von ungeeigneten Alternativen und Fehlern
- + Integrierte Risikoabwägung
- + Betrachtung von Alternativen
(Kauf, Out-Sourcing, Wiederverwendung von SW)

Nachteile:

- Hoher Managementaufwand
- Schlechter für kleine und mittlere Projekte

 Sehr flexibles Modell zur Betrachtung von Alternativen.

Das Prototypen-Modell

Das Prototypen-Modell (Ziele)

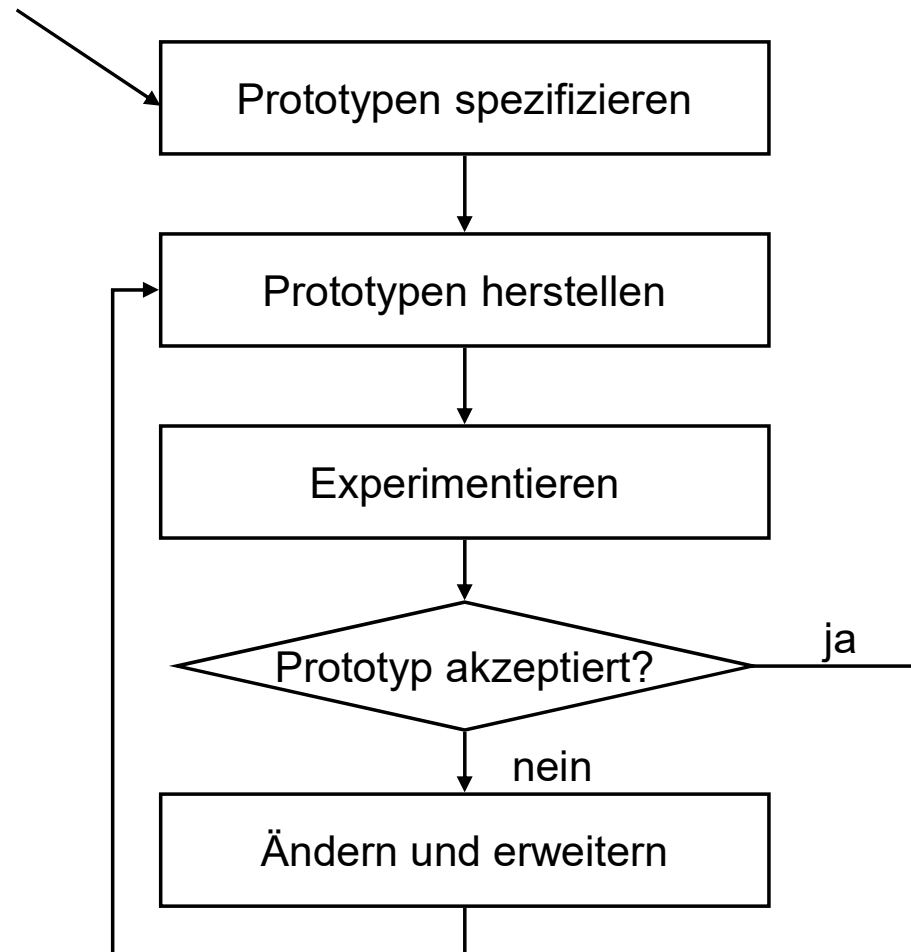
Ziel: Lösung der folgenden Probleme:

- Schwierigkeiten, Anforderungen vollständig zu definieren
- Einbeziehen von Anwendern in die Entwicklung
- Auswahl alternativer Lösungsmöglichkeiten
- Sicherstellung der Realisierbarkeit
- Frühzeitiges Marketing

Lösungsansatz: Frühzeitige Erstellung von lauffähigen Prototypen für

- Tests und Klärung von Problemen
- reine Produktdefinition, danach Neuentwicklung
- frühe Produktversion mit inkrementeller Weiterentwicklung

Das Prototypen-Modell (Schema)



Arten von Prototypen

Demonstrations-Prototyp

- „*Throw-Away Programming*“
- Ein erster Eindruck des Produktes (Rapid Prototyping)
- Meist für die Auftragsakquisition

Labormuster

- „*Experimental Programming*“
- Problemanalyse
- Beantwortung von Konstruktionsfragen

Pilotsystem

- „*Exploratory/Evolutionary Programming*“
- Entwicklung des Kerns eines Produktes
- Bei einem bestimmten Stand wird der Prototyp zum Produkt

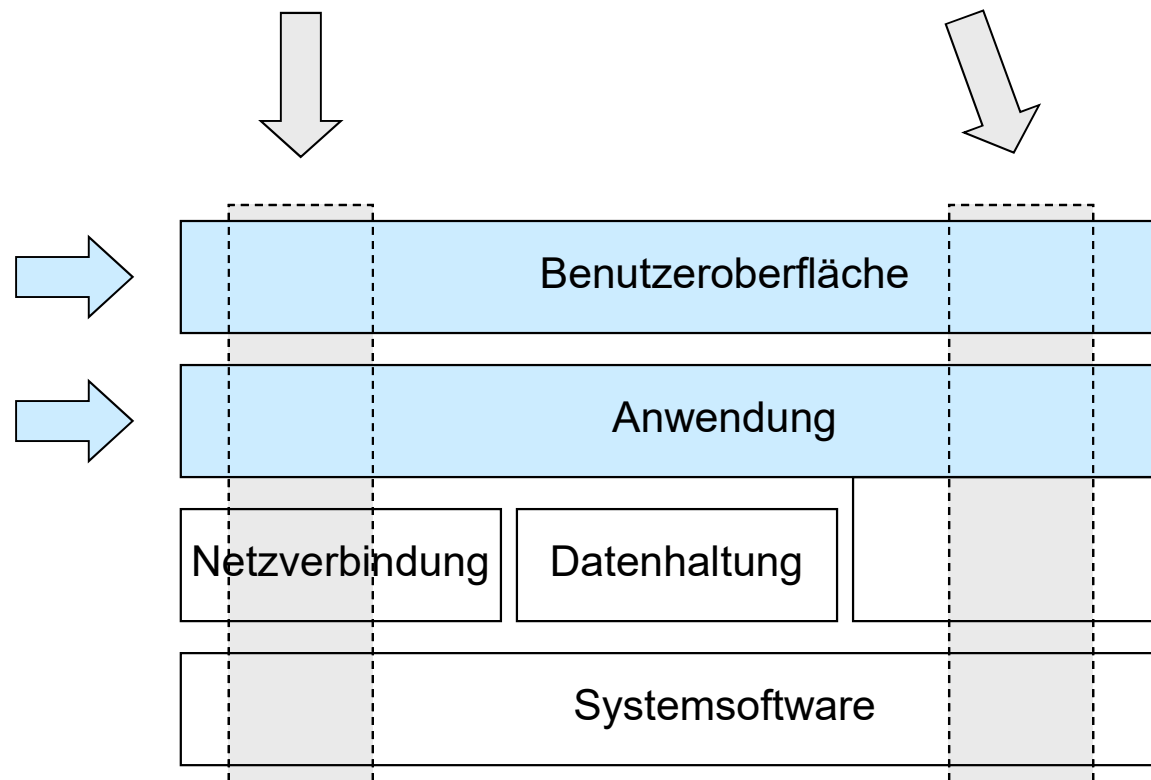
Vertikale und Horizontale Prototypen

Vertikaler Prototyp

- Vollständige Funktionalität von Teilaspekten eines Systems

Horizontaler Prototyp

- Implementierung einer vollständigen Ebene ohne dahinter stehender Funktionalität



Das Prototypen-Modell (Vorteile)

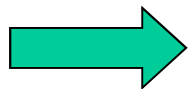
Vorteile:

- + Reduzierung des Entwicklungsrisikos
- + Sinnvolle Integration in andere Prozessmodelle möglich
- + Schaffung einer starken Rückkopplung zwischen Endbenutzern und Herstellern
- + Schnelle Erstellung von Prototypen durch geeignete Werkzeuge
- + Verbessert die Planung der Software-Entwicklung
- + Labormuster fordern die Kreativität für Lösungsalternativen.

Das Prototypen-Modell (Nachteile)

Nachteile:

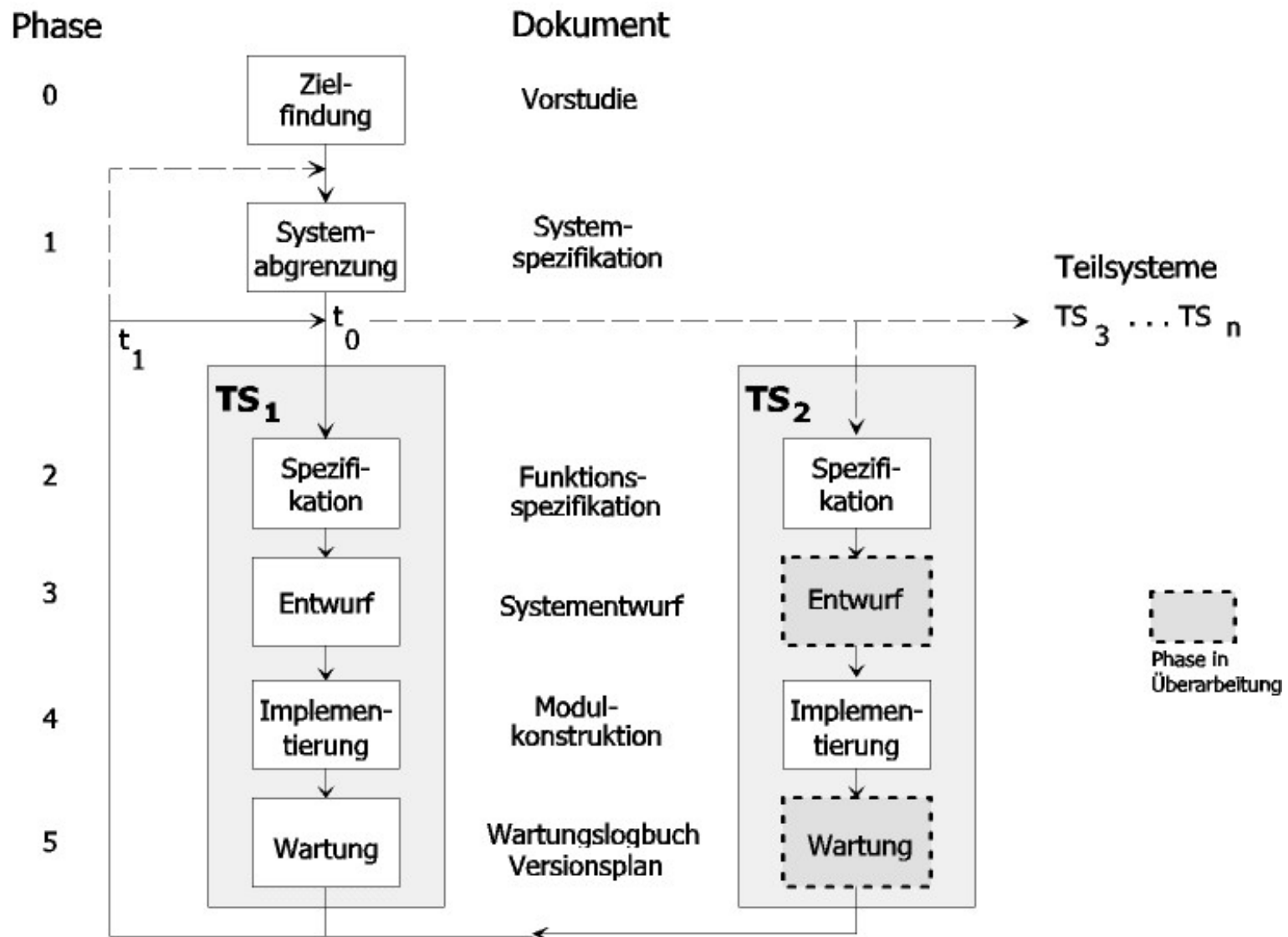
- Höherer Entwicklungsaufwand durch zusätzliche Herstellung von Prototypen .
- Gefahr der Integration eines Wegwerf-Prototyps im Produkt aus Termingründen.
- Prototypen werden oft als Ersatz für die fehlende Dokumentation gesehen.
- Die Beschränkungen und Grenzen von Prototypen sind oft nicht bekannt.
- Prototypen ersetzen oft fehlende Dokumentation



Gut geeignet zur Reduzierung des Entwicklungsrisikos!.

Das nebenläufige Modell

Das nebenläufige Modell (Schema)



Das nebenläufige Modell (Eigenschaften)

- Parallelisierung von sequentiell organisierten Vorgängen (Wasserfallmodell für einzelne Teilprodukte)
- Minimierung des Improvisierens und „*trial and errors*“
- Förderung der Zusammenarbeit der einzelnen Personengruppen
- Reduzierung von Wartezeiten und Zeitverzögerungen

Das nebenläufige Modell (Bewertung)

Vorteile:

- + Optimale Zeitausnutzung
- + Frühes Erkennen und Vermeiden von Problemen durch Beteiligung aller betroffenen Personengruppen

Nachteile:

- Hoher Personal- und Planungsaufwand
- Ehrgeiziges Ziel: „*Right the first time*“
- Risiko, grundlegende Entscheidungen zu spät zu treffen



Ziel: Auslieferung des vollständigen Produkts!

Übersicht Vorgehensmodelle

Prozessmodell	Primäres Ziel	Antreibendes Moment	Benutzerbeteiligung	Eigenschaften
Wasserfall-Modell	Minimaler Managementaufwand	Dokumente	gering	Sequentiell, volle Breite
V-Modell	Maximale Qualität	Dokumente	gering	Sequentiell, volle Breite, Validation, Verifikation
Spiralmodell	Risikominimierung	Risiko	mittel	Entscheidung pro Zyklus über weiteres Vorgehen
Prototypen-Modell	Risikominimierung	Code	hoch	Nur Teilsysteme
Nebenläufiges Modell	Minimale Entwicklungszeit	Zeit	hoch	volle Breite, nebenläufig

Das objektorientierte Modell

Das objektorientierte Modell

Ziel und Vorteil einer OO-Entwicklung: Wiederverwendbarkeit!

Wiederverwendungsebenen:

- Ebene der OOA-Modelle,
- Ebene der technischen Entwürfe
- Ebene der implementierten Klassen und Klassenbibliotheken

Wiederverwendungsgebiete:

- Anwendungs- bzw. branchenspezifische Klassen und Subsysteme
- Klassen, die die Anbindung einer Anwendung an die Umgebung ermöglichen
- Klassen, die die Anbindung an die Systemsoftware ermöglichen

Das objektorientierte Modell (Wiederverwendbarkeit (2))

Herkunft der Klassen :

- frühere Entwürfe,
- auf dem Markt eingekaufte Klassenbibliotheken

Zeitpunkte des Einsatzes wiederverwendbarer Klassen:

- während der laufenden Entwicklung,
- am Ende der laufenden Entwicklung
- nach einer nachträglichen Analyse abgeschlossener Entwicklungen



Starker *bottom-up*-Aspekt aufgrund der Wiederverwendbarkeit

Das objektorientierte Modell (Eigenschaften)

Eigenschaften:

- Tendenz zur Entwicklung in voller Breite.
- Fokus auf Wiederverwendung.
- Gut kombinierbar mit dem evolutionären, dem inkrementellen und dem Prototypen-Modell.

Das objektorientierte Modell (Bewertung)

Vorteile:

- + Verbesserung der Produktivität und Qualität;
- + Konzentration auf die eigenen Stärken, den Rest zukaufen.

Nachteile:

- Nur voll nutzbar, wenn OO-Methoden eingesetzt werden.
- Geeignete Infrastruktur (Wiederverwendungsarchiv) muss vorhanden sein.
- Firmenkultur der Wiederverwendung muss aufgebaut sein.
- Technische Probleme müssen überwunden werden.

Agile Softwareentwicklung

Agile Softwareentwicklung

(Quelle: Gabler Wirtschaftslexikon)

Beispiele:

Crystal, eXtreme Programming, Scrum, Feature Driven Development.

Ziele:

- Transparenz und Flexibilität im SWE-Prozess erhöhen
- schnellerer Einsatz der entwickelten Systeme
- Risiken im Entwicklungsprozess minimieren

Kernidee:

- Teilprozesse möglichst einfach und somit beweglich (=agil) halten

=> Manifest für agile Softwareentwicklung (2001, 4 Werte und 12 Thesen)

Agile Softwareentwicklung

Manifest für agile Softwareentwicklung (2001)

4 Werte:

1. Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge
2. Funktionierende Software ist wichtiger als umfassende Dokumentation
3. Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlung
4. Reagieren auf Veränderung ist wichtiger als das Befolgen eines Plans

Zwar notwendig:

formale Grundlagen (standardisierte Prozesse, Dokumentation, vorgegebene Rahmen und Handlungsanweisungen durch Verträge)

Jedoch mindestens ebenso wichtig:

„weiche Kriterien“ (Kommunikation, Rücksichtnahme auf Beteiligte und flexibles Agieren)

Agile Softwareentwicklung

Manifest für agile Softwareentwicklung (2001)

12 Thesen:

1. Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen
2. Heiße Anforderungsänderungen sind 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 Entwicklungsteam zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.

Agile Softwareentwicklung

Manifest für agile Softwareentwicklung (2001)

12 Thesen (Forts.):

7. Funktionierende Software ist das wichtigste Fortschrittsmaß.
8. Agile Prozesse fördern nachhaltige Entwicklung. 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.“

Agile Softwareentwicklung

Bewertung:

Wie bei jedem Vorgehensmodell muss auch hier gut geplant, analysiert und entworfen werden!

Analyse und Entwurf erfolgen auf Modul- und Komponentenebene und im Laufe des Projekts

Vorteile:

- + Einfach einzuführen und zu kontrollieren, wenig Administration, wenige Dokumente
- + Beteiligung des Auftraggebers/Anwenders frühzeitig und laufend
- + Schnelle Auslieferung von Funktionen (Früher Prototyp kann bereits Wertschöpfung für Auftraggeber bedeuten)
- + Hohe Kommunikation der Beteiligten (Auftraggeber und Entwickler)
- + Tests werden meist sehr gut integriert (je nach Methodik)
- + Für kleinere (wenige Personenjahre) Projekte sehr gut geeignet

Agile Softwareentwicklung

Nachteile:

- Passendes Entwicklerteam erforderlich (Fähigkeiten und Motivationen)
- (Zu) große Teams provozieren zu lange Besprechungszeiten
 - => Teambuilding-Maßnahmen sehr sinnvoll,
 - Aufteilung in kleinere Gruppen
- Weniger geeignet für sehr kleine, sehr große, sicherheitskritische oder bereits vertraute Projekte (Aufwand).
- Einzelne wichtige Phasen wie QM, Betrieb und Wartung werden teilweise wenig beachtet (je nach Methodik)
- Einbeziehung externer Ressourcen unter Vertragsgesichtspunkten schwer
- Mehraufwand durch missverständliche Kommunikation oder verspätete Entscheidungen leichter möglich

Agile Methoden im Vergleich:

<http://www.computerwoche.de/a/agile-methoden-im-vergleich,2352712>

Agile Softwareentwicklung:

Scrum

Scrum

Was ist Scrum?

- aus dem Englischen für „[das] Gedränge“ (beim Rugby → gemeinsam erfolgreich)
- seit Mitte der 1990er Jahre bekannt
- **3 Rollen** für direkt am Entwicklungsprozess beteiligte:
 - **Product Owner** (stellt fachliche Anforderungen und priorisiert sie),
 - **Scrum Master** (managt den Prozess und beseitigt Hindernisse) und
 - **Team** (3-9 Personen, entwickelt das Produkt).

Daneben gibt es die **Stakeholders** (Kunden, Anwender, Management ...)

- Entwicklung ist Backlog-getrieben (Backlog => Liste von Anforderungen)
- Inkrementelle (*Milestones*) und iterative (*Sprint*) Vorgehensweise

Scrum

Sprint

- *Time-Box* (Arbeitsabschnitt) für die Implementation eines *Increment of Potentially Shippable Functionality* (Entwicklungseinheit) durch das Scrum-Team
- Länge: meist 30 Kalendertage
- Beginnt mit dem *Sprint Planning Meeting* (8 Std., Arbeitspakete schnüren)
- Endet mit dem *Sprint Review Meeting* (Ergebnispräsentation mit Feedback)
- *Daily Scrum Meetings* (15 Min., Abstimmung und Informationsaustausch)
- Während des Sprints:
keine Unterbrechung des Teams durch neue oder geänderte Anforderungen.
→ Kontinuität und konzentriertes Arbeiten auf das vorgegebene Ziel hin.
- Nicht-Team-Mitglieder (insbesondere der *Product Owner* u.a. *Stakeholders*) stehen während des Sprints für Rückfragen zur Verfügung.
- In seltenen Fällen: *Sprint Cancellation* durch *Product Owner*.

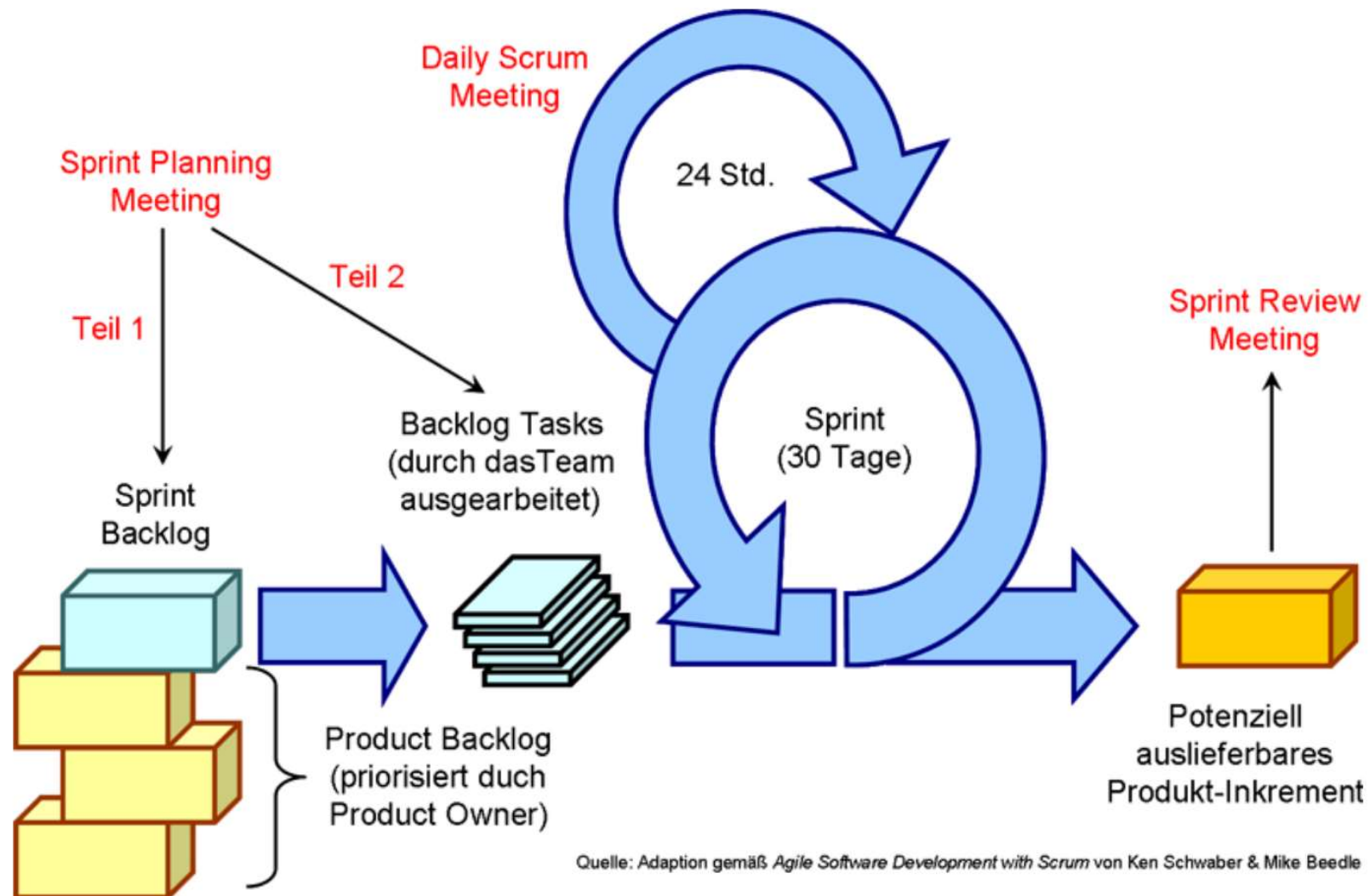
Scrum

Sprint Planning

beantwortet die folgenden Fragen:

- Was ist in dem Produktinkrement des kommenden Sprints enthalten?
- Was kann in diesem Sprint fertiggestellt werden?
- Wie wird die für die Lieferung des Produktinkrements erforderliche Arbeit erledigt?

Scrum



Literatur

1. Helmut Balzert,
Lehrbuch der Softwaretechnik Band 2, Spektrum Akademischer Verlag, Heidelberg, Berlin,
1998, ISBN 3-8274-0065-1
2. <http://www.vorgehensmodelle.de/>,
Gesellschaft für Informatik, Fachgruppe WI-VM: „Vorgehensmodelle für die betriebliche
Anwendungsentwicklung“
3. <http://www.kbst.bund.de>,
Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der
Bundesverwaltung (Innenministerium)
4. Barry W. Boehm: *"A Spiral Model of Software Development and Enhancement"*, Computer
No. 5 Vol. 21 1988, S.61-72
5. Agile Methoden im Vergleich: <http://www.computerwoche.de/a/agile-methoden-im-vergleich,2352712>
6. Alexander Kriegisch, Agiles Projektmanagement, <https://scrum-master.de>
7. Ken Schwaber und Jeff Sutherland: Der Scrum Guide,
<https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-German.pdf>