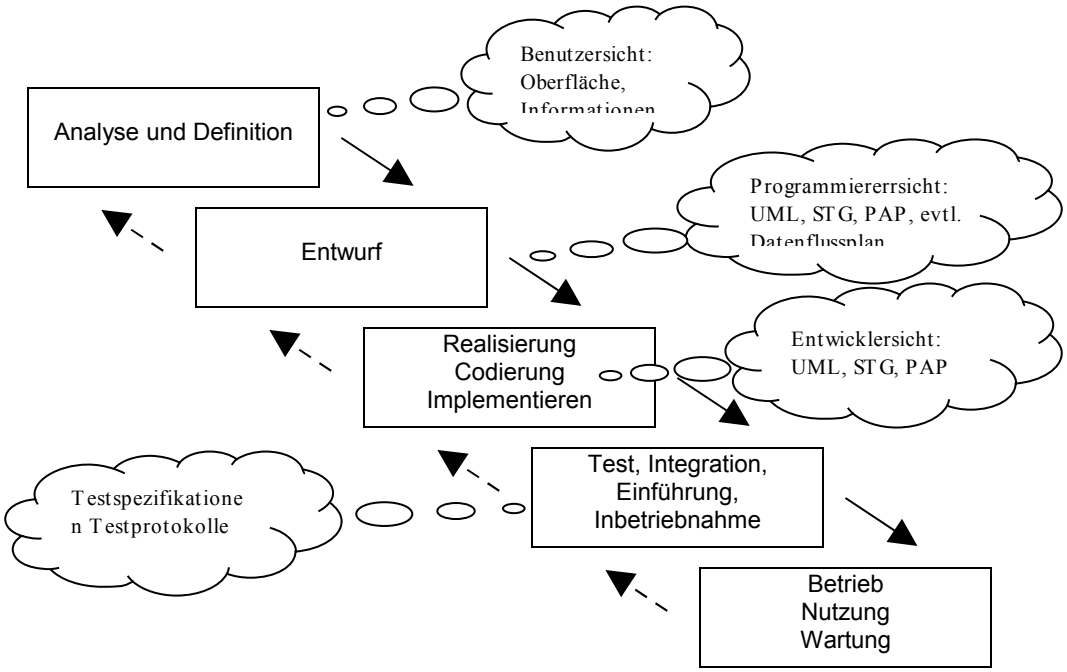
# Wasserfallmodell

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Das Wasserfallmodell ist ein lineares (nicht-iteratives) Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in Phasen organisiert wird. Dabei gehen die Phasenergebnisse wie bei einem Wasserfall immer als bindende Vorgaben für die nächst tiefere Phase ein.

Im Wasserfallmodell hat jede Phase vordefinierte Start- und Endpunkte mit eindeutig definierten Ergebnissen. In Meilensteinsitzungen am jeweiligen Phasenende werden die Ergebnisdokumente verabschiedet. Zu den wichtigsten Dokumenten zählen dabei das Lastenheft sowie das Pflichtenheft. In der betrieblichen Praxis gibt es viele Varianten des reinen Modells. Es ist aber das traditionell am weitesten verbreitete Vorgehensmodell.

Der Name „Wasserfall“ kommt von der häufig gewählten grafischen Darstellung der fünf bis sechs als Kaskade angeordneten Phasen.




*Ein Wasserfallmodell mit Rücksprungmöglichkeiten (gestrichelt)*

Erweiterungen des einfachen Modells (Wasserfallmodell mit Rücksprung) führen iterative Aspekte ein und erlauben ein schrittweises „Aufwärtslaufen“ der Kaskade, sofern in der aktuellen Phase etwas schieflaufen sollte, um den Fehler auf der nächsthöheren Stufe beheben zu können.

# Phasen

1. Anforderungsanalyse und -spezifikation (engl. *Requirement analysis and specification*)
2. Systemdesign und -spezifikation (engl. *System design and specification*)
3. Programmierung und Modultests (engl. *Coding and module testing*)
4. Integrations- und Systemtest (engl. *Integration and system testing*)
5. Auslieferung, Einsatz und Wartung (engl. *Delivery, deployment and maintenance*)

# Eigenschaften

1. Jede Aktivität ist in der vorgegebenen Reihenfolge und in der vollen Breite vollständig durchzuführen.
2. Am Ende jeder Aktivität steht ein fertiggestelltes Dokument, d.h. das Wasserfall-Modell ist ein "dokument-getriebenes" Modell.
3. Der Entwicklungsablauf ist sequentiell, d.h. jede Aktivität muss beendet sein, bevor die nächste anfängt.
4. Es orientiert sich am sogenannten top-down-Verfahren.
5. Es ist einfach, verständlich und benötigt nur wenig Managementaufwand.
6. Eine Benutzerbeteiligung ist nur in der Anfangsphase vorgesehen, anschließend erfolgen der Entwurf und die Implementierung ohne Beteiligung des Benutzers bzw. Auftraggebers. Weitere Änderungen stellen danach Neuaufträge dar.

# Vorteile

- klare Abgrenzung der Phasen
- einfache Möglichkeiten der Planung und Kontrolle
- bei stabilen Anforderungen und klarer Abschätzung von Kosten und Umfang sehr effektives Modell

# Nachteile

- Das Modell ist nur auf einfache Projekte anwendbar
- Unflexibel gegenüber Änderungen und im Vorgehen (Phasen müssen sequenziell abgearbeitet werden)
- Frühes Festschreiben der Anforderungen ist sehr problematisch → eventuell teure Änderungen (mehrmalig wiederholtes Durchlaufen des Prozesses bei Änderungen)
- Einführung des Systems sehr spät nach Beginn des Entwicklungszyklus → später *return on investment*
- Fehler werden unter Umständen erst spät erkannt und müssen mit erheblichem Aufwand entfernt werden

Da es schwierig ist, bereits zu Projektbeginn alles endgültig und im Detail festzulegen, besteht das Risiko, dass die letztendlich fertiggestellte Software nicht den tatsächlichen Anforderungen entspricht. Um dem zu begegnen, wird oftmals ein unverhältnismäßig hoher Aufwand in der Analyse- und Konzeptionsphase betrieben. Zudem erlaubt das Wasserfallmodell nicht bzw. nur sehr eingeschränkt, im Laufe des Projekts Änderungen aufzunehmen. Die fertiggestellte Software bildet folglich nicht den aktuellen, sondern den Anforderungsstand zu Projektbeginn wieder. Da größere Softwareprojekte meist auch eine sehr lange Laufzeit haben, kann es vorkommen, dass eine neue Software bereits zum Zeitpunkt ihrer Einführung inhaltlich veraltet ist.

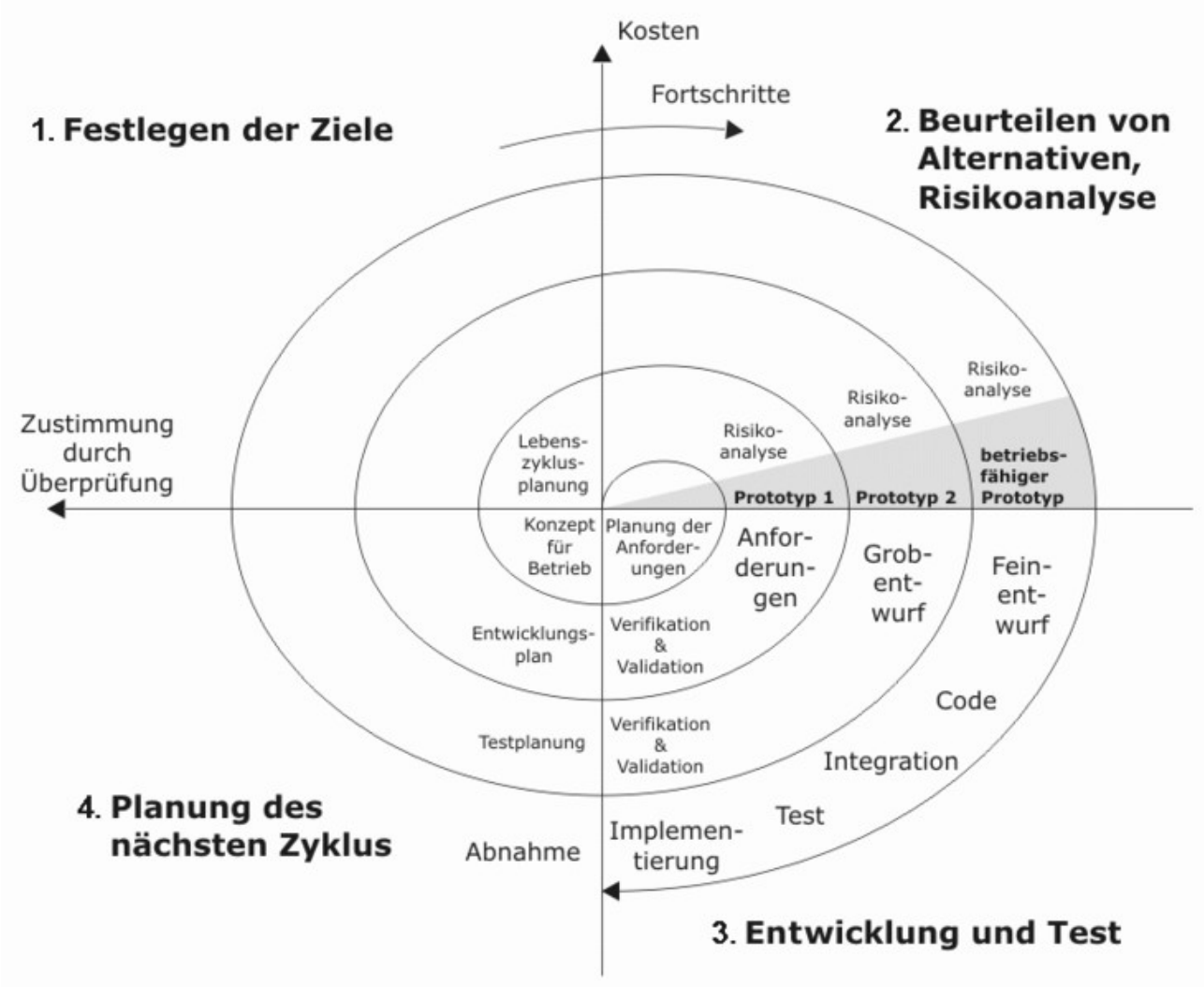
# Spiralmodell

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Das Spiralmodell ist ein Vorgehensmodell in der Softwareentwicklung, das im Jahr 1988 von Barry W. Boehm beschrieben wurde. Es ist ein generisches Vorgehensmodell und daher offen für bereits existierende Vorgehensmodelle. Das Management kann immer wieder eingreifen, da man sich spiralförmig voran entwickelt.

Das Spiralmodell fasst den Entwicklungsprozess im Software-Engineering als iterativen Prozess auf, wobei jeder Zyklus in den einzelnen Quadranten folgende Aktivitäten enthält:

1. Festlegung von Zielen, Identifikation von Alternativen und Beschreibung von Rahmenbedingungen
2. Evaluierung der Alternativen und das Erkennen, Abschätzen und Reduzieren von Risiken
3. Realisierung und Überprüfung des Zwischenprodukts
4. Planung des nächsten Zyklus der Projektfortsetzung.



Am Ende jeder Windung der Spirale steht ein Betrachten des Projektfortschritts (engl. *Review*). Dabei wird auch der Projektfortgang geplant und verabschiedet.

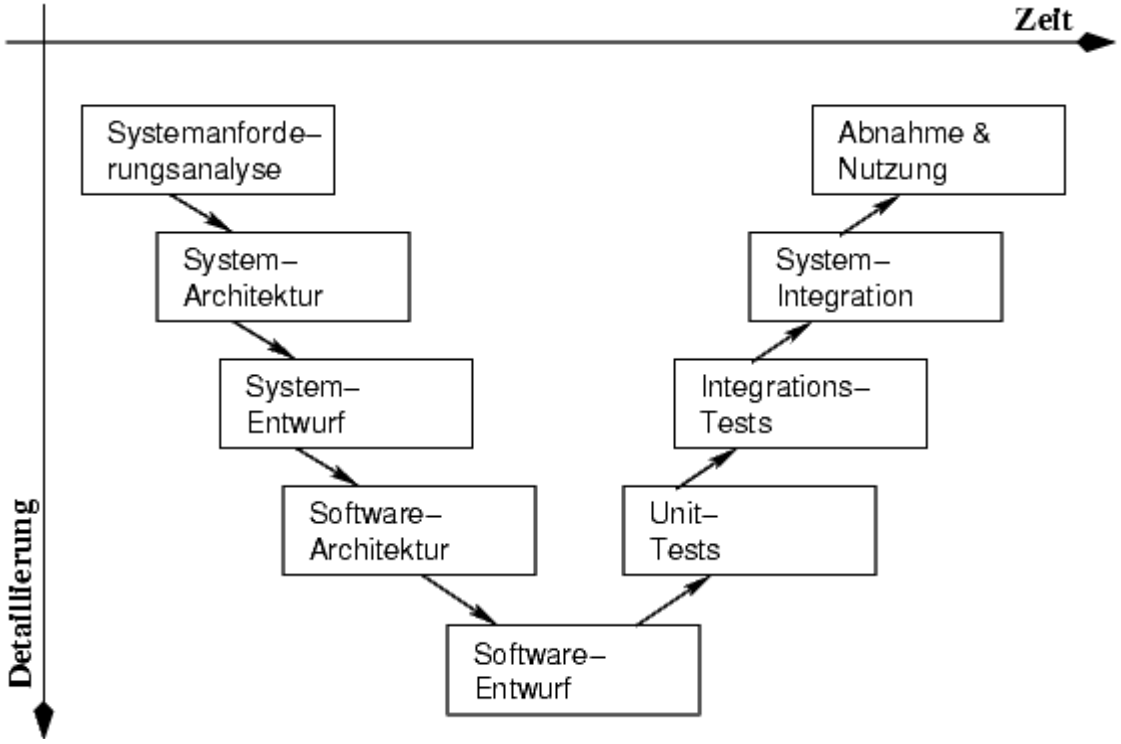
Das Spiralmodell gehört zu den inkrementellen oder iterativen Vorgehensmodellen. Es ist eine Weiterentwicklung des Wasserfallmodells, in der die Phasen mehrfach spiralförmig durchlaufen werden.

Das inkrementelle und iterative Vorgehensmodell sieht daher eine zyklische Wiederholung der einzelnen Phasen vor. Dabei nähert sich das Projekt langsam den Zielen an, auch wenn sich die Ziele während des Projektfortschrittes verändern. Durch das Spiralmodell wird nach Boehm das Risiko eines Scheiterns bei großen Softwareprojekten entscheidend verringert.

# V-Modell

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Das V-Modell ist eine abstrakte, umfassende Projektmanagement-Struktur für die IT-Systementwicklung. Sein Name bezieht sich auf die V-förmige Darstellung der Projektelemente wie IT-Systemdefinitionen und Tests, gegliedert nach ihrer groben zeitlichen Position und ihrer Detailtiefe (siehe Abbildung).




Phasen des V-Modells über Zeit und Detaillierung

Im Gegensatz zu einem klassischen Phasenmodell werden im V-Modell lediglich Aktivitäten und Ergebnisse definiert und keine strikte zeitliche Abfolge gefordert. Insbesondere fehlen die typischen Abnahmen, die ein Phasenende definieren. Dennoch ist es möglich, die Aktivitäten des V-Modells z.B. auf ein Wasserfallmodell oder ein Spiralmodell abzubilden.

Das V-Modell fasst eine Reihe von ähnlich gelagerten Tätigkeiten zu einem so genannten Vorgehensbaustein zusammen. Einige dieser Vorgehensbausteine finden bei allen Projekten Anwendung und werden daher als V-Modell-Kern bezeichnet. Dazu gehören:

1. **PM**: Projektmanagement
2. **QS**: Qualitätssicherung
3. **KM**: Konfigurationsmanagement
4. **PA**: Problem- und Änderungsmanagement

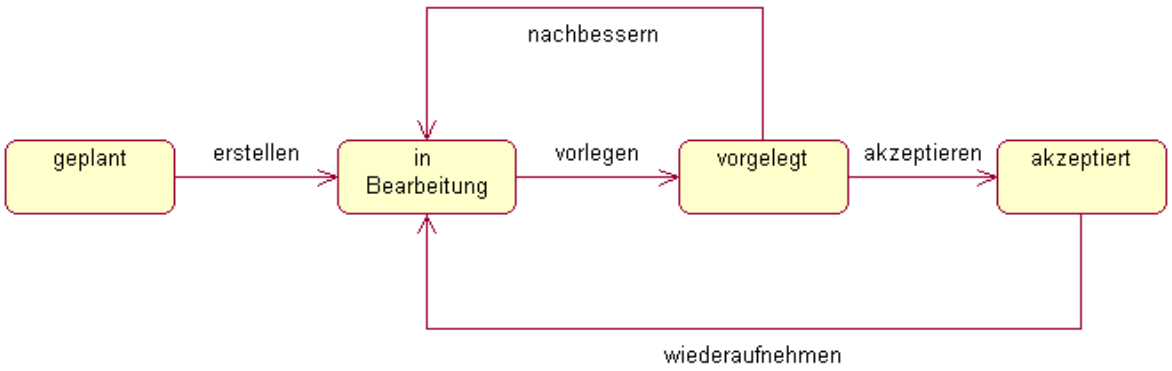
## Produkte und Aktivitäten

Das V-Modell definiert eine Reihe von Dokumenten, die als *Produkte* bezeichnet werden. Diese setzen sich aus einzelnen *Themen* zusammen. Produkte, die einen starken inhaltlichen Zusammenhang haben, werden wiederum derselben *Produktgruppe* zugeordnet.

Jedes definierte Produkt durchläuft vier Zustände:

1. geplant
2. in Bearbeitung
3. vorgelegt
4. akzeptiert

wobei folgende Übergänge zwischen diesen Zuständen möglich sind:




Tätigkeiten, die *Produkte* verändern, bezeichnet man als *Aktivitäten*; diese sind ihrerseits aus einzelnen *Teilaktivitäten* zusammengesetzt, die dann jeweils genau ein *Thema* behandeln. Inhaltlich verwandte *Aktivitäten* werden dabei wiederum zu *Aktivitätengruppen* zusammengefasst. Zu jeder Aktivität ist genau hinterlegt, welche *Produkte* sie benötigt bzw. verändert und welche Arbeitsschritte notwendig sind, um die gewünschte Modifikation herbeizuführen. Zu diesem Zweck ist jeder Aktivität ein *Produktfluss* und eine *Abwicklung* definiert. Während der Produktfluss beschreibt, aus welchen Aktivitäten die benötigten Eingabeprodukte mit welchem Zustand kommen, um dann in modifizierter Form bzw. modifiziertem Zustand an eine nachfolgende Aktivität weitergereicht zu werden, beinhaltet die Abwicklung genauere Anweisungen zur Durchführung der Aktivität.

Die zeitliche Abfolge der Aktivitäten ergibt sich somit aus der Verfügbarkeit der benötigten (Teil-)Produkte in einem bestimmten Zustand.

# Feature Driven Development (FDD)

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Feature Driven Development (Abk. FDD) ist eine Sammlung von Arbeitstechniken, Strukturen, Rollen und Methoden für das Projektmanagement im Rahmen agiler Softwareentwicklung.

# Grundlagen

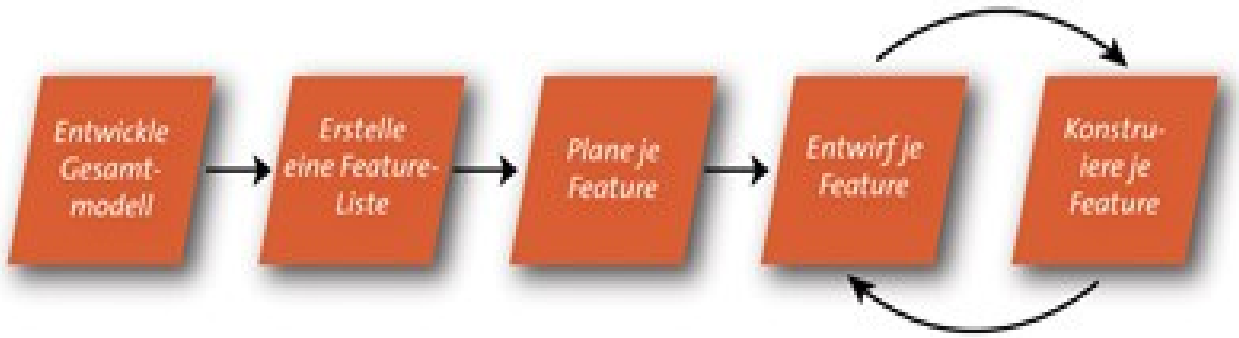
FDD stellt den Feature-Begriff in den Mittelpunkt der Entwicklung. Jedes Feature stellt einen Mehrwert für den Kunden dar. Die Entwicklung wird anhand eines Feature-Plans organisiert. Eine wichtige Rolle spielt der Chefarchitekt (engl. Chief Architect), der ständig den Überblick über die Gesamtarchitektur und die fachlichen Kernmodelle behält. Bei größeren Teams werden einzelne Entwicklerteams von Chefprogrammierern (engl. Chief Programmer) geführt.

FDD definiert ein Prozess- und ein Rollenmodell, die gut mit existierenden klassischen Projektstrukturen harmonieren. Daher fällt es vielen Unternehmen leichter, FDD einzuführen als XP oder Scrum. Außerdem ist FDD ganz im Sinne der agilen Methoden sehr kompakt. Es lässt sich auf 10 Seiten komplett beschreiben.

# FDD-Prozessmodell

FDD-Projekte durchlaufen fünf Prozesse.

1. Prozess #1: Entwickle ein Gesamtmodell (Rollen: alle Projektbeteiligte)
2. Prozess #2: Erstelle eine Feature-Liste (Rollen: in der Regel *(i. d. R.)* nur die Chefprogrammierer)
3. Prozess #3: Plane je Feature (Rollen: Projektleiter, Entwicklungsleiter, Chefprogrammierer)
4. Prozess #4: Entwurf je Feature (Rollen: Chefprogrammierer, Entwickler)
5. Prozess #5: Konstruiere je Feature (Rollen: Entwickler)




Die ersten drei Prozesse werden innerhalb weniger Tage durchlaufen. Die Prozesse 4 und 5 werden in ständigem Wechsel durchgeführt, weil jedes Feature in maximal zwei Wochen realisiert wird.

## Prozess #1: Entwickle ein Gesamtmodell

Im ersten Prozess definieren Fachexperten und Entwickler unter Leitung des Chefarchitekten Inhalt und Umfang des zu entwickelnden Systems. In Kleingruppen werden Fachmodelle für die einzelnen Bereiche des Systems erstellt, die in der Gruppe vorgestellt, ggf. überarbeitet und schließlich integriert werden. Das Ziel dieser ersten Phase ist ein Konsens über Inhalt und Umfang des zu entwickelnden Systems sowie das

fachliche Kernmodell.

## Prozess #2: Erstelle eine Feature-Liste

Im zweiten Prozess detaillieren die Chefprogrammierer die im ersten Prozess festgelegten Systembereiche in Features. Dazu wird ein dreistufiges Schema verwendet: Fachgebiete (engl. Subject Areas) bestehen aus Geschäftstätigkeiten (engl. Business Activities), die durch Schritte (engl. Steps) ausgeführt werden. Die Schritte entsprechen den Features. Die Features werden nach dem einfachen Schema <Aktion> <Ergebnis> <Objekt> aufgeschrieben, z.B. „Berechne Gesamtsumme der Verkäufe“. Ein Feature darf maximal zwei Wochen zu seiner Realisierung benötigen. Das Ergebnis dieses zweiten Prozesses ist eine kategorisierte Feature-Liste, deren Kategorien auf oberster Ebene von den Fachexperten aus Prozess #1 stammen.

## Prozess #3: Plane je Feature

Im dritten Prozess planen der Projektleiter, der Entwicklungsleiter und die Chefprogrammierer die Reihenfolge, in der Features realisiert werden sollen. Dabei richten sie sich nach den Abhängigkeiten zwischen den Features, der Auslastung der Programmierteams sowie der Komplexität der Features.

Auf Basis des Plans werden die Fertigstellungstermine je Geschäftsaktivität festgelegt. Jede Geschäftsaktivität bekommt einen Chefprogrammierer als Besitzer zugeordnet. Außerdem werden für die bekannten Kernklassen Entwickler als Besitzer festgelegt (engl. Class Owner List).

## Prozess #4: Entwerfe je Feature

Im vierten Prozess weisen die Chefprogrammierer die anstehenden Features Entwicklerteams auf Basis des Klassenbesitztums zu. Die Entwicklerteams erstellen ein oder mehrere Sequenzdiagramme für die Features und die Chefprogrammierer verfeinern die Klassenmodelle auf Basis der Sequenzdiagramme. Die Entwickler schreiben dann erste Klassen- und Methodenrümpfe. Schließlich werden die erstellten Ergebnisse inspiziert. Bei fachlichen Unklarheiten können die Fachexperten hinzugezogen werden.

## Prozess #5: Konstruiere je Feature

Im fünften Prozess programmieren die Entwickler die im vierten Prozess vorbereiteten Features. Bei der Programmierung werden Komponententests und Code-Inspektionen zur Qualitätssicherung eingesetzt.

# Rational Unified Process (RUP)

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Der Rational Unified Process (RUP) ist ein objektorientiertes Vorgehensmodell zur Softwareentwicklung und ein kommerzielles Produkt der Firma Rational Software, die seit 2002 Teil des IBM Konzerns ist. IBM entwickelt den RUP und die zugehörige Software weiter. Die 9. Version ist die derzeit (2006) aktuelle Version. Der RUP benutzt die Unified Modeling Language (UML) als Notationssprache. Der RUP wurde von Philippe Kruchten in seiner Urform erstmals 1996 vorgestellt.

Der Unified Process basiert auf mehreren Prinzipien

- Anwendungsfällen
- Architektur im Zentrum der Planung
- inkrementellem und iterativen Vorgehen

Eine konkrete Implementierung des Unified Process ist der Rational Unified Process. Die erste Version des RUP aus dem Jahre 1999 führte die Vorschläge dieser drei Begründer für eine einheitliche Modellierungsmethode zusammen.

# Statische Aspekte

Der RUP legt grundlegende Arbeitsschritte fest:

Kernarbeitsschritte

- Geschäftsprozessmodellierung (englisch *Business Modeling*)
- Anforderungsanalyse (englisch *Requirements*)
- Analyse & Design (englisch *Analysis & Design*)
- Implementierung (englisch *Implementation*)
- Test (englisch *Test*)
- Auslieferung (englisch *Deployment*)

Unterstützende Arbeitsschritte

- Konfigurations- und Änderungsmanagement (englisch *Configuration & Change Management*)
- Projektmanagement (englisch *Project Management*)
- Infrastruktur (englisch *Environment*)

# Dynamische Aspekte

Orthogonal dazu gibt es im RUP vier Phasen, in welchen jeder der Arbeitsschritte mehr oder weniger intensiv zur Anwendung kommt:

- Konzeptionsphase (englisch *Inception*)
- Entwurfsphase (englisch *Elaboration*)
- Konstruktionsphase (englisch *Construction*)
- Übergabephase (englisch *Transition*)

Diese Phasen sind in Iterationen unterteilt. Somit ist RUP iterativ/inkrementell. Resultate der Phasen sind die sogenannten Meilensteine (englisch *milestone*s):

- Lifecycle *objectives milestone* (Vision inklusive rudimentäres Anwendungsfallmodell (wesentliche Funktionalität), tentative/provisorische Architektur, Identifikation der wesentlichsten Risiken, Planung der Ausarbeitungsphase)
- *Lifecycle architecture milestone* (Architekturprototyp, detailliertes

Anwendungsfallmodell, Planung der Konstruktionsphase)
- *Initial operational capability milestone* (Entwurfsmodelle und Beta-Release der Software)
- *Product release milestone* (Release in Produktionsqualität)

# Scrum

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Scrum (engl. das Gedränge) ist eine Sammlung von Arbeitstechniken, Strukturen, Rollen und Methoden für das Projektmanagement im Rahmen agiler Softwareentwicklung. Es ist ein Vorgehensmodell, das wenige Festlegungen trifft. Teams bzw. Entwickler organisieren sich weitgehend selbst und wählen auch die eingesetzten Methoden. Das Vorgehen und die Methoden werden fortlaufend aktuellen Erfordernissen angepasst.

## Scrum-Rollen

Bei Scrum gibt es drei klar getrennte Rollen, die von Mitarbeitern ausgeführt werden, die beim gleichen Projekt zusammen arbeiten und damit auch das gleiche Ziel haben. Damit jeder für das, was er kann, zuständig und verantwortlich ist, werden die Zuständigkeiten wie folgt aufgeteilt:

| Rolle | Aufgabe |
|---|---|
| *Product Owner* | Der *Product Owner* legt das gemeinsame Ziel fest, das das Team erreichen muss. Er stellt das Budget zur Verfügung. Er setzt regelmäßig die Prioritäten der einzelnen Product-Backlog-Elemente. Dadurch legt er fest, welche die wichtigsten Features sind, aus denen das Entwicklungsteam eine Auswahl für den nächsten Sprint trifft. |
| *Team* | Das Team schätzt die Aufwände der einzelnen Backlog Elemente ab, und beginnt mit der Implementierung der für den nächsten Sprint machbaren Elemente. Das Team arbeitet selbstorganisiert im Rahmen einer Time Box (dem Sprint), und hat das Recht (und die Pflicht), selbst zu entscheiden, wieviele Elemente des Backlogs nach dem nächsten Sprint erreicht werden müssen, man spricht dabei von *commitments*. |
| *Scrum Master* | Der *Scrum Master* hat die Aufgabe, die Prozesse der Entwicklung und Planung durchzuführen und die Aufteilung der Rollen und Rechte zu überwachen. Er hält die Transparenz während der gesamten Entwicklung aufrecht, und fördert das Zu-Tage-Treten der bestehenden Verbesserungspotentiale. Er steht dem Team zur Seite, ist aber weder *Product Owner* noch Teil des Teams. Der *Scrum Master* sorgt mit allen Mitteln dafür, dass das Team produktiv ist, also die Arbeitsbedingungen stimmen und die Teammitglieder zufrieden sind. |

## Product Backlog

Das **Product Backlog** enthält die Features des zu entwickelnden Produkts. Vor jedem Sprint werden die Elemente des Product Backlogs neu bewertet und priorisiert, dabei können bestehende Elemente entfernt sowie neue hinzugefügt werden. Hoch priorisierte Features werden von den Entwicklern im Aufwand geschätzt und in den Sprint Backlog übernommen.

## Sprint Backlog

Das **Sprint Backlog** enthält alle Aufgaben, die notwendig sind, um das Ziel des Sprints zu erfüllen. Eine Aufgabe sollte dabei nicht länger als 16h dauern. Längere Aufgaben sollten in kurze Teilaufgaben zerlegt werden. Bei der Planung des Sprint werden nur so viele Aufgaben eingeplant, wie das Team an Kapazität aufweisen kann.

## Impediment List

In die **Impediment List** werden alle Hindernisse des Projekts eingetragen. Der Scrum-Master ist dafür zuständig, diese gemeinsam mit dem Team auszuräumen.

## Sprint

Zentrales Element von Scrum ist der Sprint. Ein Sprint bezeichnet die Umsetzung einer Iteration, Scrum schlägt ca. 30 Tage als Iterationslänge vor. Vor dem Sprint werden die Produkt-Anforderungen des Kunden in einem *Product Backlog* gesammelt. Auch technische und administrative Aufgaben werden dort aufgenommen. Das Product-Backlog muss nicht vollständig sein; es wird laufend fortgeführt. Für einen Sprint wird ein Sprint-Backlog erstellt, in diesen werden Anforderungen übernommen, die während des Sprints umgesetzt werden sollen. Die Entscheidung, welche Anforderungen umgesetzt werden, wird vom Kunden nach von ihm festgelegten Prioritäten getroffen. Zum Sprint organisiert sich das Entwicklungsteam selbst, braucht also keine detaillierten methodischen Vorschriften.

## Scrum-Meeting

An jedem Tag findet ein kurzes (maximal 15-minütiges) **Scrum-Meeting** statt.

Im Scrum-Meeting werden vom Scrum Master drei Fragen an jeden Entwickler gestellt:

- "Bist Du gestern mit dem fertig geworden, was Du Dir vorgenommen hast?"
- "Welche Aufgaben wirst Du bis zum nächsten Meeting bearbeiten?"
- "Gibt es ein Problem, das dich blockiert?"

Die Sitzung dient dem Informationsaustausch im Team. Erledigte Aufgaben und Features werden anschließend im Burndown Graph aktualisiert. Falls neue Hindernisse erkannt wurden, müssen diese vom Scrum Master bearbeitet werden. Dazu werden sie in das Impediment Backlog eingetragen.

Größere Projekte können modularisiert werden. Dann wird von den jeweiligen Leitern der Teilprojekte ein Scrum-of-Scrum-Meeting durchgeführt, bei dem die Ergebnisse der einzelnen Gruppen zusammengefasst werden.

## Review

Nach einem Sprint wird das Sprint-Ergebnis einem informellen Review durch Team und Kunden unterzogen. Dazu wird das Ergebnis des Sprints (die laufende Software) vorgeführt, eventuell werden technische Eigenschaften präsentiert. Der Kunde prüft, ob das Sprint-Ergebnis seinen Anforderungen entspricht, eventuelle Änderungen werden im Product Backlog dokumentiert.

## Retrospektive

In der Retrospektive wird die zurückliegende Sprint-Phase betrachtet. Alle Teilnehmer schreiben alle Punkte auf, welche ihnen zu den Fragen "Was war gut?" (Best practice) bzw. "Was könnte verbessert werden?" (Verbesserungspotential) einfallen. Jedes Verbesserungspotential wird priorisiert und einem Verantwortungsbereich (Team oder Organisation) zugeordnet. Alle der Organisation zugeordneten Themen werden vom Scrum Master aufgenommen und in das Impediment Backlog eingetragen. Alle teambezogenen Punkte werden in das Product Backlog aufgenommen.

# Extreme Programming (XP)

# Gemeinsam auf die Spitze treiben

*Von Mario Sixtus* | © DIE ZEIT 22.12.2003 Nr.1

*stark gekürzt, Einfügungen mit []*

Neue Software zu entwerfen ist teuer und endet häufig als Fiasko. „Extremprogrammierer“ denken und arbeiten daher in Zweierteams. Das soll Fehler verhindern und die Entwicklung beschleunigen

Das [Scheitern von ca. 40% der Software-Projekte] wollen die Verfechter des Extreme Programming nun ändern. Bislang wird die neue Methode in Deutschland von einigen hundert Programmierern praktiziert, mit steigender Tendenz. Die Extremprogrammierung soll dabei den Produktionsprozess von Software vom Kopf wieder auf die Füße stellen. „Bei XP verzichten wir ganz auf den theoretischen Überbau eines Pflichtenheftes. Wir halten die Wünsche des Kunden in Form von Storys fest“, erläutert Westphal. Eine solche Story spielt nicht in höheren, abstrakten Gedankengefilden, sondern beschreibt eine konkrete Funktionalität in der realen Welt: Eine Chipkarte wird gelesen, eine Zugangsschranke öffnet sich, eine Kreditkartennummer wird abgefragt. Diese Geschichten werden auf altmodischen Karteikarten notiert und gemeinsam mit dem Kunden nach Geschäftswert geordnet.

Das Wichtigste wird so quasi zwangsläufig zuerst erledigt und alles Nachrangige nach hinten geschoben. Sollte gegen Ende des Projekts die Zeit knapp werden, dann bleiben lediglich die unbedeutenderen Funktionen auf der Strecke. Der Kunde bekommt dann immer noch ein lauffähiges Programm, das zwar nicht alles hat, was er sich wünschte, mit dem er aber immerhin arbeiten kann.

Nach jedem Veröffentlichungszyklus, der so kurz wie möglich sein sollte, sortiert der Kunde seine Prioritäten neu. „Es ist auch schon vorgekommen, dass der Klient einzelne Karteikarten kurzerhand zerrissen hat“, erzählt Westphal, „weil er im Laufe des Prozesses gelernt hat, dass diese Eigenschaften gar nicht benötigt werden.“

Im Arbeitsraum eines Extreme-Programming-Teams herrscht ein lebendiges, permanentes Grundgemurmel. Bis auf die sprichwörtlichen Pizzaschachteln entspricht hier nichts dem Klischee vom Programmierer als blassem, kontaktscheuem Eigenbrötler, der sein Leben in selbstgewählter Einzelhaft vor Tastatur und Monitor verbringt. Noch auffälliger ist, dass an jedem Rechner zwei Personen arbeiten, die sich Tastatur, Maus und Monitor teilen. „Pair Programming“ heißt dieses Doppelspiel, bei dem die Partner sich regelmäßig mit Schreiben und Gegenlesen abwechseln. „Die Qualität des Programmcodes wird dadurch einfach besser“, sagt Jutta Eckstein. Die Münchnerin ist freie Beraterin und Prozesstrainerin für Extreme Programming. „Es handelt sich dabei um eine verschärfte Form des permanenten Reviews. Außerdem arbeitet jeder Einzelne konzentrierter, man lernt fortwährend voneinander, und es wissen immer mindestens zwei Leute über jede Stelle im Code Bescheid.“

„Für viele ist Paarprogrammierung anfangs sicherlich ein Kulturschock“, räumt Jutta Eckstein ein, „wenn sie es aber erst einmal ausprobiert haben, wollen die meisten gar nicht mehr anders arbeiten.“

Aber ist das wirklich immer nur spaßig? Der Hamburger Torsten Mumme deutet vorsichtig auch gelegentliche Schwierigkeiten an: „Pair Programming ist so, als würden Sie sich nackt hinstellen. Sie haben keine Geheimnisse mehr und können niemandem etwas vormachen. Das bedarf natürlich auch einer gewissen menschlichen Reife.“ Auch der Einstieg des 46-Jährigen in die XP-Praxis verlief nicht ganz ohne Mühen: „Als ich damit

anfing, musste ich nach 15 Jahren Berufserfahrung Programmieren wieder neu lernen.“ Das lag allerdings weniger an den Zweierbanden als vielmehr an einem Grundprinzip der XP-Logik: „Jedes Konzept wird nur ein einziges Mal ausgedrückt. Das klappt nur, wenn Sie sehr kleine Teile haben.“ Einer der Leitsätze der Extremprogrammierer lautet dann auch: *„Do it once and only once!“*

Anstatt nun solch einen Software-Teil anzufertigen und ihn anschließend auszuprobieren, dreht der XPler die Reihenfolge um: Zuerst bastelt er sich einen Test, der natürlich fehlschlägt, da er nichts zu testen hat. Erst dann wird der eigentliche Code geschrieben, dessen einzige Aufgabe es nun ist, diese Probe zu bestehen. Zuerst kommt der Rahmen, dann das Bild. „Der Test sagt mir vor allen Dingen auch, wann ich fertig bin“, erklärt Mumme, „und verhindert, dass ich mir mehr Arbeit mache, als nötig ist.“

Die Geschichte vom ersten XP-Praxiseinsatz beginnt mit einem schiffbrüchig gewordenem Großprojekt: 1996 wurde Kent Beck, einer der XP-Väter, vom Autohersteller Chrysler um Unterstützung gebeten, da die Neukonstruktion des hauseigenen Gehaltsabrechnungssystems auch nach mehreren Jahren intensiver Arbeit nicht spürbar vorankam. Beck verordnete den Autobauern einen kompletten Neustart, und bereits nach drei Wochen konnte er ein System vorweisen, das wenigstens in der Lage war, einen Lohnscheck auszudrucken. Es wird erzählt, dieser Scheck hänge heute noch eingerahmt in irgendeinem Chrysler-Büro.

**Iteration**: Zeitraum von meist 2-4 Wochen, der geplant wird. Am Ende steht immer ein getestetes, fertiges Produkt.

**Story**: Funktionalität, die für den Kunden nach der Implementierung einen praktischen Nutzen hat. Mehrere Stories werden in einer Iteration implementiert.

**Story Point**: Abstraktes Maß, um die Komplexität einer Story zu schätzen. Z.B. 1-9 SP

**Velocity**: „Geschwindigkeit“ der letzten Iteration in Story Points / Tag. Dient zur Planung, wie viele Stories in die nächste Iteration passen („*Yesterdays Weather*“).

**Planning Game**: Treffen Kunde(n) / Programmierteam, um Stories zu definieren, zu schätzen und und für die nächste Iteration zu priorisieren.

**Customer on Side**: Der Kunde ist Teil des Teams, gibt Auskunft, setzt mit automatische Tests auf.

**Iteration Review**: Gemeinsamer Rückblick des Teams auf die letzte Iteration. Was war gut? Was könnte verbessert werden? Wahl eines Mottos für die nächste Iteration.

**Task**: Aufgabe, die zwei Programmierer im Paar erledigen, idealerweise max. ein Tag lang. Jede Story wird unterteilt in Tasks.

**Tests**: Funktional und Unit Tests, immer *vor* dem Implementieren, automatisch auf jedem Rechner ausführbar.

**Continuous Integration**: Automatisches Bauen (Compilieren, ...) und Ausführen aller Tests.

**Collective Code Ownership**: Jeder darf an allen Stellen beliebig ändern – aber nur im Paar und zum gemeinsamen Code nur hinzufügen, wenn die lokale Integration ok war.

# Testgetriebene Entwicklung

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Als testgetriebene Entwicklung (auch *testgesteuerte Programmierung*, engl. *test first development* oder *test-driven development*, Abkürzung *TDD*, manchmal auch scherzhaft Extreme Testing) bezeichnet man eine Agile Methode zur Entwicklung eines Computerprogramms, bei der die Programmierer Software-Tests vor den zu testenden Komponenten entwickeln.

## Vorgehensweise

Die Vorgehensweise unterscheidet sich zwischen dem Testen im Kleinen (Unit-Test, White-Box-Test) und dem Testen im Großen (System-Test, Black-Box-Test, Akzeptanztest).

Bei Unit-Tests werden Test-Gerüst und Unit-Gerüst parallel entwickelt. Die eigentliche Programmierung erfolgt in kleinen Mikroiterationen. Eine solche Iteration, die nur wenige Minuten dauern sollte, besteht aus drei Hauptschritten:

1. Schreibe einen kleinen Test für den nächsten zu implementierenden Funktionalitätshappen. Dieser Test sollte **nicht** funktionieren.
2. Erfülle den Test mit möglichst wenig Code, um schnell wieder zum "grünen Balken" (alle Tests laufen) zurückzukehren.
3. Räume den Code auf, dazu gehört die Entfernung von Duplikation, Einführung von notwendigen Abstraktionen und Umsetzen der Codekonventionen. Ziel dieses Aufräumens ist die *einfache Form* des Codes.

Diese Schritte werden solange wiederholt, bis dem Entwickler keine sinnvollen Tests mehr einfallen, die fehlschlagen würden und damit die Unit für den Augenblick fertig ist. Die konsequente Umsetzung dieses Vorgehens führt zu evolutionärem Design, einem Entwurfsstil, der die ständige Weiterentwicklung eines Systems in den Vordergrund rückt.

Weil der einzelne Unit Test sowohl Eigenschaften eines White-Box-Tests als auch eines Black-Box-Tests aufweist, wird er als Grey-Box-Test bezeichnet.

Bei System-Tests werden die System-Tests vor dem eigentlichen System entwickelt oder zumindest spezifiziert. Aufgabe des Systems ist bei *testgetriebener Entwicklung* nicht mehr, schriftlich formulierte Anforderungen, sondern spezifizierte System-Tests zu erfüllen.

In beiden Fällen wird großer Wert auf eine möglichst vollständige Testautomatisierung gelegt. Für *testgetriebene Entwicklung* ist es erforderlich, dass sämtliche Tests einfach („per Knopfdruck“) und möglichst schnell ausgeführt werden können. Für Unit-Tests bedeutet das wenige Sekunden, für Systemtests meist maximal einige Minuten.

Neben dem Entgegenwirken einiger Nachteile des klassischen "Hinterher"-Testens besitzt die testgetriebene Entwicklung noch einige weitere Vorteile.

Nennenswerte Vorteile sind:

- Die Erfüllung der Anforderungen wird eine leicht messbare Metrik.
- Die Unit-Tests sind Grey-Box-Tests statt White-Box-Tests.
- Die Durchführung von Refaktorisierungen ist mit weniger Fehlern behaftet (im Idealfall fehlerfrei).
- Die einfache, schnelle Ausführung von Tests ermöglicht es den Programmierern, die meiste Zeit an einem korrekten System zu arbeiten.
- Die Unit-Testsuite stellt eine „ausführbare Spezifikation“ dar. Was ein Softwaresystem leisten soll, ist so in maschinenlesbarer und maschinell

überprüfbarer Form niedergelegt und schlägt so eine Brücke zum Bereich der Software-Dokumentation.

## Umsetzung

Die wesentlichen Bestandteile der Umsetzung sind:

- Framework zur Testentwicklung
- Framework zur Testautomatisierung
- Framework zur Build-Automatisierung, z.B. CruiseControl

In Java kommen dafür meist JUnit und Ant zum Einsatz. Für die meisten anderen Programmiersprachen existieren ähnliche Werkzeuge, wie z.B. für PHP die PEAR-Erweiterung PHPUnit.

Bei der Entwicklung komplexer Systeme in denen mehrere Komponenten zusammenarbeiten müssen, die unabhängig voneinander entwickelt werden z.B. die Entwicklung einer Geschäftsanwendung die eine Datenbank als Persistenzmedium verwendet, werden Mock-Objekte verwendet, um einzelne Komponenten wie z. B. die Datenbankzugriffe ohne die notwendige Existenz einer Datenbank testen zu können. Das Mock Objekt simuliert in dem Fall das Verhalten einer Datenbank mit einer festgelegten Verhaltensweise.

## Kritische Betrachtung

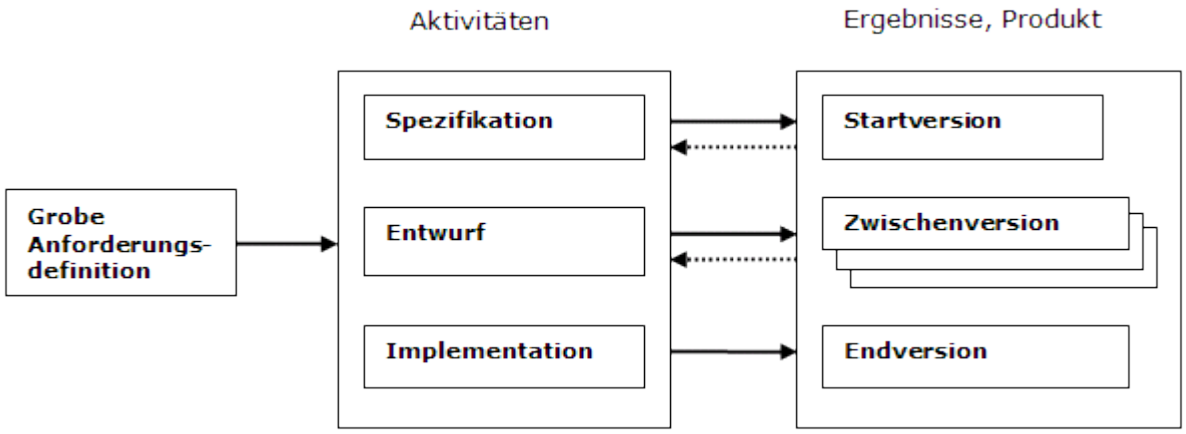
Programmierern, die noch keine Erfahrung in der *testgetriebenen Entwicklung* besitzen, finden die Umsetzung manchmal schwierig oder gar unmöglich. Sie fragen sich, wie man etwas testen soll, das noch nicht vorhanden ist. Das kann dazu führen, dass sie die testgetriebene Entwicklung vernachlässigen, was insbesondere bei agilen Methoden wie dem Extreme Programming Schwierigkeiten oder sogar den Zusammenbruch des Entwicklungsprozesses auslösen kann. Ohne ausreichende Unit-Tests wird keine ausreichende Testabdeckung für z.B. Refaktorisierungen und die gewünschte Qualität erreicht. Dem kann man mit Paarprogrammierung und Schulung entgegenwirken.

# Evolutionäres Prototyping

Ein neuerer Ansatz ist das evolutionäre Prototyping. Evolutionär bedeutet, dass Software häufig Weiterentwicklungen unterworfen ist und der sachgemäße Umgang mit Änderungen zu einem verbesserten Produkt führen kann. Prototyping liefert frühe Versionen eines Softwaresystems zur Begutachtung durch den Anwender. Es ist geeignet, wenn die vollständigen Anforderungen noch nicht vorliegen und alternative Lösungsmöglichkeiten erprobt werden sollen. Das muss natürlich methodisch erfolgen und keinesfalls als trial-and-error-Murkelei.

Evolutionäres Prototyping

- Der Prototyp ist ein Pilotsystem, das bereits im Anwendungsbereich eingesetzt werden kann
- Bildet den Kern des zu entwickelnden Systems
- Liefert keinen Wegwerf-Prototyp
- Muß nach allen Regeln der Kunst entwickelt werden
- Entspricht einem Wachstumsmodell



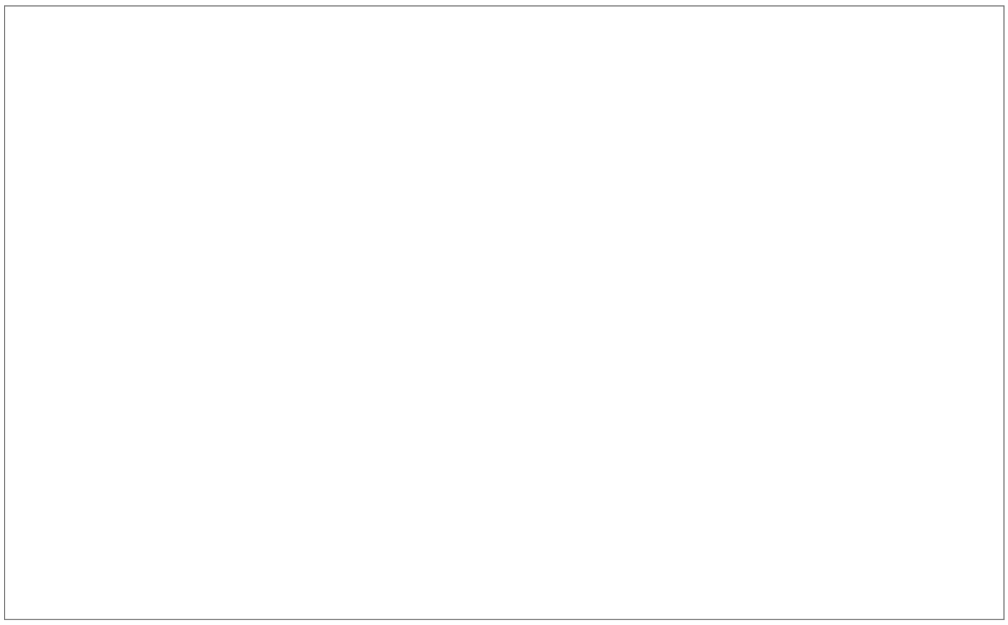

Die Zwischenversionen sind Basis für neue, ergänzende Anforderungen. Im Unterricht ist es geeignet für kleine Projekte, die sich in wenigen Tagen mit (geplanten) verschiedenen Versionen durchführen lassen.

http://oszhdl.be.schule.de/gymnasium/faecher/informatik/softwareprojekte/slc_3.htm

# Produktlebenszyklus

*gekürzt, aus Wikipedia, der freien Enzyklopädie*

Der Produktlebenszyklus ist ein Konzept der Betriebswirtschaftslehre und beschreibt den Prozess zwischen der Markteinführung bzw. Fertigstellung eines marktfähigen Gutes und seiner Herausnahme aus dem Markt. Dabei wird die "Lebensdauer" eines Produktes in mehrere Phasen unterteilt, die die Hauptaufgaben der aktiven Produktpolitik im Rahmen des Lebenszyklus-Managements (*engl. life cycle management*) darstellen.
Die Produktlebenszyklus-Theorien, die auf die Arbeiten von Vernon (1966) und Hirsch (1967) zurückgehen, unterteilen das "Leben" eines Produktes auf dem Markt in vier Phasen: *Entwicklung und Einführung*, *Wachstum*, *Reife/Sättigung* und *Schrumpfung/Degeneration*.



Darstellung der Umsatz- und Gewinnkurve eines Produkts im zeitlichen Verlauf des Produktlebenszyklus

Die einzelnen Phasen des PLZ werden in der Literatur unterschiedlich fein heruntergebrochen:

- Einführungsphase
- Wachstumsphase
- Reifephase
- Sättigungsphase
- Degenerationsphase
- ggf. wird auch noch von einer Nachlaufphase gesprochen.

## Einführung

Mit Beginn der **Einführungsphase** hat das Unternehmen bereits durch Werbung und Public Relations auf das neue Produkt aufmerksam gemacht. Somit steigen die Umsätze allmählich an. In dieser Phase werden aufgrund der vorangegangenen Kosten für die Produktentwicklung und der anhaltenden Kommunikationsausgaben jedoch noch keine Gewinne erzielt. In dieser Phase entscheidet sich, ob der Markt das Produkt überhaupt annimmt. Der Imageaufbau entsteht hier aufgrund der Aussagen der Marktkommunikation. Bei Produkten, die schon bei der Markteinführung ein Verkaufshit sind, gibt es am Anfang oft Fertigungsprobleme bezüglich der großen Nachfrage. Die Einführungsphase ist

beendet, wenn der Break-Even erreicht ist, die Erlöskurve also die Gesamtkosten durchbricht.

## Wachstum

Mit Beginn der **Wachstumsphase** werden erstmalig Gewinne erzielt, obwohl die Ausgaben für Promotion und Kommunikation anhaltend hoch sind. Die Phase ist durch schnelles Wachstum gekennzeichnet und Werbung beschleunigt dieses Wachstum am Markt, die Preis- und Konditionenpolitik nimmt an Bedeutung zu. Auch die Konkurrenten werden auf das Produkt aufmerksam (*Free-Rider-Problem*). Die Wachstumsphase endet, wenn die Umsatzkurve von einer progressiven auf eine degressive Steigung wechselt.

## Reife

Die **Reifephase** ist meist die längste Marktphase. Diese Phase ist die profitabelste, da die Gewinnkurve hier am höchsten ist. Aufgrund der ansteigenden Konkurrenz sinken zum Ende der Phase die Gewinne. Trotzdem besitzen die Unternehmen immer noch einen hohen Marktanteil. Diesen können sie durch ein geeignetes Erhaltungsmarketing und durch Produktvariationen erhalten bzw. weiter ausbauen.

## Sättigung

In aller Regel tritt dann irgendwann die **Sättigungsphase** ein. Das Produkt hat kein Marktwachstum mehr - Umsätze und Gewinne gehen zurück. Durch verschiedene Modifikationen kann man nun versuchen, mehr Kunden zu gewinnen. Ein Beispiel dafür ist Coca-Cola - von Stagnation kann da nicht die Rede sein. Die Sättigungsphase endet, wenn die Umsatzerlöse die Deckungsbeitragsgrenze wieder unterschreiten, wenn also keine Gewinne mehr erzielt werden können.

## Rückgang

Die nächste Phase ist die **Rückgangsphase** (Degeneration): Der Markt schrumpft und der Umsatzrückgang kann auch durch gezielte Marketingmaßnahmen nicht abgefangen werden. Das Produkt verliert Marktanteile und hat ein negatives Wachstum, die Gewinne sinken und das Portfolio sollte bereinigt werden, es sei denn, es bestehen Verbundbeziehungen mit anderen Produkten (Economies of Scope). Wenn hier nicht richtig und schnell gehandelt wird, entstehen unnötige Kosten für ein Produkt, das kaum noch Umsätze einfährt. Zeichnet sich die Rückgangsphase ab, kann auch der Relaunch (Rekonsolidierungsphase) eines Produktes erwogen werden. Dabei wird das Produkt erheblich modifiziert und neu positioniert. Zielsetzung dieser Maßnahme ist, dass das Produkt einen weiteren Lebenszyklus durchläuft. Ein Beispiel dafür ist die Umstellung vom Golf I auf den Golf II. Wird kein Relaunch gestartet, ist die Rückgangsphase mit dem Sinken des Umsatzes auf Null beendet - die Produktion wird eingestellt. Damit hat das Produkt seinen Lebenszyklus durchlaufen - es ist gewissermaßen *gestorben*.

## Nachlauf

Die **Nachlaufphase** umfasst alle nach Einstellung der Produktion anfallenden Aktivitäten im Zusammenhang mit dem Produkt, wie Garantieleistungen, Ersatzteilversorgung, Rücknahme und Entsorgung von Alt-Produkten sowie die Desinvestition von Betriebsmitteln. Meist übersteigen dann die Auszahlungen die Einzahlungen, so dass der Gesamterfolg des Produktes sinkt.

# Lastenheft und Pflichtenheft

## Lastenheft

Ein Lastenheft (auch Anforderungsspezifikation oder Requirement Specification) beschreibt die unmittelbaren Anforderungen, Erwartungen und Wünsche an ein geplantes Produkt.

Gemäß DIN 69905 (Begriffe der Projektabwicklung) beschreibt das Lastenheft die „vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrages“. Das Lastenheft beschreibt in der Regel also, was und wofür etwas gemacht werden soll (Fachkonzept). Die Adressaten des Lastenhefts sind der (externe oder firmeninterne) Auftraggeber sowie die Auftragnehmer. In der Softwaretechnik ist das Lastenheft das Ergebnis der Planungsphase und wird in der Regel von den Entwicklern als Vorstufe des Pflichtenhefts erarbeitet, meist in tabellarischer Form vorliegend.

## Pflichtenheft

Das Pflichtenheft beschreibt dann, was und womit etwas realisiert werden soll.

Dabei können gewöhnlich jeder Anforderung des Lastenhefts eine oder mehrere Leistungen des Pflichtenheftes zugeordnet werden. So wird auch die Reihenfolge der beiden Dokumente im Entwicklungsprozess deutlich: Die Anforderungen (requirements) werden durch Leistungen (features) erfüllt.

Nach DIN 69905 enthält das Pflichtenheft die „vom Auftragnehmer erarbeiteten Realisierungsvorgaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenheftes“.

Je nach Einsatzgebiet und Branche können sich Lastenhefte in Aufbau und Inhalt stark unterscheiden.

## Bemerkung:

- In der Praxis werden die Begriffe Lastenheft, Pflichtenheft und Spezifikation oft nicht klar gegeneinander abgegrenzt oder gar synonym verwendet.
- Die unklare Verwendung der Begriffe Lastenheft und Pflichtenheft ist häufig Ursache für Missverständnisse.

# Aufgabe zu Lasten- und Pflichtenheft

### Arbeitsschritte

- Verschaffen Sie sich einen Überblick.
- Beachten Sie die Fragen unten!
- Bereiten Sie das Thema so auf, dass Sie es anschließend in 5 Minuten präsentieren können.
- Grafiken stehen über den Beamer zur Verfügung.
- Sie haben 20 Minuten Zeit zur Vorbereitung!

### Hilfestellung zu Vorgehensmodellen

- Wer erstellt das Lastenheft, wer das Pflichtenheft?
- Wird erst Lasten- oder Pflichtenheft erstellt?
- Überlegen Sie sich eine einfache Programmerweiterung für eine Standardsoftware. Was würde im Lastenheft stehen, was im Pflichtenheft?
- Haben Sie in Ihrer Firma schon ein Lasten-/Pflichtenheft gesehen?

# Aufgabe zu den Vorgehensmodellen

### Arbeitsschritte

- Verschaffen Sie sich einen Überblick.
- Beachten Sie die Fragen unten!
- Bereiten Sie das Thema so auf, dass Sie es anschließend in 5 Minuten präsentieren können.
- Grafiken stehen über den Beamer zur Verfügung.
- Sie haben 20 Minuten Zeit zur Vorbereitung!

### Hilfestellung zu Vorgehensmodellen

- Wie wird zeitlich unterteilt?
- Gibt es Rollen, die Mitarbeitern übertragen werden (z.B. Chefarchitekt?)
- Wie ist der Kunde eingebunden?
- Wie findet die Kommunikation statt?
- Wird dieses Vorgehensmodell in Ihrer Firma angewandt?

# Aufgabe zum Produktlebenszyklus

### Arbeitsschritte

- Verschaffen Sie sich einen Überblick.
- Beachten Sie die Fragen unten!
- Bereiten Sie das Thema so auf, dass Sie es anschließend in 5 Minuten präsentieren können.
- Grafiken stehen über den Beamer zur Verfügung.
- Sie haben 20 Minuten Zeit zur Vorbereitung!

### Hilfestellung zu Vorgehensmodellen

- Wie wird zeitlich unterteilt?
- Wie sind die Zeitpunkte definiert?
- Suchen Sie ein Beispiel für einen Relaunch im Software-Bereich
- In welchen Phasen befinden sich wahrscheinlich die verschiedenen Windows-Versionen?

# Aufgabe zur Testgetriebenen Entwicklung

### Arbeitsschritte

- Verschaffen Sie sich einen Überblick.
- Beachten Sie die Fragen unten!
- Bereiten Sie das Thema so auf, dass Sie es anschließend in 5 Minuten präsentieren können.
- Grafiken stehen über den Beamer zur Verfügung.
- Sie haben 20 Minuten Zeit zur Vorbereitung!

### Hilfestellung zu Vorgehensmodellen

- Wie wird zeitlich unterteilt?
- Welche technischen Hilfsmittel kommen zum Einsatz?
- Finden Sie eine typische Phase in einem Vorgehensmodell, wo Testgetriebene Entwicklung vorkommen kann.
- Wird in Ihrer Firma testgetrieben gearbeitet?

# Aufgabe zur Evolutionären Entwicklung

### Arbeitsschritte

- Verschaffen Sie sich einen Überblick.
- Beachten Sie die Fragen unten!
- Bereiten Sie das Thema so auf, dass Sie es anschließend in 5 Minuten präsentieren können.
- Grafiken stehen über den Beamer zur Verfügung.
- Sie haben 20 Minuten Zeit zur Vorbereitung!

### Hilfestellung zu Vorgehensmodellen

- Wie wird zeitlich unterteilt?
- Welche technischen Hilfsmittel kommen zum Einsatz?
- Finden Sie eine typische Phase in einem Vorgehensmodell, wo Evolutionäre Entwicklung vorkommen kann.
- Wird in Ihrer Firma evolutionär gearbeitet?