Software Life Cycle

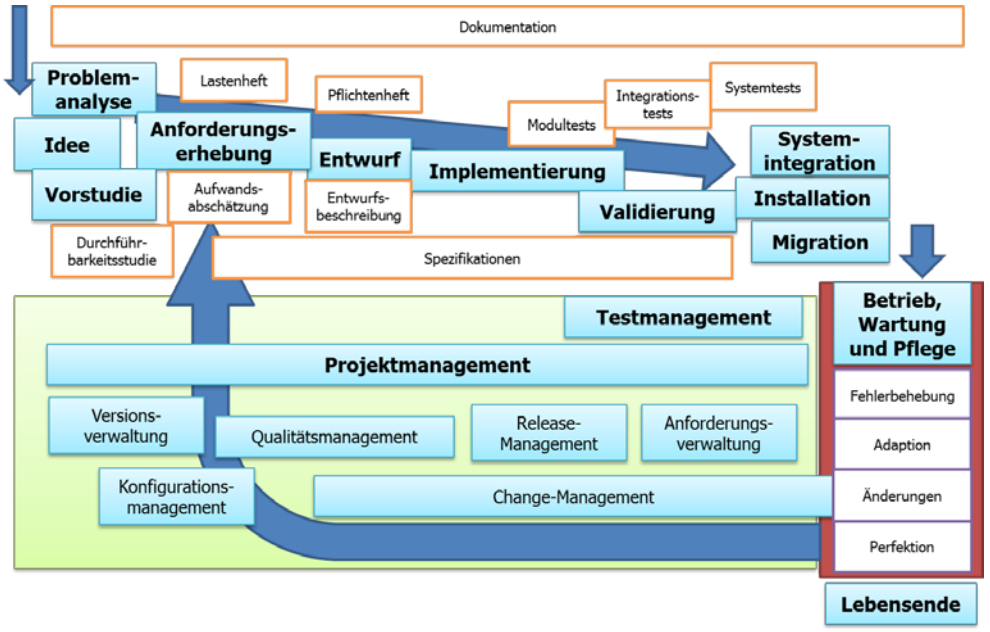
Eine Übersicht über den Lebenszyklus von Software und Vorgehensmodellen

Inhalt

Phasen der Softwareentwicklung

Überlegt man sich die Phasen einer Software-Entwicklung im Detail, so folgen diese eigentlich dem gesunden Menschenverstand:

- Zuerst muss bekannt sein, wofür eigentlich entwickelt werden soll, was das Ziel eines Entwicklungsprojekts ist
- Wenn die Ziele bekannt sind ist es erforderlich, die Anforderungen zu ermitteln und zu dokumentieren, also „was entwickelt werden soll“
- Wenn man weiß, was entwickelt werden soll, wird die Aufgabenstellung in ein Design oder in eine Architektur zerlegt – wie kann die Aufgabenstellung unter den gegebenen Bedingungen und Anforderungen auf verschiedene Module, auf verschiedene Systeme etc. am besten aufgeteilt werden
- Nun erfolgt erst die eigentliche Entwicklung, also die Umsetzung der Anforderungen unter Berücksichtigung der Architektur und der Ziele
- Ist die Entwicklung fertig, muss das Ergebnis getestet werden, sprich eine Prüfung vorgenommen werden, ob das Software-Produkt dem entspricht, was eigentlich benötigt wurde
- Eine fertige Software muss installiert werden, Benutzer müssen geschult werden, eventuell müssen auch vorhandene Daten übernommen werden
- Ist die Software in Betrieb ergeben sich mit hoher Wahrscheinlichkeit Änderungen in Folge von Fehlerbehebungen, Wünschen und Bedürfnissen von Benutzern oder durch Änderungen in den Rahmenbedingungen, beispielsweise bei Änderungen von Gesetzen
- Wird die Software nicht mehr benötigt oder gibt es beispielsweise die dafür erforderliche Hardware nicht mehr, ist die Software am Lebensende angelangt



Life Cycle

Diese Phasen nennt man den Life Cycle einer Software. Obige Aufzählungen lassen sich grob in die Teile Entwicklung und Pflege/Wartung einteilen.

Die Entwicklungsphase lässt sich wiederum in die Phasen Analyse (Ziele, Anforderungen), Entwurf (Design/Architektur), Implementierung (Software-Programmierung) und Test (Überprüfung des Verhaltens) unterscheiden.

Die Wartungsphase lässt sich in die Phasen Installation (Ausbringung des Systems zur Verwendung), eventueller Migration (Übernahme bestehender Daten und Verfahren), Änderungen (durch Wünsche, durch Änderung der Anforderungen oder zwecks Fehlerbehebung) sowie dem Lebensende (Deinstallation, Vernichtung etc.) unterteilen.

Phasen

Diese Phasen des Software-Engineerings kann man auch wie folgt sehen:

Phase 1- Planung:

- Begleitendes Projektmanagement
- Erstellung des Lastenhefts, d.h. der Liste der Anforderungen = „WAS wird benötigt“
- Erstellung des Pflichtenhefts, d.h. der präziseren Beschreibung, WIE die Anforderungen umgesetzt werden
- Eine Aufwandsabschätzung zur Projektplanung und zur Kalkulation eines Angebots

Phase 2 – Analyse:

- Analyse der Anforderungen im Detail
- Analyse von verwendeten und zugehörigen Daten
- Analyse der vorgegebenen oder notwendigen Prozesse
- Analyse des vorhandenen oder gewünschten Systems
- Strukturierte Analyse oder objektorientierte Analyse der Aufgabenstellung

Phase 3 – Entwurf:

- Entwurf der Softwarearchitektur – Hardware, Netzwerk, Module etc.
- Strukturiertes Design oder objektorientiertes Design

Anmerkung: bisher wurde keine einzige Zeile programmiert!

Phase 4 – Programmierung:

- Strukturierte Programmierung oder objektorientierte Programmierung (OOP)

Phase 5 – Test:

- Modultests – einzelne Programmteile werden auf deren Korrektheit und Funktionsfähigkeit überprüft
- Integrationstests – das Zusammenspiel mehrerer Module – auch von verschiedenen Programmierern – wird getestet
- Systemtests - die Funktion eines Teilsystems oder des gesamten Systems wird überprüft
- Akzeptanztests – das Gesamtsystem wird einer Abnahmeprozedur unterzogen

Phase 6 – Projektmanagement:

- Incident und Problem Management – wie wird mit ungeplanten Ereignissen umgegangen
- Change Management – wie werden Änderungen im Entwicklungsverlauf sauber dokumentiert und implementiert
- Release Management – wann kann welcher Entwicklungszustand eines Softwareprojekts zur Benutzung freigegeben werden, wie werden Versionen verwaltet
- Configuration Management – wie werden einzelne Varianten der Software und deren Abhängigkeiten gesteuert

Phase 7 – begleitendes Qualitätsmanagement:

- Softwareergonomie – ist die Software gut bedienbar und entspricht sie dem Arbeitsablauf der Benutzer?
- Softwaremetrik – werden vereinbarte Reaktionszeiten z.B. auf Suchabfragen eingehalten?

Phase 8 – Dokumentation:

- Systemdokumentation zur Weiterentwicklung und Fehlerbehebung – Dokumentation der Entwicklungsschritte von den Anforderungen über das Design bis zur Programmierung
- Betriebsdokumentation für den Betreiber und für das Service von Anwendungen, z.B. für den Wiederanlauf nach Ausfall einer Hardware
- Bedienungsanleitung für die Anwender
- Verfahrensdokumentation zur Beschreibung rechtlich relevanter Softwareprozesse

Hinzu kommen die Phasen, die in der Wartung des Lebenszyklus notwendig sind. Diese führen oft in wiederum eine – wenn auch meist kürzere – Entwicklungsphase, da die meisten Schritte bei Änderungen oder Fehlerbehebungen nochmals durchlaufen werden müssen. Beispielsweise müssen die Anforderungen der Änderungen erhoben werden, das Design gehört auf Tauglichkeit überprüft, eine entsprechende Entwicklung ist durchzuführen, das Ergebnis muss getestet werden und Handbücher müssen ergänzt werden usw.

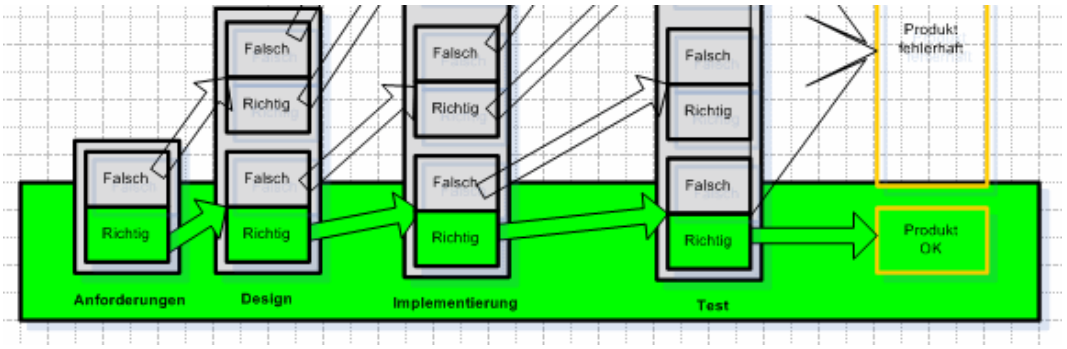
Aus obiger Darstellung ergibt sich, dass Softwareentwicklung ein komplexer Prozess ist und sich aus mehr Teilbereichen als „nur der Programmierung“ zusammensetzt. Die reinen Programmierzeiten eines Entwicklungsprojekts betragen in der Regel etwa 10...20% der gesamten Projektdauer.

Anforderungen

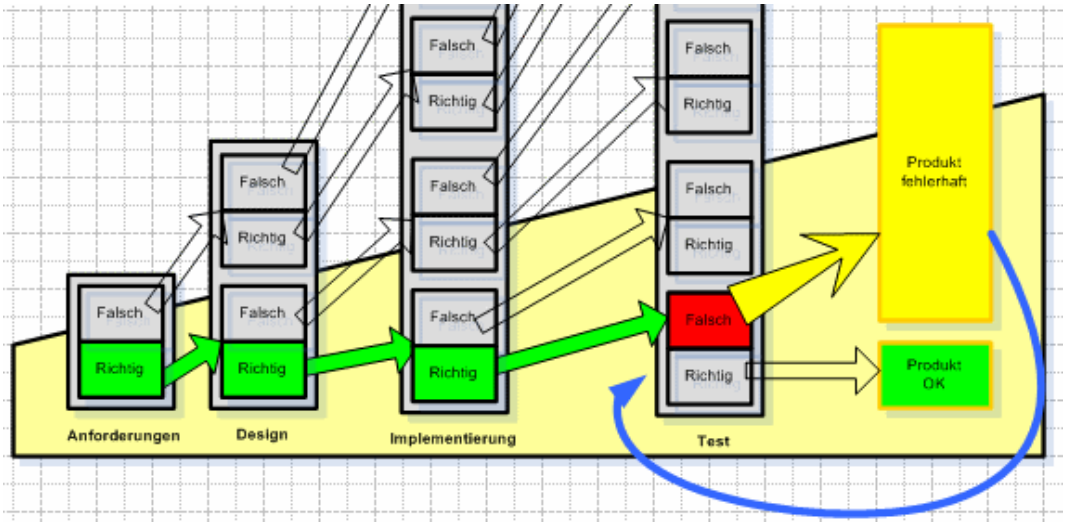
Ein wesentlicher Teil der Zeit wird – besser gesagt sollte – für die Analysephase aufgewendet (werden). In dieser Phase werden die meisten Fehler mit teils immenser Auswirkung auf Qualität, Zeit und Kosten verursacht, während Fehler in der Testphase „nur mehr“ gefunden werden können. Folgende Bilder sollen das verdeutlichen:

Die Entwicklungsphase lässt sich in vier grobe Teile unterteilen – Analyse/Anforderungen, Entwurf/Design, Implementierung und Test. In jeder dieser Phasen kann alles richtig gemacht werden oder es entstehen Fehler durch Missverständnisse, fehlerhafte Umsetzung oder durch Unvollständigkeit etc.

Die einzige Möglichkeit, zu einem „fehlerfreien“ Produkt zu kommen ist es, in allen Phasen alles richtig zu machen:

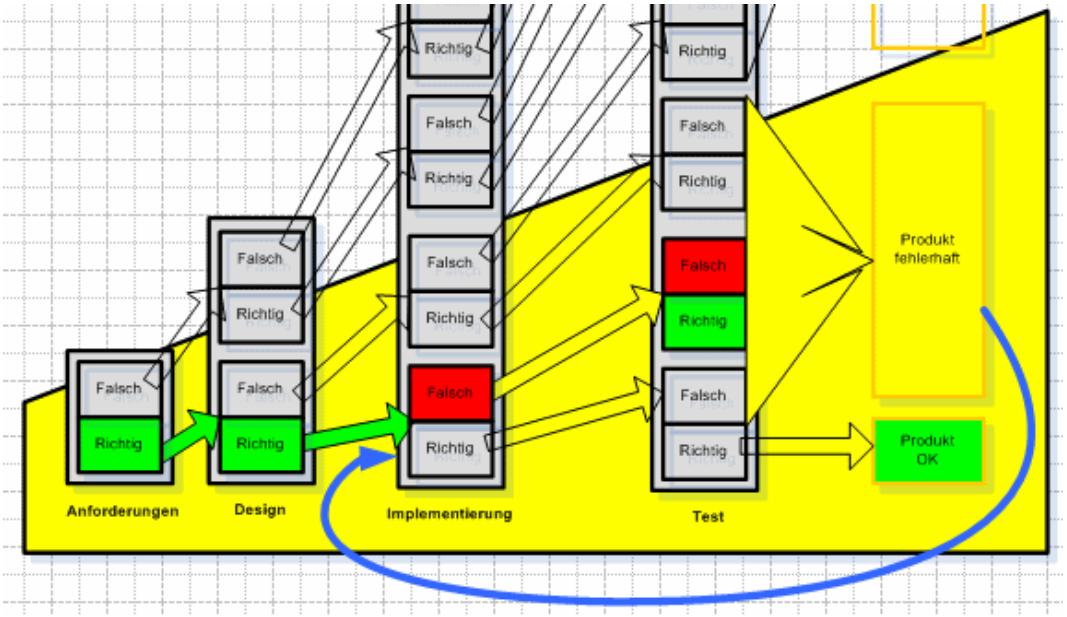


Werden nun Fehler „nur“ in der Testphase gemacht, führt dies zu einem zumindest scheinbar fehlerhaften Produkt:



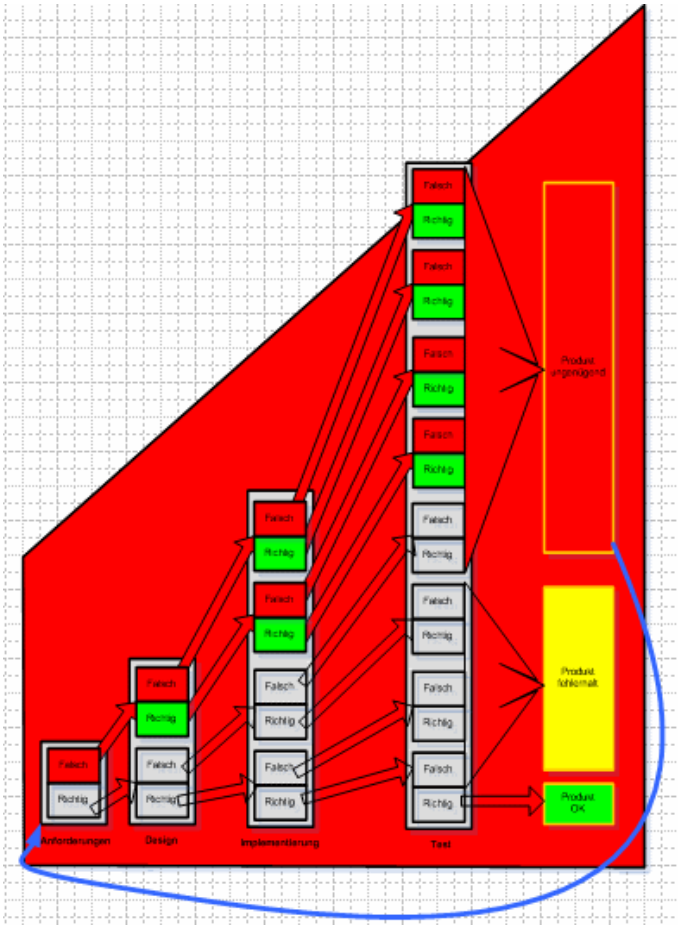
Die Korrektur ist vergleichsweise einfach – es müssen nur jene Tests wiederholt werden, die fehlerhaft durchgeführt wurden oder fehlerhafte Ergebnisse geliefert haben.

Fehler in der Implementierungsphase führen hingegen bereits unweigerlich zu einem fehlerhaften Produkt, unabhängig davon, ob in der nachfolgenden Testphase alles richtig gemacht wird oder vielleicht dort auch Fehler entstehen:



Konsequenz ist: Wiederholung der fehlerhaften Entwicklungsschritte plus notwendiger, unter Umständen aller zugehörigen, Tests.

Dramatisch werden Fehler in der Anforderungsphase. Wird hier etwas falsch verstanden oder sind nicht alle Anforderungen bekannt führt dies mit Sicherheit zu einem unbrauchbaren Produkt:

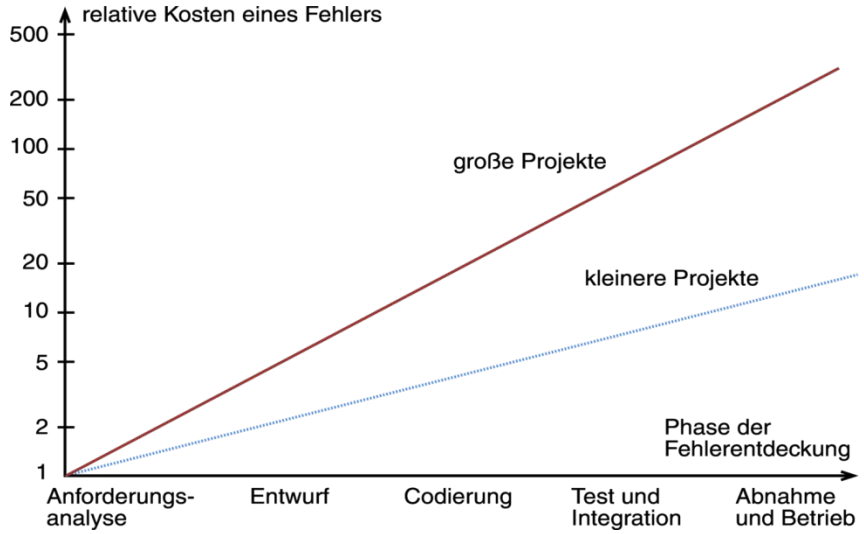


Konsequenz ist: zurück an den Start und nochmaliges Durchlaufen des teilweisen oder vollständigen Entwicklungszyklusses – beispielsweise können übersehene Anforderungen Einfluss auf das Design des Systems haben, wodurch sich umfangreiche Änderungen ergeben können.

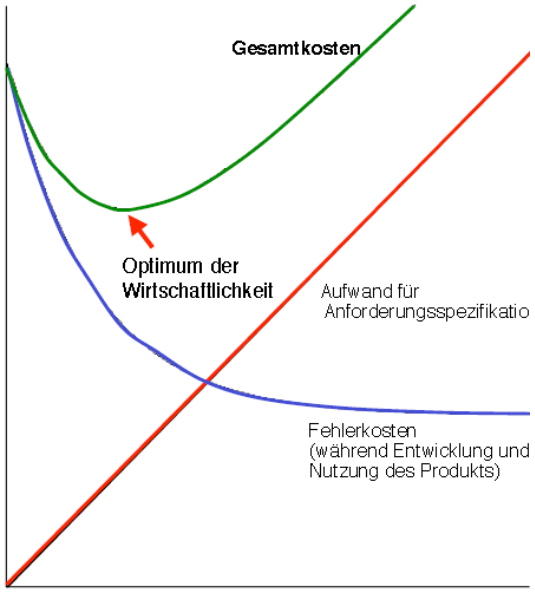
Die obige Darstellung zeigt zwei Dinge:

- Es ist essentiell wichtig, von Beginn an alles möglichst richtig zu machen. Auch in der heutigen Zeit hat sich diese Erkenntnis noch nicht vollständig durchgesetzt und führt dazu, dass die Phasen der Anforderungen und des Designs mit den daraus resultierenden Konsequenzen oft vernachlässigt werden. Requirements Engineering und Business Analyse sind in den letzten Jahren zu zwei eigenständigen Disziplinen herangewachsen, um diesen Problemen zu begegnen

- Die Kosten für Fehlerbehebungen steigen exponentiell an – je später ein Fehler entdeckt wird, desto teurer wird dessen Behebung. Man spricht von 100- bis 1000-fachen Kosten, wenn beispielsweise Fehler erst nach der Auslieferung anstelle in der Analysephase entdeckt werden



Gleichzeitig ist es nicht wirtschaftlich, beliebig viel Aufwand in die Analysephase zu stecken, da – ähnlich wie in der Testphase – ab einem gewissen Punkt der Nutzen nur mehr marginal im Vergleich zum Aufwand steigt:



Vorgehensmodelle

Nachdem die oben dargestellten Zusammenhänge erkannt wurden, sind in etwa ab 1970 laufend verschiedene Modelle entwickelt worden, um die Vorgehensweise in der Software-Entwicklung zu professionalisieren und deren Kosten zu minimieren.

Software-Entwicklung ist seit Beginn an auch zunehmend komplexer geworden, insbesondere was den Umfang von Software betrifft. Die Entwicklung von Klein-Software unterscheidet sich fundamental von der Entwicklung größerer Software, wobei letzteres mittlerweile den Normalfall darstellt.

Viele Probleme existieren für Klein-Software gar nicht. Die Erfahrung, dass die Entwicklung kleiner Programme recht einfach ist, ist der Hauptgrund für den Trugschluss, dass Software-Entwicklung generell etwas Einfaches sein müsse. Dem ist jedoch nicht so: Der Aufwand für die Erstellung von Software steigt mit wachsender Produktgröße überproportional an, was auch empirisch erhärtet ist.

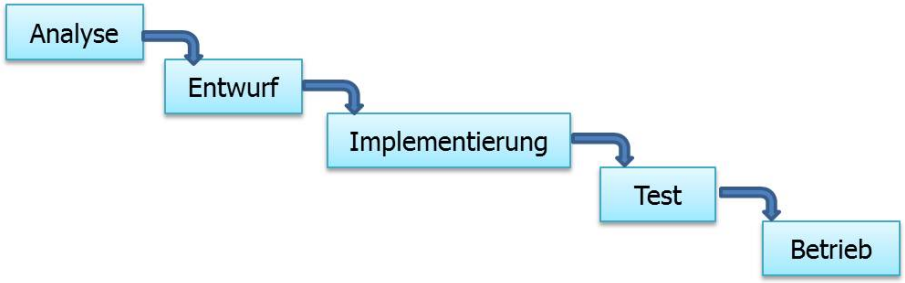
Ursache für dieses exponentielle Wachstum ist unter anderem, dass die Aufwände zur Beherrschung der wachsenden Komplexität und der Kommunikationsaufwand überproportional wachsen. Das Wachstum einzelner Faktoren ist dabei nicht kontinuierlich, sondern weist Quantensprünge auf, weil sich die Anzahl der Kommunikationspfade vervielfacht:

Anzahl beteiligter Personen	1	2	3	4	5
					
Anzahl von Kommunikationspfaden	0	1	3	6	10

Ab zwei beteiligten Personen wird eine Kommunikation überhaupt erforderlich. Ab vier beteiligten Personen übersteigt die Anzahl der Kommunikationspfade die Anzahl der Personen.

Wasserfallmodell

Eines der ersten und einfachsten Vorgehensmodelle war das sogenannte Wasserfallmodell. Es handelt sich dabei um eine lineare Abfolge der einzelnen Lebenszyklus-Phasen:

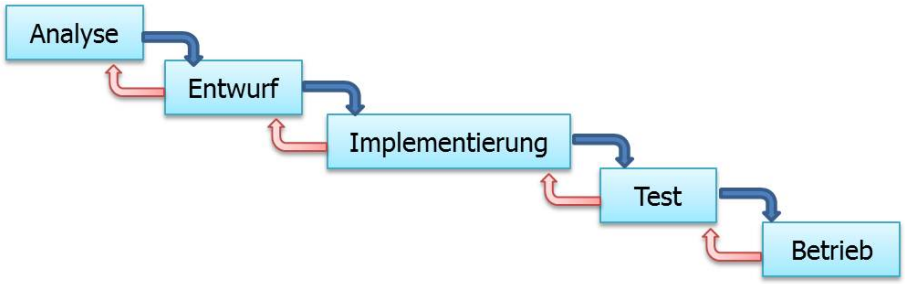


Das reine Wasserfallmodell ist für kleine Softwareprojekte durchaus geeignet. Für größere Projekte ergeben sich jedoch erhebliche Nachteile:

- Eine Überprüfung des Ergebnisses ist erst am Ende vorgesehen – nicht vermeidbare Fehler in einer oder mehreren Phasen führen zu einem nochmaligen Durchlauf kompletter Abschnitte
- Es wird davon ausgegangen, dass bereits zu Beginn alle Anforderungen feststehen und vor allem, dass sich diese während der Entwicklungsphase nicht mehr ändern, was meist nicht der Realität entspricht

Verbesserungen des Modells haben dazu geführt, dass

- Nach jeder Phase Prüfschritte durchgeführt werden um sicherzustellen, dass eine Phase erst verlassen wird, wenn alle notwendigen Ergebnisse in geprüfter und akzeptierter Form vorliegen
- Iterationen zwischen Phasen möglich sind, also für Probleme, die in der vorhergehenden Phase entstanden sind, durch Rückkehr in jene Phase behoben werden können
- Rückkopplungen auch über mehrere oder alle Phasen hinweg möglich sind



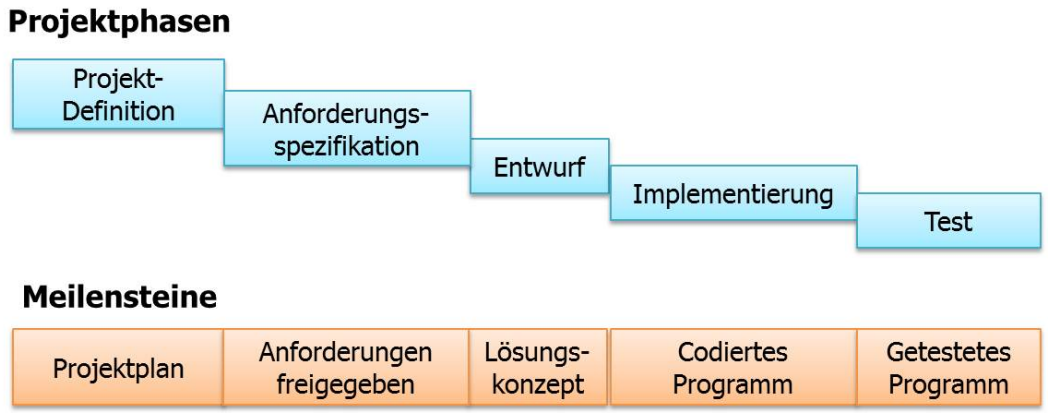
Es gibt eine große Zahl von Varianten des ursprünglichen Wasserfall-Modells, welche sich meist geringfügig in Benennung und Reihenfolge der Phasen sowie in den Möglichkeiten der Rückkopplung unterscheiden. Allen diesen Modellen ist jedoch gemeinsam, dass die Unterteilung der Entwicklung in Phasen nach einer linearen Reihenfolge von Tätigkeiten erfolgt.

Aufgrund der Software-Evolution und aufgrund der Tatsache, dass die erkannten Fehler nicht immer nur in der gegenwärtigen und der vorangegangenen Phase gemacht wurden, sind bei der Anwendung des Wasserfall-Modells häufig Iterationen über mehrere Phasen hinweg erforderlich. Dies erschwert die Projektführung und hat dem Wasserfallmodell sehr viel Kritik eingetragen. Trotz seiner unbestreitbaren Vorteile (Einfachheit, Modellierung des natürlichen Lebenslaufs) und seiner immer noch beachtlichen Verbreitung sollte das Wasserfallmodell heute nur mehr für kleine Software-Projekte eingesetzt werden. Gleiches gilt für alle ähnlichen Modelle, welche die Phasen als Tätigkeiten auffassen.

Ergebnisorientierte Phasenmodelle

Ergebnisorientierte Phasenmodelle machen wie das Wasserfallmodell den Software-Lebenslauf zur Grundlage des Prozesses. Sie gehen also ebenfalls davon aus, dass zunächst alle Anforderungen bekannt und dokumentiert sind, dann das System vollständig entworfen wird, als nächstes codiert und getestet wird usw.

In einem ergebnisorientierten Phasenmodell ist eine Phase jedoch keine Tätigkeit wie beim Wasserfall-Modell, sondern ein Zeitintervall. Die Phasen sind nach dem Ergebnis oder nach der vorherrschenden Tätigkeit, die zum Ergebnis führt, benannt. Es gibt keine Phasen-Iteration. In jeder Phase können bzw. müssen notwendige Nacharbeiten an in früheren Phasen erzielten Ergebnissen durchgeführt werden.



Jeder Phasenabschluss bildet einen Meilenstein in der Entwicklung. Die geforderten Ergebnisse der frühen Phasen sind typisch Dokumente; im weiteren Verlauf bestehen die Ergebnisse aus Code und Dokumenten.

Vorteile solcher Modelle sind

- Dass sie leicht verständlich sind und dem natürlichen Life Cycle folgen
- Dass sie ein planvolles Vorgehen mit sorgfältiger Spezifikation und Dokumentation fördern
- Und dass sie damit wesentlich dazu beitragen, Fehler möglichst früh erkennen zu können und damit Zeit und Kosten sparen

Wesentliche Nachteile bleiben jedoch aufrecht:

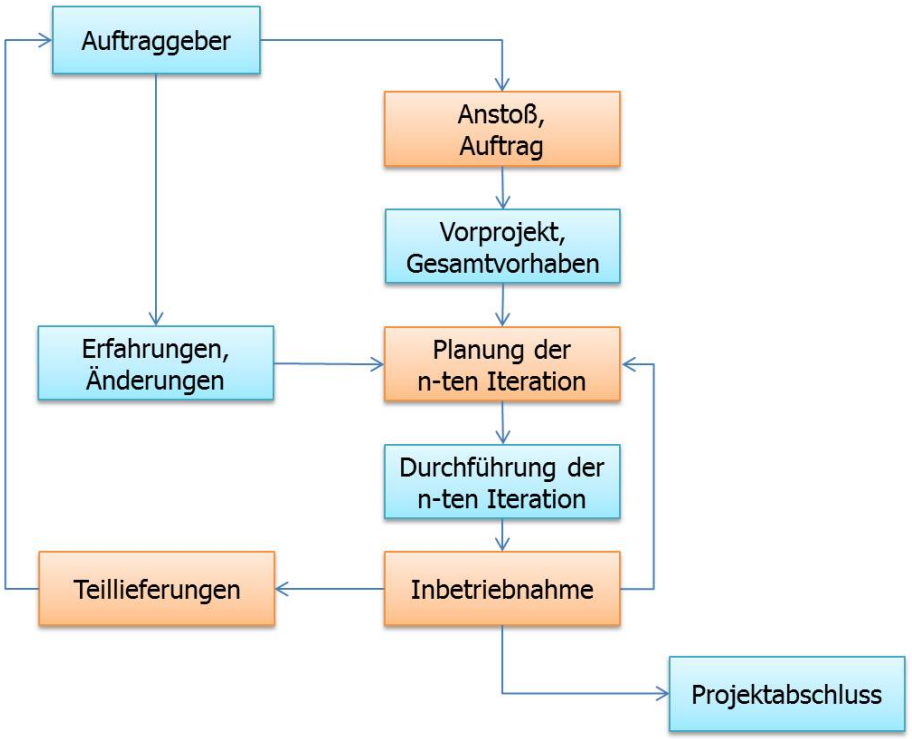
- Die Grundannahme, dass ganze Systeme genauso wie Einzelkomponenten planvoll in einer Abfolge von Schritten konstruiert werden können, ist vor allem für große Systeme mit langer Entwicklungs- und Lebensdauer meistens falsch. Solche Systeme sind in der Regel nicht von Beginn an konstruierbar, sondern sie wachsen evolutionär
- Lauffähige Systemteile entstehen erst relativ spät: die lange Dauer, in der nur Dokumentation produziert wird, ist nachteilig für die Motivation der Projektbeteiligten.
- Das System muss in einem Schritt komplett in Betrieb genommen werden.

Somit eignen sich auch ergebnisorientierte Phasenmodelle eher für Projekte, die klein sind, wo die Aufgaben gut bekannt und beschrieben sind und wo das Risiko nicht sehr groß ist.

Wachstumsmodelle

Wachstumsmodelle stellen den Gedanken der Software-Evolution in den Mittelpunkt. Ein System wird nicht von Beginn an vollständig konstruiert, sondern es wächst in einer Reihe aufeinanderfolgender Schritte. Es gibt verschiedene Varianten von Wachstumsmodellen unter verschiedenen Namen, z.B. Versionen-Modell, evolutionäres Modell oder inkrementelles Modell.

Der Grundgedanke ist, dass - ausgehend von einer groben Zusammenstellung der Gesamtfunktionalität des gewünschten Systems - eine Folge von Lieferungen mit Lieferumfang und Liefertermin definiert wird. Jede Lieferung ist betriebsfähig und kann nach Auslieferung sofort in Betrieb genommen werden. Erfahrungen mit frühen Lieferungen können bei späteren Lieferungen berücksichtigt werden:

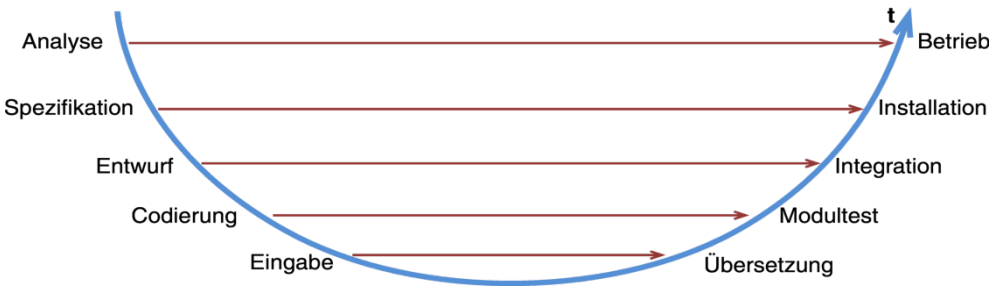


Jede Lieferung wird als mehr oder weniger autonomes Teilprojekt organisiert. Da diese Teilprojekte in der Regel vergleichsweise klein und kurz sind, kann innerhalb einer Iteration ein ergebnisorientiertes Phasenmodell als Vorgehensmodell verwendet werden.

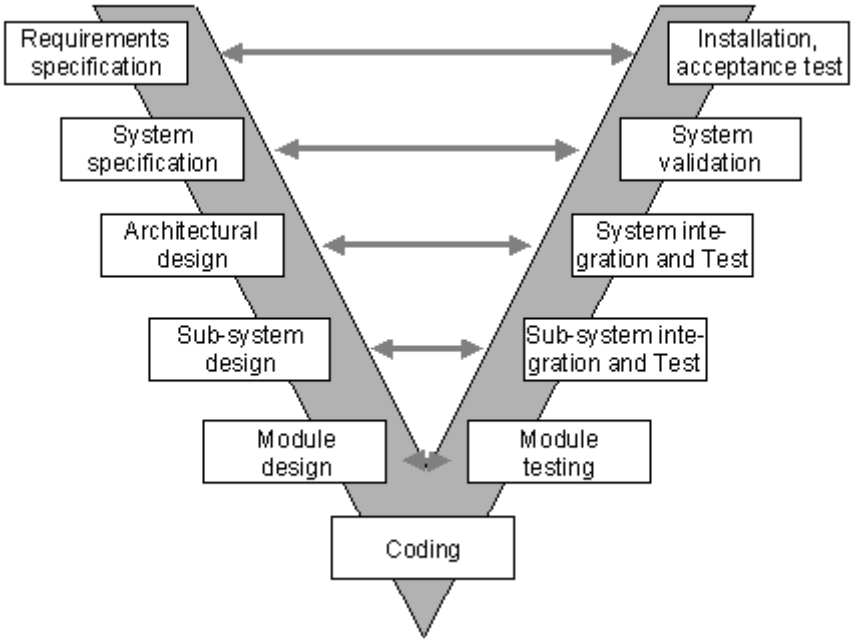
Bei der Definition des Umfangs und der Reihenfolge der Lieferungen wird immer ein schrittweiser Ausbau des Systems bis zum gewünschten Endzustand geplant. Dabei können verschiedene Merkmale schrittweise ausgebaut werden, zum Beispiel die Funktionalität, der Bedienungskomfort, die Leistung oder die Verwendbarkeit. In Fällen, wo der Auftraggeber keine Teillieferungen will bzw. wo man nicht mit Teilprodukten auf den Markt gehen kann, kann ein Wachstumsmodell trotzdem sinnvoll eingesetzt werden. Die Lieferungen erfolgen dann intern. Vorteile sind eine bessere Projektkontrolle und lieferfähige Teile, wenn der geplante Liefertermin für das Gesamtsystem nicht eingehalten werden kann.

V-Modell

Ausgehend von der Erkenntnis, dass Fehler typisch auf derselben Abstraktionsebene entdeckt werden, wo diese entstehen, hat sich das sogenannte V-Modell durchgesetzt und weit verbreitet.



Daraus ergibt sich das V-Modell typisch wie folgt:



Beim Vorgehen nach dem V-Modell wird ebenso schrittweise wie in anderen Modellen vorgegangen. Das Modell wird von links oben – beginnend mit der Anforderungsspezifikation - nach unten über die Implementierung (engl.: Coding) bis nach rechts oben zur Inbetriebnahme durchlaufen.

Die Querverbindungen zeigen an, auf welchen Ebenen Tests durchgeführt werden. Modul-Tests werden gegenüber den Modulspezifikationen durchgeführt. Systemtests beinhalten bereits einzeln getestete (Modultests) sowie im Zusammenspiel getestete (Integrationstests) Teile. Am Ende wird mit einem Akzeptanztest die gesamte Funktion des Systems gegenüber den ursprünglichen Anforderungen validiert.

Die aktuelle Version des V-Modells wird als V-Modell XT bezeichnet. Die Bezeichnung XT bedeutet „eXtreme Tailoring“ und bezeichnet damit eine flexible Vorgangsweise, die eine Anpassung des Modells an die jeweils benötigten Randbedingungen – also beispielsweise der Projektgröße – ermöglicht.

Das V-Modell ist ein Leitfaden zum Planen und Durchführen von Projekten. Mit einer V-Modell-konformen Durchführung eines Projekts werden die folgenden Ziele verfolgt:

Minimierung der Projektrisiken: Das V-Modell gibt standardisierte Vorgehensweisen vor, beschreibt die zugehörigen Ergebnisse und die verantwortlichen Rollen. Damit erhöht das V-Modell die Projekttransparenz und verbessert die Planbarkeit von Projekten...

Verbesserung und Gewährleistung der Qualität: Als standardisiertes Vorgehensmodell stellt das V-Modell sicher, dass die zu liefernden Ergebnisse vollständig und von gewünschter Qualität sind. Die durch das Modell definierten Zwischenergebnisse können auf diese Weise frühzeitig überprüft werden...

Eindämmung der Gesamtkosten über den gesamten Projekt- und Systemlebenszyklus: Durch die Anwendung des standardisierten Vorgehensmodells lasst sich der Aufwand für die Entwicklung, die Herstellung, den Betrieb und die Pflege und Wartung eines Systems auf transparente Weise kalkulieren, abschätzen und steuern...

Verbesserung der Kommunikation zwischen allen Beteiligten: Die standardisierte und einheitliche Beschreibung aller relevanten Bestandteile und Begrifflichkeiten ist die Basis des wechselseitigen Verständnisses aller Projektbeteiligten...

(Auszug aus V-Modell XT Gesamt Version 1.4).

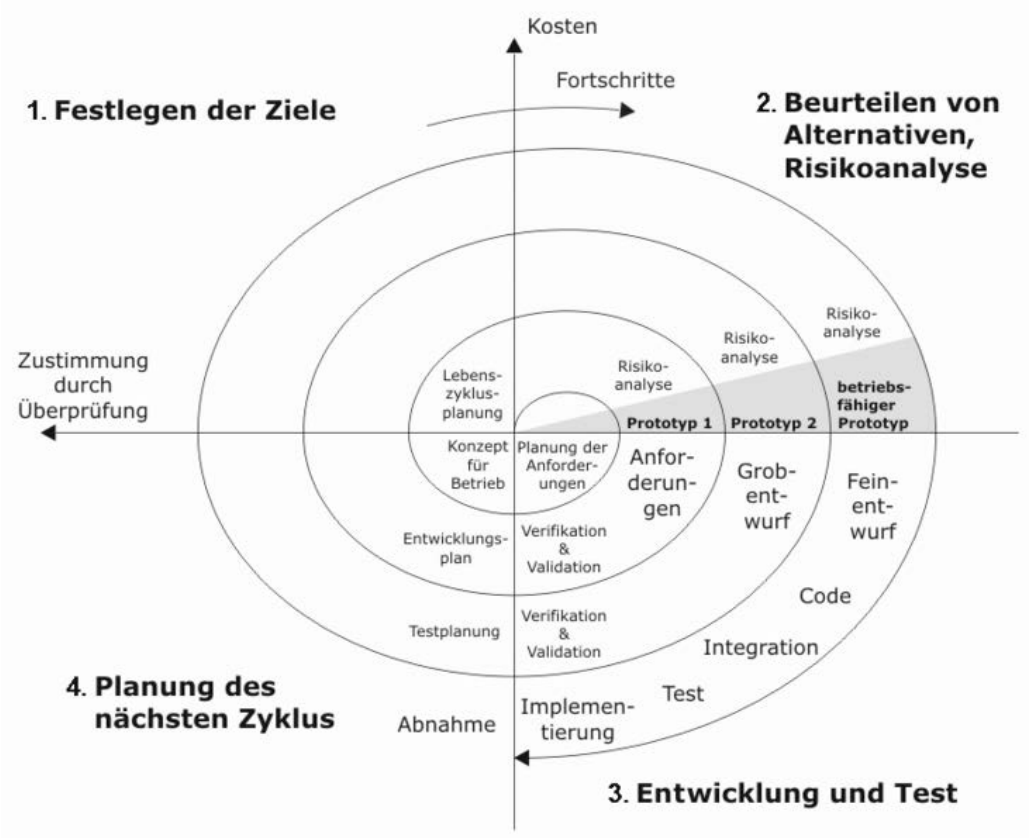
Interessant und wichtig ist, dass in diesem Modell auch Rollen mit deren Aufgaben und Verantwortungen beschrieben sind.

Das V-Modell kann in Teilen auch iterativ durchschritten werden oder anders formuliert: es dient als Basis für andere Vorgehensweisen. Beispielsweise werden in agilen Vorgehensmodellen (siehe unten) die Phasen Anforderungen, Entwurf, Entwicklung bis hin zur Auslieferung lauffähiger Software im Sinne eines Wachstumsmodells mehrfach durchlaufen.

Spiralmodell

Das Spiralmodell ist eine Weiterentwicklung des Wasserfallmodells. Es war vor allem für die Entwicklung großer, risikoreicher Systeme gedacht. Anstelle der zyklischen Abfolge "Entwickeln, prüfen, entwickeln, prüfen..." tritt beim Spiralmodell ein vierteiliger Zyklus:

- Planung des ersten bzw. nächsten Spiralumlaufs
- Zielsetzung und gegebenenfalls Zielkorrektur oder Zielanpassung
- Untersuchung von Varianten und ihren Risiken, Variantenentscheid
- Entwicklung und Prüfung (dieser Schritt korrespondiert mit dem Wasserfall-Modell).



Der Zyklus wird viermal durchlaufen, nämlich für Systemdefinition, Anforderungen an die Software, Konzipierung (Architektur, Entwurf) und Realisierung (Detailentwurf, Codierung, Test, Integration und Installation). Wichtig ist vor allem der dritte Schritt im Zyklus, wo die Risiken der Entwicklung erkannt und kontrolliert werden. Ein wesentliches Element hierfür sind Prototypen.

Gelegentlich findet man unter dem Namen „Spiralmodell“ auch Modelle, die eigentlich Wachstumsmodelle sind, bei denen aber jeder Wachstumsschritt als ein Spiralumlauf dargestellt ist.

Agile Vorgehensmodelle

Agile Vorgehensmodelle sind etwa um das Jahr 2000 herum entstanden. Grund war die Unzufriedenheit einiger Autoren, dass trotz dem Einsatz von Software-Engineering-Methoden die Ergebnisse von Entwicklungsprojekten immer noch nicht befriedigend waren, sprich Zeit- und Kostenüberschreitungen mit sich brachten. Das Ergebnis ist das Agile Manifest mit folgendem Inhalt:

Manifest für Agile Softwareentwicklung

Wir erschließen bessere Wege, Software zu entwickeln,
indem wir es selbst tun und anderen dabei helfen.
Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

Individuen und Interaktionen mehr als Prozesse und Werkzeuge
Funktionierende Software mehr als umfassende Dokumentation
Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
Reagieren auf Veränderung mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden,
schätzen wir die Werte auf der linken Seite höher ein (22)

Agil bedeutet laut Duden „flink, wendig, beweglich, rege“. Der agile Gedanken hat den Grundsatz „so wenig wie möglich, so viel wie nötig“, der leider auch oft missverstanden und missbraucht wurde, nämlich indem einige Personen die falsche Meinung vertraten, deshalb beispielsweise keine Dokumentationen erstellen zu müssen.

Im Laufe der Zeit haben sich verschiedenste Richtungen und Schwerpunkte ausgebildet wie etwa XP (eXtreme Programming), Scrum, Crystal Family, Test driven development (TDD) und dergleichen mehr.

Der agilen Idee liegt eine adaptive Vorgangsweise ähnlich den Wachstumsmodellen zugrunde:

- Projekt laufend planen
- Projekt führen und laufend überwachen
    - Kontinuierliches Risikomanagement und Qualitätsmanagement durchführen
    - Auf Veränderungen und Anpassungen rasch reagieren
    - Auf Kundenzufriedenheit und motivierte Teams besonders achten
    - Angemessen handeln
- Projekt abschließen

Agile Projekte

- Achten mehr auf motivierte Teams als auf perfekte Regeln
- Legen mehr Wert auf gute Kommunikation im Team als auf formales Berichtswesen
- Achten darauf, dass ein erreichtes Ergebnis mehr zählt als die sture Abarbeitung von Aufgaben
- Akzeptieren kontinuierliche Änderungen und halten nicht an Planvorgaben fest
- Schätzen die Erfahrung wertvoller ein als Lehrbücher
- Managen aktiv die Risiken statt Krisen zu bewältigen

Anwendungsgebiete agiler Methoden sind:

- Software-Entwicklung für kleine und mittlere Projekte
- Projekte mit kurzer Time to Market
- Projekte ohne sicherheitskritische Elemente
- Änderungsanfällige Projekte
- Projekte mit reifen Teilnehmern (Team, Kunde)

Zusammenfassung

Kein Vorgehensmodell ist für alle Aufgabenstellungen geeignet. Die Evolution über die Jahrzehnte hat zahlreiche Verbesserungen und gute Ideen hervorgebracht. Es bleibt jedoch nicht erspart, für jede Situation das am besten geeignete Modell einzusetzen und unter Umständen auch anzupassen.

Somit frei nach Dr. Peter Hruschka (Atlantic Systems Guild):

- Eine Methode alleine genügt nicht, die Kenntnis über mehrere muss verfügbar sein
- Haben Sie auch den Mut, etwas wegzulassen, wenn es zu einer Situation nicht passt
- Setzen Sie jeweils die gerade ausreichende und zur Größenordnung und zum Risiko des Projekts passende Methode ein
- Das Ergebnis ist wichtiger als der Weg dorthin