

Airbus-Unfall (1): Was damals passierte

- **Am 14.09.1993 verunglückt ein Airbus A 320 bei der Landung auf dem Flughafen von Warschau.**

Schlechte Wetterbedingungen (starker Regen und im letzten Moment auftretender Rückenwind) → die Maschine fliegt mit erhöhter Geschwindigkeit an

Die Maschine setzt zu spät auf und kommt nicht mehr rechtzeitig zum Stehen.

Der Zusammenprall mit einem Erdwall am Ende der Rollbahn löst ein Feuer aus, das das Flugzeug zerstört.

Bei diesem Unfall spielen nicht nur, aber auch die Computer-Systeme des A 320 eine entscheidende Rolle.

Airbus-Unfall (2): Die technischen Hintergründe

Steuersysteme des A320 initiieren den Bremsvorgang, wenn:

beide Hauptfahrwerke Bodenkontakt haben **und**
entweder die Rotationsgeschwindigkeit mindestens eines der
Räder 130km/h beträgt
oder die Höhe unter 3,5m ist

**Gemäß dieser Spezifikation arbeitete das System korrekt, leitete
den Bremsvorgang aber zu spät ein wegen:**

der wetterbedingten überhöhten Geschwindigkeit
des Aquaplanings
der Schräglage, mit der das Flugzeug aufsetzte

Grundannahmen der SE

***Softwareentwicklung* - der Begriff fasst die idealisierenden Grundannahmen über die Konstruktion von Software zusammen. Diese Grundannahmen sind:**

- Gegenstand der Softwareentwicklung ist die Herstellung eines Produktes.
- Softwareentwicklung beruht auf vorgegebenen Problemen mit fest definierbaren Anforderungen.
- Der Herstellungsprozess ist anhand von Prozessmodellen formalisierbar und durchführbar.
- Der Einsatzkontext von Software kann ausgeklammert werden.
- Die Arbeitsteilung kann anhand der Produktstruktur frühzeitig und durchgehend erfolgen. Kommunikation wird über Dokumente geregelt

1. Einblick in die Erstellung professioneller Softwareprodukte

Ziel: Sensibilisierung des Lesers für die speziellen Probleme, die diese Thematik mit sich bringt.

- Das Entwickeln von Software birgt eine Vielzahl von Problemen
- Die Kommunikation stellt einen wesentlichen Bestandteil der Softwareentwicklung dar.

1.1 Begriffsdefinitionen:

- *Software* ist eine Menge von Programmen und Daten zusammen mit begleitenden Dokumenten, die für ihre Anwendung notwendig oder hilfreich sind
- *Softwaresystem* sind mehrere Programme, Daten und Dokumentationen, die zueinander in Beziehung stehen und ein gemeinsames Ganzes bilden
- *Softwareentwicklung* bezeichnet den Herstellungsprozess eines Softwareproduktes

1.2 Eigenschaften von Software:

- Software kann nur durch den *Ablauf auf einer konkreten Hardware* wahrgenommen werden
- Software unterliegt *keinen Verschleißerscheinungen*. *Wartung* von Software hat demnach eine andere Bedeutung als bei materiellen Produkten:
 - Behebung von erkannten Fehlern
 - notwendige Anpassung an neue Rahmenbedingungen
 - ➔ die Software überlebt die Hardware oft über mehrere Generationen

1.3 Klassifizierung von Software:

Einteilung der Software in verschiedene Arten:

- *nach Größe:* hierbei gibt es verschiedene Maße:
 - nach Anzahl der Codezeilen (LOC)
 - nach Anzahl der Monate, die Entwickler an einem Projekt gearbeitet haben (Mannmonate)
 - Kosten
 - Speicherplatzbedarf
 - Hardwareanforderungen

- *nach Anwendungsgebiet*
 - abhängig von der Verwaltung von Massendaten (z.B. Versicherungsunternehmen)
 - abhängig von Geschwindigkeitsaspekten (z.B. Anwendung in einer Automatisierungsstraße)

1.3 Klassifizierung von Software:

➤ *nach Betriebssystemnähe:*

- Entwicklung von *Anwendungssoftware* für die klassischen Anwendungsfelder
- Entwicklung von *Systemsoftware* welche die Plattform für die Entwicklung und die Anwendung der Anwendersoftware darstellt z.B. (Betriebssystementwicklung)

➤ *nach Spezialisierungsgrad*

- *Individualsoftware*, wird für einen speziellen Kunden genau zugeschnitten entwickelt (der Kunde redet bei der Entwicklung mit)
- *Standardsoftware*, wird für eine größere Gruppe von Benutzern konzipiert (hier wird von allgem. Kundenwünschen und Bedürfnissen ausgegangen)

2. Geschichtliches zur Softwareentwicklung

Die Softwareentwicklung *früher:*

- künstlerisch-kreative Programmentwicklung
- Wissenschaft und Technik waren die wichtigsten Anwendungsgebiete
- bereits bekannte mathematische Algorithmen wurden in Rechenprogramme übertragen

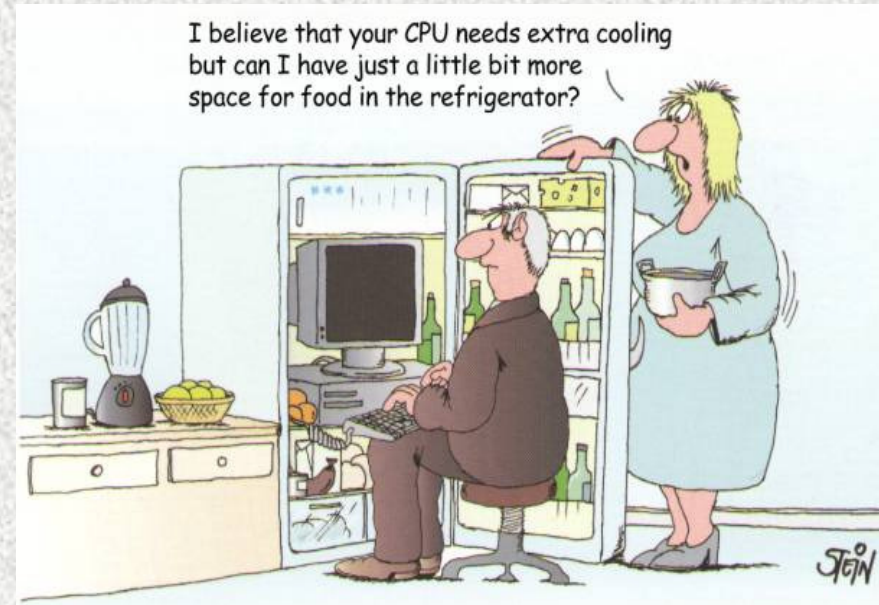
2. Geschichtliches zur Softwareentwicklung

Die Softwareentwicklung heute:

Programme für nicht mehr rein numerisch, naturwissenschaftliche Probleme.
Die Randbedingungen für den Einsatz der Produkte ändern sich häufig.

Die Komplexität nimmt immer mehr zu

→ *als entscheidende Schwierigkeit erwies sich die Notwendigkeit, eine komplexe Aufgabenstellung in überschaubare Teilaufgaben zu zerlegen*



2. Geschichtliches zur Softwareentwicklung

Die Softwareentwicklung

heute:

- Software bekommt eine zunehmende Bedeutung für alle Bereiche des täglichen Lebens, was zu veränderten Qualitätsanforderungen führt:
- die Benutzer sind keineswegs mehr Spezialisten im Umgang mit dem Computer
- Anforderung an den Computer, einen bei den alltäglichen Tätigkeiten zu unterstützen
- viele Anwendungen stellen erhöhte Sicherheitsanforderungen
- *Die Wünsche der Anwender richtig zu analysieren ist dabei eines der größten Probleme*

2. Geschichtliches zur Softwareentwicklung

Die Rolle des Programmierers *früher:*

ein einzelner kontaktscheuer
Entwickler, der in einer Garage
geniale Hardware und zugehörige
Programme entwickelt:



Zu Beginn des Computerzeitalters wurden Rechner wesentlich
anders genutzt als heute. Der Entwickler von Software war
meist auch gleichzeitig der Anwender!

2. Geschichtliches zur Softwareentwicklung

Die Rolle des Programmierers *heute:*

es vollzog sich eine Trennung der Personenkreise:
Softwareentwickler und -Nutzer. Die Software muss daher
erhöhten Qualitätskriterien genügen, weil ein Eingriff der
Entwickler bei der Nutzung praktisch nicht mehr möglich ist.



**“Dear Mom and Dad...How have you been?
I am fine. I miss you. If my hard drive
ever crashes, I will come downstairs to visit
you sometime. PS: Please e-mail me some food.”**

2. Geschichtliches zur Softwareentwicklung

**in den 50er und 60er Jahre:
*zusammengefasst:***

- sprunghaft wachsende Leistungsfähigkeit von Rechnern
- Computer hält Einzug in weite Bereiche der Wirtschaft, Verwaltung und Militär
- Entwicklung höherer Programmiersprachen
- komplexe Systemprogramme wurden erforderlich
- Software und Hardware werden allmählich abgegrenzt
- keine einheitlichen, klaren Vorstellungen über das neue, eigenständige Tätigkeitsfeld von Programmieren
- keine gesicherten Ausbildungsgrundlagen
- ➔ Programmierer als Künstler

2. Geschichtliches zur Softwareentwicklung

1965: Software-Krise

in dieser Zeit boomte die Hardwareentwicklung.
Die Aufgabenstellungen wurden umfangreicher und komplexer.

Für die Lösung komplexer Probleme gab es jedoch weder theoretische Grundlagen noch irgendwelche methodischen Hilfsmittel und so kam es wegen Softwareproblemen zu keiner zufriedenstellenden Nutzung der leistungsfähigen Computer

→ *diese Probleme wuchsen sich in den 60er Jahren zu der genannten Softwarekrise aus*

2. Geschichtliches zur Softwareentwicklung

Symptome:

- Aufwand der Softwareentwicklung nicht abschätzbar
 - Software zu teuer
 - Software nicht termingerecht fertig
 - Software entspricht nicht den Anforderungen, die ausgelieferten Produkte wiesen noch erhebliche Mängel auf
 - Anforderungen entsprechen nicht den realen Gegebenheiten (= schlechte Spezifikation)
 - Software enthält zum Teil unbehebbar Fehler
 - Unerwünschte Folgekosten und Akzeptanzprobleme
 - Nachbesserung ist unerwartet aufwendig
- ➔ *dadurch entstanden große wirtschaftliche Schäden*

2. Geschichtliches zur Softwareentwicklung

Ende der 60er Jahre wird schließlich gefordert, dass ein ingenieurmäßiges Vorgehen für die Softwareentwicklung vonnöten sei:

Einberufung von zwei internationalen Konferenzen unter Regie der NATO .

Experten aus Wissenschaft, Wirtschaft und Militär konstituieren den Fachbereich **Software-Engineering**

Programme sind Industrieprodukte, d.h die Programmierung wird nicht mehr als Kunst verstanden, sondern als ingenieurmäßige Software-Entwicklung

Die Herstellung von Software muss den Charakter einer Ingenieursdisziplin aufweisen

2. Geschichtliches zur Softwareentwicklung

Abhilfe der Krise

Von der Disziplin Software-Engineering wird nun erwartet, dass sie *Methoden, Werkzeuge, Normen und Hilfsmittel bereitstellt*, die es ermöglichen, die technischen Probleme (wie Spezifikation, Entwurf, Implementierung, Test, Effizienz, Wiederverwendbarkeit, Dokumentation, Wartung) sowie die organisatorischen Probleme (Projektorganisation, Schnittstellendefinition), die bei der Herstellung von Software auftreten, in den Griff zu bekommen.

Mit dem Ziel:

- *Qualitativ akzeptable Software mit wirtschaftlich vertretbarem Aufwand zu produzieren und einzusetzen*

2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

In Teilen Bewältigung der Softwarekrise der 60er Jahre, denn:

- ausgereifte und lehrbare Konzepte für Modellierung, Softwarearchitektur und Programmierung liegen vor.
- Die Qualität der Software kann durch Anwendung von Modellen verbessert werden!

2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

aber:

- Große Diskrepanz zwischen Wissenschaft und Praxis auch heute noch. Der theoretisch vertretene Anspruch kann häufig nicht im betrieblichen Alltag eingelöst werden.
- Das Problem, Unformales in Formales zu übersetzen, ist nicht grundsätzlich gelöst.



2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

Verbliebene und neue Probleme sind daher:

- mangelnde Umsetzbarkeit des erlernten SE-Wissens
- Akzeptanzprobleme für (stets) wechselnde Methoden
- Mehrarbeit bei der Softwareentwicklung durch Methoden und Werkzeuge

2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

besondere Anforderungen an das Software Engineering:

- *Altlasten*: viele große Softwaresysteme sind bereits vor vielen Jahren entwickelt worden. deren Wartung ist sehr schwierig, da sie noch nicht ingenieurmäßig programmiert wurde und die ursprünglichen Entwickler nicht mehr greifbar sind
- *Heterogenität*: viele Systeme arbeiten mittlerweile als verteilte Systeme, die über ein Netzwerk verbunden sind und somit auf unterschiedlichster Systemsoftware laufen müssen. Die Software muss sehr flexibel auf solche Anforderungen reagieren

2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

besondere Anforderungen an das Software Engineering:

- *Zeitdruck*: viele Techniken der Softwareentwicklung sind zeitaufwändig. Gleichzeitig sind die Auslieferungszeiten für eine neue Software sehr kurz geworden.
 - ➔ der Druck auf Entwickler und Management wird größer und geht zu Lasten der Qualität!

2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

Fakten aus der TA-Forschung (1)

TA (Technology Assessment): Zu Software und Softwareprojekten*

- Jedes sechste Datenverarbeitungsprojekt wird ohne Ergebnis abgebrochen
- Alle Projekte überzogen den Zeit- und Kostenrahmen um 100-200 %
- Auf 100 ausgelieferte Programmzeilen entfallen im Durchschnitt 3 Fehler
- Auf sechs große SW-Systeme, die eingesetzt werden, entfallen zwei gestrichene.

*Quelle: Schinzel - Schnittstellen

2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

Fakten aus der TA-Forschung (2)

- Es gibt keine korrekte SW nennenswerter Größe und Funktionalität.
- Für 60-70 % aller Softwarefehler sind Fehler in der Problemanalyse, der Spezifikation und der Modellierung verantwortlich.
- Fehler und Unzulänglichkeiten in der Modellierung und Formalisierung werden, wenn überhaupt, meist während der Entwicklung festgestellt und korrigiert, Fehler in der Spezifikation erst im nachhinein.

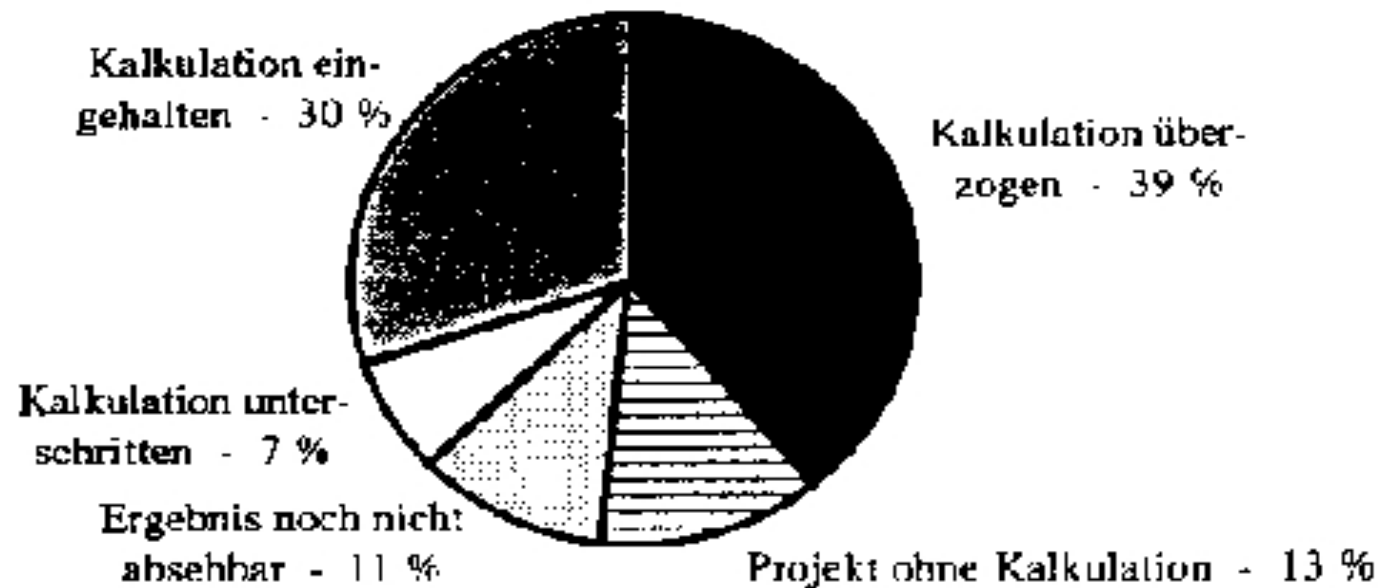
2. Geschichtliches zur Softwareentwicklung

Der gegenwärtige Stand

Fakten aus der TA-Forschung (3)

- Die Anwendung einer großen Zahl von SW-Engineering-Methoden auf aktuelle Projekte ergibt nur eine 30%-ige Verbesserung der Verlässlichkeit der SW-Systeme
- Integration von Benutzern der SW in die SW-Entwicklung schlägt sich z.B. in einer besseren Wartbarkeit der SW nieder.
- Prototyping bedarf zwar am Anfang eines größeren Aufwands, führt aber zu adäquateren Lösungen. Problematisch beim Prototyping ist, dass der erste Prototyp die Vorstellungskraft der Nutzerinnen leitet.

Kosten und Termine: Auch heute selten richtig geschätzt



Quelle: Weltz, Ortman: Das Softwareprojekt, S. 36

3. Qualitätskriterien für Softwareprodukte

Zwischen 1977 und 1994 ist die Defektrate auf den vermeintlich sehr guten Wert von 0,2 bis 0,05 Fehlern pro 1000 Zeilen Quellcode gesenkt worden (um ca. das 100-fache, verglichen mit den Vorjahren).

Eine Defektrate von 0,1% bedeutet allerdings:

- pro Jahr ca. 300 versagende Herzschrittmacher
- pro Woche ca. 500 Fehler in einer medizinischen OP
- pro Tag 16000 verlorene Briefe
- pro Stunde 22000 Schecks, die falsch verbucht werden

Es ist daher leicht einzusehen, dass die Qualitätssicherung einer Software ein wichtiger Aspekt bei Ihrer Entwicklung ist

3. Qualitätskriterien für Softwareprodukte

Qualität muss ein Ziel der Software-Entwicklung sein und wirkt sich sowohl auf die Planung, wie auch auf die Durchführung und die Kontrolle der Softwareherstellung aus!

Die Qualitätsanforderungen an Softwareprodukte führten daher zur *klaren Definition von Qualitätsmerkmalen*:

3. Qualitätskriterien für Softwareprodukte

- Korrektheit
- Zuverlässigkeit
- Benutzerfreundlichkeit
 - Adäquatheit
 - Erlernbarkeit
 - Robustheit
- Wartungsfreundlichkeit
 - Lesbarkeit
 - Erweiterbarkeit
 - Testbarkeit
- Effizienz
- Portabilität

Diese Qualitätsmerkmale sind durchaus nicht unabhängig voneinander:

Eine Verbesserung hinsichtlich eines Merkmals kann die Verschlechterung eines anderen bewirken!

3. Qualitätskriterien für Softwareprodukte

Korrektheit

Unter der Korrektheit versteht man die Eigenschaft, dass das Programmsystem die der Programmentwicklung zugrundegelegte Spezifikation erfüllt.

Besonders kritisch ist das Problem der Korrektheit eines Programms, das in ein komplexes Programmsystem eingebettet werden soll.

Ist p die Wahrscheinlichkeit dafür, dass ein einzelnes Programm korrekt ist, so gilt für die Wahrscheinlichkeit P der Korrektheit eines Programmsystems, das aus n Programmen besteht:

$$P = p^n$$

→ Wenn n sehr groß ist, muss p fast 1 sein, damit P überhaupt wesentlich von Null verschieden ist!

3. Qualitätskriterien für Softwareprodukte

n \ p	0.99	0.95	0.90
1	0.99	0.95	0.90
10	0.90	0.60	0.35
50	0.61	0.08	0.01
100	0.37	0.01	0.00

3. Qualitätskriterien für Softwareprodukte

Zuverlässigkeit

Die Zuverlässigkeit eines Programms wird durch die Korrektheit und durch die Verfügbarkeit bestimmt, d.h. die Zuverlässigkeit eines Programmsystems resultiert *aus der Korrektheit innerhalb eines bestimmten Zeitintervalls!*

Das Programm kann als zuverlässig angesehen werden, wenn es eine geringe Fehlerrate aufweist (*die Wahrscheinlichkeit, dass in einem bestimmten Zeitintervall ein bestimmter Fehler auftritt*)

3. Qualitätskriterien für Softwareprodukte

Benutzerfreundlichkeit

1.) Adäquatheit :

- bezieht sich auf die vom Benutzer verlangte *Eingabe*
- soll sich auf das Notwendigste beschränken
- soll flexible Dateneingaben gestatten
- soll Plausibilitätskontrollen der Eingaben durchführen
- bietet einheitliche, klare und einfache Benutzerführung
- bezieht sich auf die vom Programm angebotenen *Leistungen*
- soll den Wünschen des Benutzers angepasst sein
- soll u.U. erweiterbar sein
- bezieht sich auf die vom Programm produzierte *Ergebnisse*
- Ausgabe soll übersichtlich und gut strukturiert sein
- eventuelle Fehlermeldungen müssen in einer verständlichen Form ausgegeben werden

3. Qualitätskriterien für Softwareprodukte

Benutzerfreundlichkeit

2.) Erlernbarkeit :

hängt von der Klarheit und Einfachheit des Benutzerhandbuchs ab

3. Qualitätskriterien für Softwareprodukte

Benutzerfreundlichkeit

3.) Robustheit

bedeutet, die Eigenschaft, die Auswirkungen von Bedienungsfehlern, falschen Eingaben und Hardwarefehlern abzuschwächen

3. Qualitätskriterien für Softwareprodukte

Wartungsfreundlichkeit

Darunter versteht man

- seine Eignung für die Lokalisierung von Fehlerursachen sowie
- die Eignung für die Durchführung von Fehlerkorrekturen
- die Eignung für Veränderung oder Erweiterung der Programmfunktionen

3. Qualitätskriterien für Softwareprodukte

Wartungsfreundlichkeit

Sie hängt ab von der

1.) Lesbarkeit

Darstellungsform, Programmierstil, Strukturiertheit, Qualität der Dokumentation

2.) Erweiterbarkeit

ist abhängig von der Strukturiertheit des Programms und den Möglichkeiten, welche die Implementierungssprache zur Verfügung stellt

2.) Testbarkeit

Eignung für die Verfolgung des Programmablaufs und für die Lokalisierung von Fehlern. Sie hängt von der Strukturierung des Programmsystems ab

3. Qualitätskriterien für Softwareprodukte

Effizienz

Darunter versteht man die Eigenschaft des Programmsystems, seinen Zweck unter bestmöglicher Ausnutzung aller Ressourcen zu erfüllen.

Dabei bezieht sich „Ressourcen“ auf Zeit, Speicherplatz, Übertragungskanäle und periphere Einheiten

3. Qualitätskriterien für Softwareprodukte

Portabilität

Darunter versteht man die Leichtigkeit, mit der es auf andere Rechner übertragen werden kann.

Sie hängt also vom Grad seiner Rechnerunabhängigkeit ab.

Die Portabilität hängt somit von der Wahl der Implementierungssprache, von der Austauschbarkeit der Programmkomponenten, sowie vom Grad der Ausnutzung spezieller Systemfunktionen und Hardwareeigenschaften ab.

Ein Programmsystem kann als portabel bezeichnet werden, wenn der Änderungsaufwand für die Übertragung wesentlich kleiner ist als der Aufwand für eine neue Implementierung.

4. Projektmanagement

Damit die angestrebten Ziele eines Projektes eingehalten werden können bedarf es eines Projektmanagements

Mittlerweile gibt es eine Vielzahl von möglichen Strategien, ein Projekt zu managen!

Es gilt dabei, für ein bestimmtes Projekt schnell und kostengünstig die passende Entwicklungsstrategie zu finden

4. Projektmanagement

Die wichtigsten *Aufgaben des Projektmanagement* liegen darin, die folgenden Fehler zu vermeiden:

- keine Planung durchzuführen
- keine Projektorganisation zu bestimmen
- nicht alle Projektaktivitäten mit einzurechnen (z.B. Meetings)
- keine Risiken mit einzukalkulieren (z.B. Mitarbeiterausfälle)
- nicht alle Randbedingungen zu berücksichtigen
- Diskrepanz zwischen Planung und Wirklichkeit übersehen (hinsichtlich Kostenentwicklung, Qualität oder Fortschritt)
- zu viele Details zu früh planen

4. Projektmanagement

Projektplanung

Die Planung eines Projektes wird beeinflusst von folgenden Faktoren:

- der Aufgabenstellung
- der Projektgröße
- der Erfahrung und Qualifikationen der Projektmannschaft
- den eingesetzten Entwicklungsmethoden

4. Projektmanagement

Projektplanung

Das Ergebnis der Planung ist :

- ein Organisationsplan
- ein Personalplan
- ein Terminplan
- ein Kostenplan
- eine Planung aller Maßnahmen zur Qualitätskontrolle
- eine Planung der Ausbaustufen eines Projekts
- eine Planung der Kundenabsprache

4. Projektmanagement

Projektorganisation

bedeutet :

- Zusammenstellen von Arbeitsteams
- Zuordnen der einzelnen Aufgaben und Verantwortlichkeiten zu den Teams oder Personen
- Festlegung genauer Aufgabenbeschreibungen inklusive Rechten und Pflichten
- Bereitstellung aller Betriebsmittel
- Wiederverwendung von fertiger Software (Programmbibliotheken)

4. Projektmanagement

Wechselwirkung bei der Softwareentwicklung

Softwareentwicklung bedeutet immer auch Interaktion und Kommunikation von Menschen:

- die Erarbeitung eines Produktes erfolgt in Zusammenarbeit der einzelnen Mitglieder der Entwicklungsmannschaft
- sowie der Produktentwickler mit den Kunden
 - ➔ dabei können Schwierigkeiten dadurch entstehen, dass Kunde und Entwickler aus einem fachlich unterschiedlichen Umfeld stammen!

4. Projektmanagement

Kontrolle

die *technische Kontrolle* überwacht die technischen Eigenschaften und die Qualität des Projektes

die *wirtschaftliche Kontrolle* überwacht die Kosten und Termine

→ dabei ist eine besondere Schwierigkeit die Aufwandsschätzung für die einzelnen Aufgaben, die gegebenenfalls stark von dem wirklich benötigten Aufwand abweichen kann

5. Lebenszyklusmodelle

**Die Entwicklung eines
Softwareprojektes ist ohne eine
systematische Vorgehensweise nicht
erfolgreich durchführbar!**

5. Lebenszyklusmodelle

Ziel einer jeden Softwareentwicklung muss es sein, das gewünschte Produkt

- **zum vereinbarten Preis**
- **zum vereinbarten Termin**
- **in der vereinbarten Qualität**

zu liefern

5. Lebenszyklusmodelle

Vorgehensmodelle

Ein Vorgehensmodell (Synonym für Lebenszyklusmodell) stellt eine Beschreibung des Entwicklungsprozesses der Software dar

5. Lebenszyklusmodelle

alle Vorgehensmodelle betrachten die Entwicklung von Software als einen Prozess, der die folgenden wesentlichen Aktivitäten enthalten muss:

- Spezifikation der Funktionalität der Software und aller Randbedingungen (was soll die Software leisten und was soll dabei berücksichtigt werden)
- Entwicklung der Software gemäß der Spezifikation der Anforderungen (Verifikation)
- Validierung der Software hinsichtlich der Kundenerwartungen
- Ermöglichen der Softwareevolution, falls sich Kundenanforderungen und/oder Randbedingungen ändern sollten

5. Lebenszyklusmodelle

Komplexe Aufgaben werden stets systematisch gegliedert.

Mit Hilfe eines Vorgehensmodells wird dabei der Ablauf des Lösungsprozesses geregelt. Er unterteilt ihn in überschaubare Abschnitte, sog. **Phasen** und ermöglicht dadurch eine

schrittweise Planung, Durchführung, Entscheidung und Kontrolle.

Die Phasen zusammengenommen und ihre zeitliche Abfolge bezeichnet man als

Software-Life-Cycle

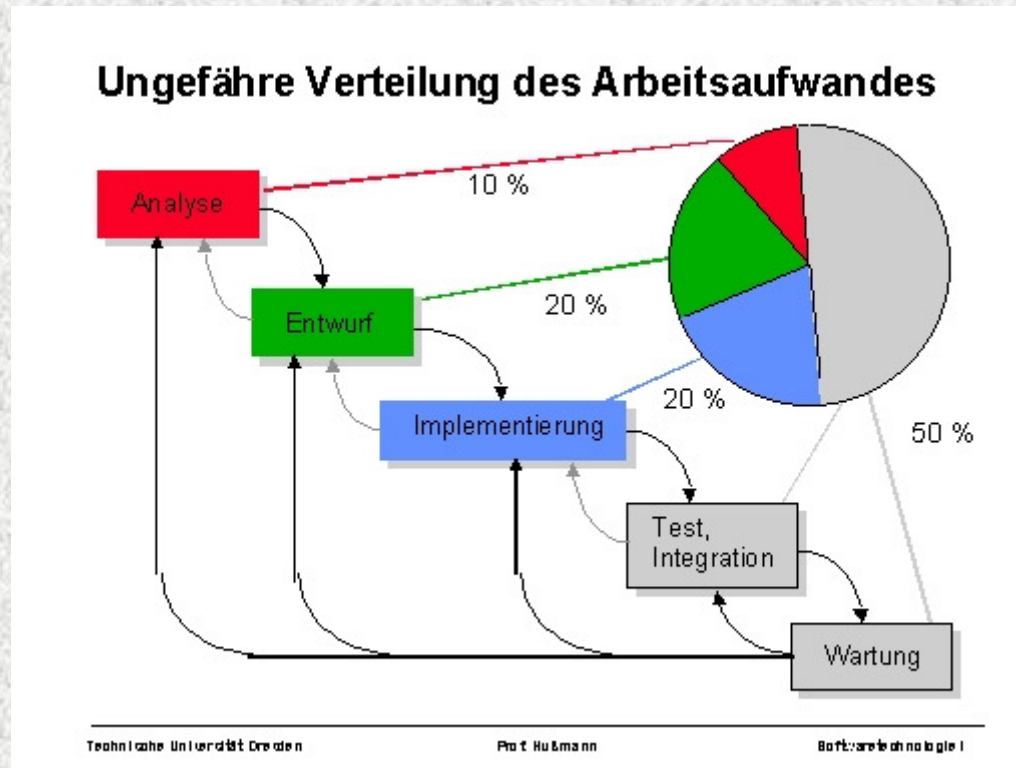
5. Lebenszyklusmodelle

Klassische Lebenszyklusmodelle umfassen in der Regel die Phasen:

- Problemdefinition
- Anforderungsanalyse
- Anforderungsspezifikation
- Planung
- Entwurf
- Kodierung
- Test
- Betrieb
- Wartung

5. Lebenszyklusmodelle

Die Phasen eines Softwareentwicklungsprozesses



5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Phase 1: **Problemdefinition**

Während dieser Phase gibt der Auftraggeber des Software-Produkts dem Entwickler *allgemeine Informationen zu den Voraussetzungen des Projekts* und seinen Wünschen.

Er beschreibt das *Einsatzfeld*, in dem die Software später genutzt werden soll.

Der Auftraggeber sollte dem Entwickler während dieser Phase auch ein *Benutzerprofil* erstellen.

Des weiteren muss abgeklärt werden, *in welcher Systemumgebung* die Software eingesetzt werden soll

5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Phase 2: **Anforderungsanalyse**

Es werden *alle Anforderungen, die der Auftraggeber an das zu entwickelnde Software-Produkt hat, gesammelt und analysiert.*

Dazu gehören neben den funktionalen oder den Systemanforderungen auch Forderungen, die der Auftraggeber an die *Qualität der Software* stellt sowie *Kontrollanforderungen*, welche die Durchführung der Entwicklung betreffen (Abnahmebedingungen, Garantie, Schulungen etc.).

5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Phase 3: **Anforderungsspezifikation**

Die Anforderungsspezifikation beschreibt den *funktionalen Inhalt* des zu implementierenden Systems.

Zur *Dokumentation* der Anforderungen gehören u.a. Schnittstellenbeschreibungen, Anforderungen an die Performance und an den Ressourcenverbrauch

Am Ende der Phase existiert also ein verbindliches Dokument
→ *die Anforderungsspezifikation.*

5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Phase 4: **Planung der Softwareentwicklung**

In dieser Phase wird der Entwicklungsprozess geplant.
Es werden für jede Aufgabe folgende Punkte berücksichtigt:

- Termine
- Verantwortungen
- Vorgaben
- benötigte Mittel und Ressourcen
- Ergebnisse
- Verifizierung

Im besonderen müssen die Tätigkeiten in die Projektplanung einbezogen werden, die zur Qualitätssicherung beitragen

5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Phase 5: **Entwurf**

In dieser Phase, dem Design der Software, wird eine *Software-Entwurfsbeschreibung* erstellt.

Es wird der *interne Aufbau* dargelegt und das Gesamtkonzept verdeutlicht.

Die Software-Entwurfsbeschreibung dient den Entwicklern der Software während aller Entwicklungsphasen als Nachschlagewerk, falls Unklarheiten über den Aufbau der Software bestehen.

5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Phase 6: **Abschluss**

Kodierung

In der Kodierungs- oder Implementierungsphase wird das Software-Produkt nach der Software-Entwurfsbeschreibung erstellt.

Test

Die Testphase dient der Überprüfung, ob die Anforderungen des Auftraggebers durch das implementierte Software-Produkt erfüllt werden.

Betrieb

Das fertige Software-Produkt wird in dieser Phase beim Auftraggeber eingesetzt. Zu dieser Phase werden die Einführung und die Schulung der Benutzer gezählt.

5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Phase 7: **Wartung**

Die Wartung umfasst *alle Arbeiten am Software-Produkt, die dessen fortgesetzte Anwendung über einen längeren Zeitpunkt gewährleisten*.

Im besonderen sollten in dieser Phase alle funktionellen und qualitativen Mängel beseitigt werden, die während der Betriebsphase entdeckt wurden.

Es kommt oft vor, dass während der Wartungsphase bereits eine neue Version (Release) des Software-Produkts geplant wird.

In diesem Fall geht die Wartungsphase wieder in die Phase der Problemdefinition über, und der Entwicklungsprozess wird erneut durchlaufen

5. Lebenszyklusmodelle

Die Phasen eines Softwareprojektes

Die hier genannten Phasen treten in den verschiedenen Vorgehensmodellen (Lebenszyklusmodellen) in unterschiedlicher Betonung und Zusammenstellung immer wieder auf

5. Lebenszyklusmodelle

Überblick

mittlerweile gibt es eine Reihe bekannter Vorgehensmodellen für die Entwicklung von Software:

- die *Wasserfall- oder Phasenmodelle* als die ältesten Vorgehensmodelle
- die *evolutionären Vorgehensmodelle*, nach denen eine Software in mehreren Evolutionsstufen entwickelt wird
- *Extreme Programming*, bei dem in sehr kurzen Iterationsschritten jeweils kleine Funktionalitätsveränderungen schnell durchgeführt werden

5. Lebenszyklusmodelle

Phasenmodelle wurden in zahlreichen Variationen beschrieben.

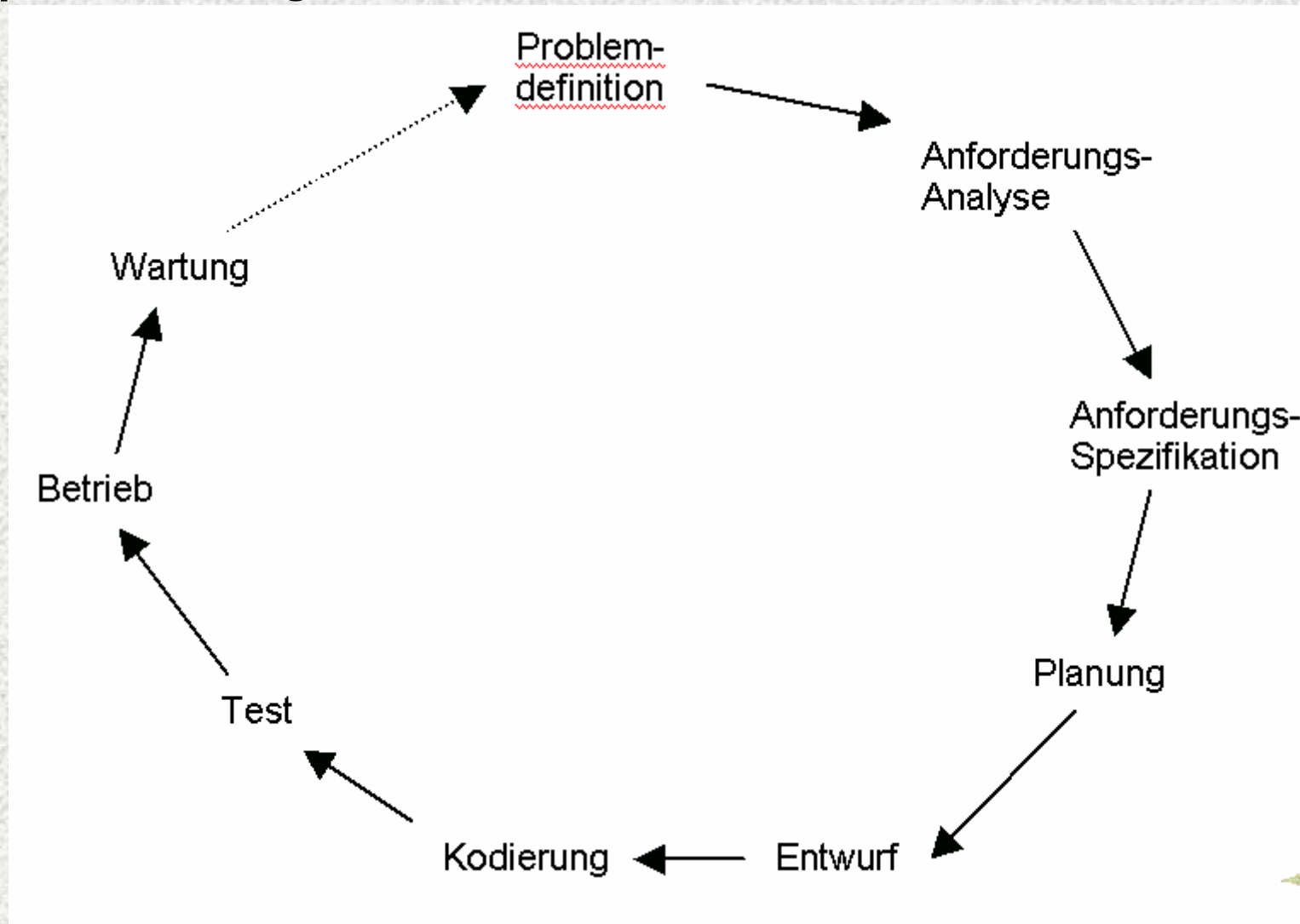
Die ursprüngliche Vorgehensweise beruht dabei auf dem Prinzip, dass für jede der Phasen klar zu definieren ist, welche Ergebnisse erzielt werden müssen und dass eine Phase erst dann in Angriff genommen werden darf, wenn die vorhergehende vollständig abgeschlossen ist.

Diese *rein sequentielle Vorgehensweise* hat sich in der Praxis allerdings als nicht durchführbar erwiesen.

Dies führte zu Modellvarianten, die diese streng sequentielle Vorgehensweise ablösten:

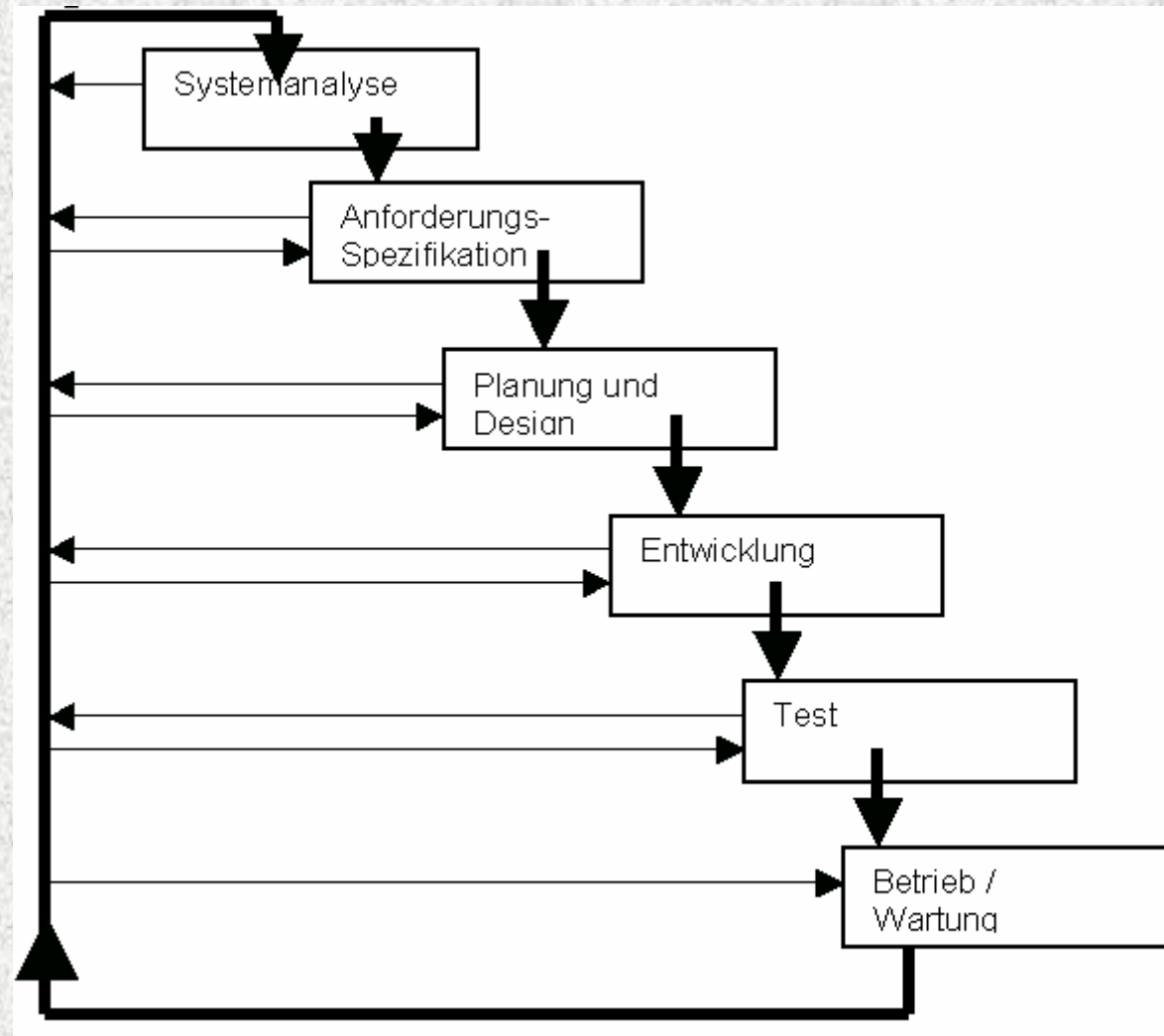
5. Lebenszyklusmodelle

Die sequentielle Vorgehensweise



5. Lebenszyklusmodelle – klassische Modelle

Das Wasserfall-Modell



5. Lebenszyklusmodelle – klassische Modelle

Das Wasserfall-Modell

Es stellt eine Verfeinerung des sequentiellen Modells dar, die streng sequentielle Vorgehensweise wird aufgeweicht

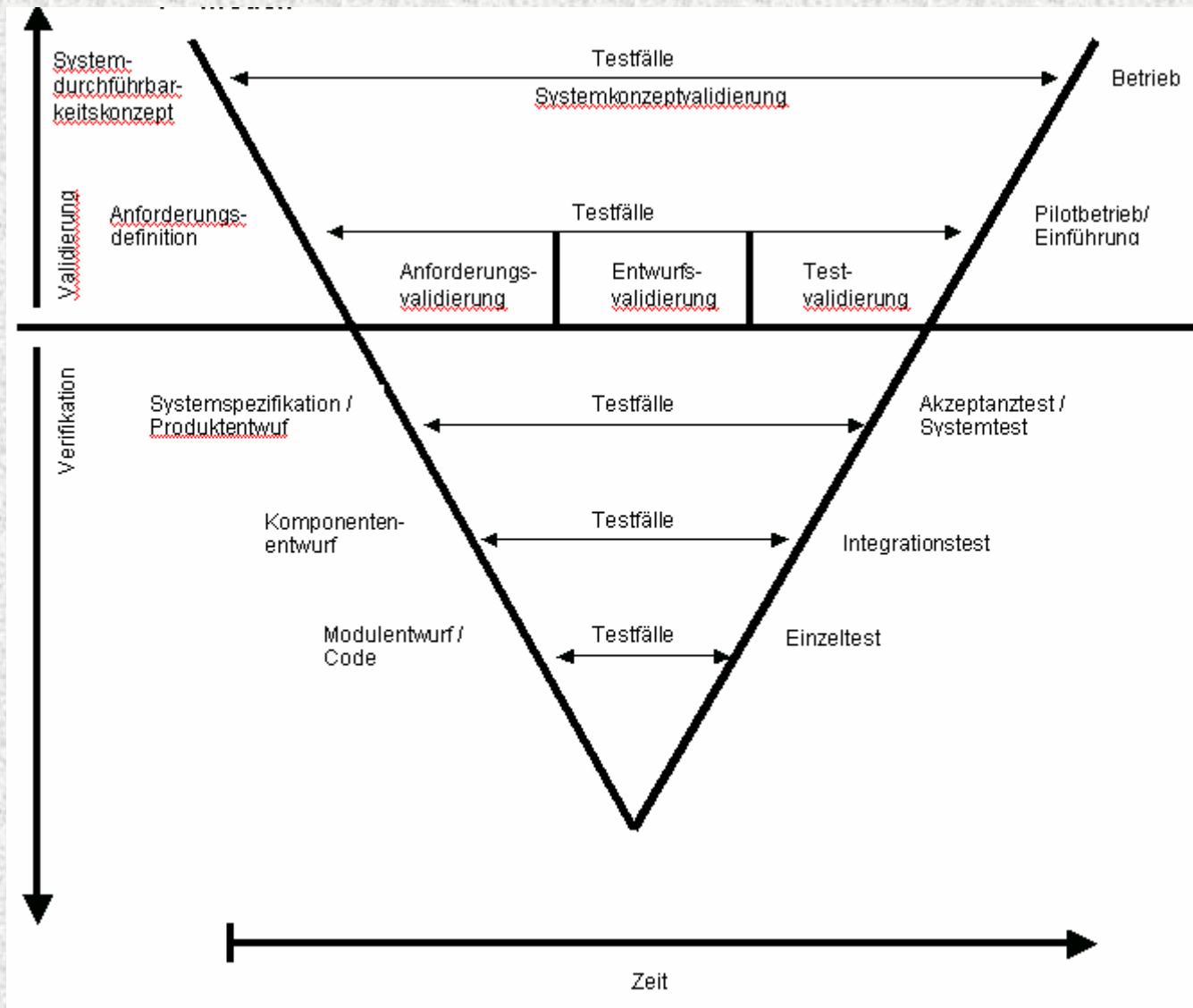
Rückkoppelung (**Iterationen**) zwischen 2 aufeinanderfolgenden Phasen finden nun statt

Einbindung der **Validierung** der Phasenergebnisse, d.h. Prüfung auf Vollständigkeit und Konsistenz der Anforderung sowie Prüfung der technischen Durchführbarkeit

dennoch: jede Phase wird vollständig bearbeitet, bevor die Bearbeitung der nächsten beginnt

5. Lebenszyklusmodelle – klassische Modelle

V - Modell



5. Lebenszyklusmodelle – klassische Modelle

V - Modell

Im Gegensatz zum Wasserfallmodell fällt auf, dass es nicht nur eine Testphase gibt, sondern mehrere.

Die waagrechten Verbindungslinien ordnen jeder Phase der Softwareerstellung eine spezielle Testphase zu

Fehler, die in einer Konstruktionsphase begangen werden, lassen sich am leichtesten in der zugehörigen Prüfphase finden.

Ferner repräsentieren die auseinanderstrebenden Schenkel des V die steigenden Korrekturkosten bei später entdeckten Fehlern.

5. Lebenszyklusmodelle – klassische Modelle

Schwächen klassischer Lebenszyklusmodelle

- Wegen der sequentiellen Abarbeitung aller Phasen wird *bei Termindruck am ehesten der Test gekürzt*
- Der *Kunde tritt nur am Anfang und am Ende* der Softwareentwicklung in Erscheinung. Dadurch besteht die Gefahr, dass sich das Produkt über die gesamte Entwicklungszeit vom Kundenwunsch weit entfernt
- Die Kosten zur Beseitigung der Fehler wachsen nicht linear mit der Dauer, die sie im Produkt verbleiben, sondern sie *vervielfachen sich von Phase zu Phase*

5. Lebenszyklusmodelle – neuere Modelle

Aspekte neuerer Lebenszyklusmodelle

Meist werden Lebenszyklusmodelle nicht vollständig neu definiert, sondern es wird ein neuer oder zusätzlicher Aspekt mit bereits existierenden Modellen kombiniert.

5. Lebenszyklusmodelle – neuere Modelle

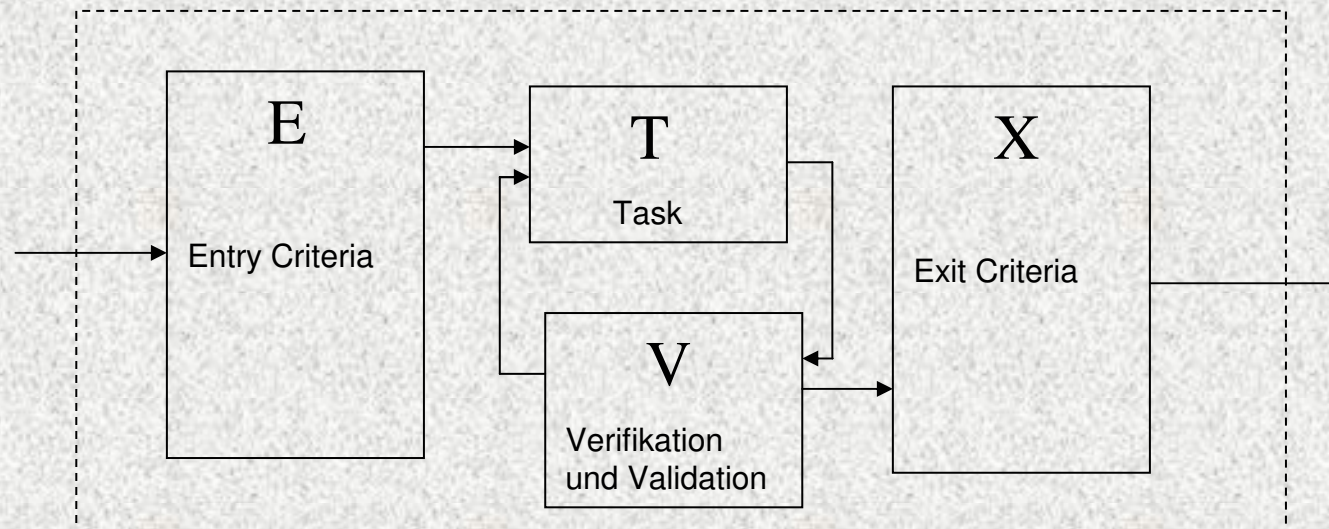
Aspekte neuerer Lebenszyklusmodelle

Bereits 1985 wurde ein sog. *Elementarprozessschema* zur Definition von Lebenszyklusmodellen publiziert. Es wird häufig *als Elementarbaustein für die aktuellen Lebenszyklusmodelle* eingesetzt:

5. Lebenszyklusmodelle – neuere Modelle

ETVX

Bereits 1985 wurde ein sog. *Elementarprozessschema* zur Definition von Lebenszyklusmodellen publiziert. Es wird häufig als Elementarbaustein für die aktuellen Lebenszyklusmodelle eingesetzt:



5. Lebenszyklusmodelle – neuere Modelle

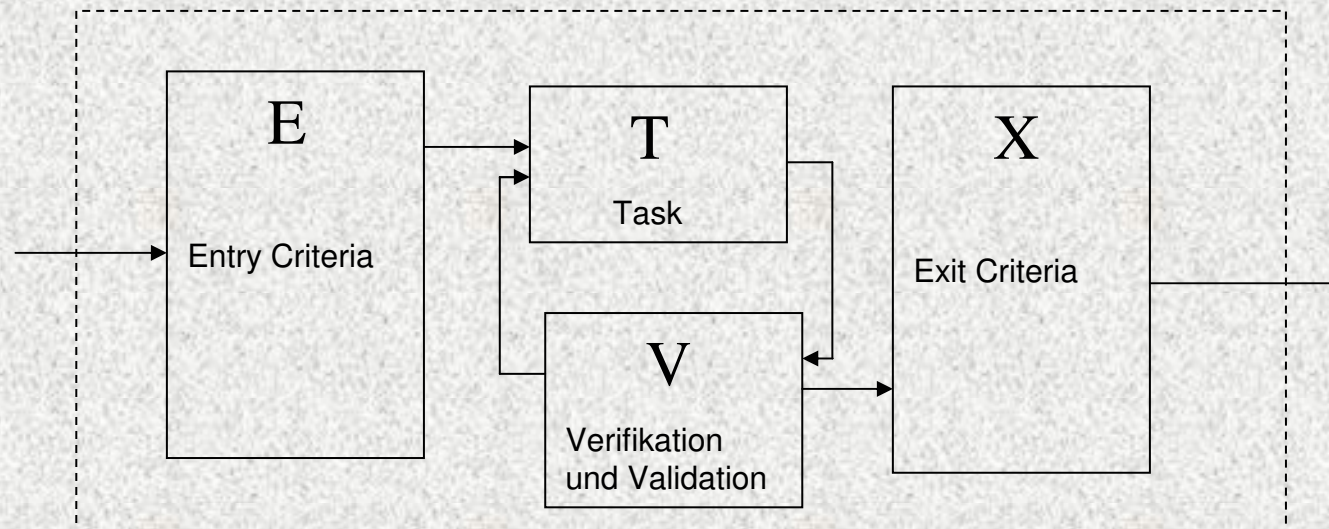
ETVX

Es werden zunächst alle Eingangskriterien geprüft. Wenn sie erfüllt sind, wird die eigentliche Aufgabe durchgeführt.

Die Qualitätssicherung der Ergebnisse findet in der Verifikation und Validierung statt.

Bei Erfüllung der Ausgangskriterien wird dieser Prozess beendet.

Das Wesentliche ist die Idee, mit jedem Arbeitsschritt untrennbar eine Qualitätssicherungsphase zu verbinden!



5. Lebenszyklusmodelle – neuere Modelle

ETVX

Verifikation

mittels Verifikation stellt man fest, ob

- die Entwicklungsschritte richtig ausgeführt wurden
- die Eingabedokumente korrekt und vollständig weiterbearbeitet wurden
- die vorgesehenen Techniken richtig angewendet wurden

Validierung

mittels Validierung überprüft man, ob der aktuelle Entwicklungsstand daraufhin ausgerichtet ist, die Kundenanforderungen zu erfüllen

5. Lebenszyklusmodelle – neuere Modelle

ETVX

Vorteil

die in das ETVX-Schema integrierte Qualitätssicherung stellt bei korrekter Ausführung sicher, dass Fehler früh gefunden werden und dadurch noch relativ kostengünstig behoben werden können.

Nachteil

Da der Kunde allerdings nur am Anfang und am Ende der Entwicklung in Erscheinung tritt, muss man die dokumentierten Anforderungen des Kunden immer wieder zum Abgleich der Zwischenprodukte heranziehen!
Sich ändernde Kundenanforderungen werden damit allerdings nicht berücksichtigt

5. Lebenszyklusmodelle – neuere Modelle

Inkrementelle Softwareentwicklung

Spiralmodell

die heute weit verbreitete inkrementelle Softwareentwicklung basiert auf dem Spiralmodell von Boehm. Dieses Modell hat sich als viel wirklichkeitsnäher herausgestellt.

Im Spiralmodell folgen die Phasen nicht linear aufeinander, sondern wiederholen sich mehrfach reihum.

Mit jedem Umlauf nähert man sich dem angestrebten Endprodukt an.

5. Lebenszyklusmodelle – neuere Modelle

Inkrementelle Softwareentwicklung

Spiralmodell

zuerst wird ein für den Kunden attraktives Teilsystem erstellt. Es durchläuft die Phasen Planung, Entwicklung, Integration und Qualitätssicherung.

Dann wird das System schrittweise durch neue Funktionalitäten erweitert. Dabei wird in jeder Umdrehung der Spirale jeweils eine neue Erweiterung bearbeitet, und zwar mit allen Phasen, die dazu gehören. Der Kunde ist bei jedem Inkrement beteiligt.

→ dadurch wird gewährleistet, dass Unzufriedenheiten oder Fehler den Entwicklern früh gemeldet und beseitigt werden können

5. Lebenszyklusmodelle – neuere Modelle

Inkrementelle Softwareentwicklung

Spiralmodell

Das Spiralmodell fasst den Entwicklungsprozess als iterativen Prozess auf, wobei jeder Zyklus folgende Aktivitäten enthält:

- Festlegung von Zielen, Alternativen und Rahmenbedingungen (Einsatzbereich, Funktionalität, Modifizierbarkeit etc.)
- Alternativen zur Realisierung des Teilprodukts
- Erkennen und Reduzieren von Risiken
- Realisierung und Überprüfung (Validierung) des Zwischenprodukts
- Planung der Projektfortsetzung.

Am Ende jeder Windung steht ein Review, in dem der Projektfortschritt bewertet wird. Zudem wird der nächste Projektfortschritt geplant und verabschiedet.

5. Lebenszyklusmodelle – neuere Modelle

Inkrementelle Softwareentwicklung

Spiralmodell

- Die radiale Ausdehnung stellt den Gesamtaufwand, der bis zu einem bestimmten Zeitpunkt geleistet wurde, dar.
- Die Winkeldimension beschreibt den Projektfortschritt in den einzelnen Spiralenzyklen
- Hauptziel ist es, dass sich alle Beteiligten über das bisher Geleistete und das im nächsten Zyklus Geplante einig sind.
- Fehler können möglichst früh erkannt und Lösungsalternativen können erwogen werden.
- Qualitätssicherungsmaßnahmen werden in den Prozess mit eingebettet!
- Die Softwareentwicklung wird als kontinuierlicher Wartungsprozess angesehen.

5. Lebenszyklusmodelle – neuere Modelle

Prototyp-Orientiertes Modell

Dabei wird ein aussagefähiges Muster der Anwendung erstellt, bevor die eigentliche Softwarelösung implementiert wird.

Die Phaseneinteilung bleibt zwar erhalten mit dem Unterschied, dass Problemanalyse und Spezifikation zeitlich stark überlappt ablaufen und Entwurf, Implementierung und Test stark ineinander verschmelzen.

Bei Anwendung der prototyp-orientierten Methode *wird so früh wie möglich implementiert* (beim streng sequentiellen Modell hingegen so spät wie möglich), um so mögliche Missverständnisse bei der Beschreibung der Anforderungen aufzudecken.

5. Lebenszyklusmodelle – neuere Modelle

Prototyp-Orientiertes Modell

z.B. wird *bereits während der Spezifikationsphase* ein Prototyp der Benutzerschnittstelle entwickelt. Anhand dieses Prototyps wird durch Experimente, die den realen Einsatzbedingungen entsprechen, untersucht, ob die Anforderungen des Anwenders erfüllt werden.

Die Praxis hat gezeigt, dass man viel eher zum Ziel kommt, wenn man die Systemspezifikation und –Architektur *an Hand eines Modells entwickelt*, anstatt durch bloße Beschreibung. Damit wird das Risiko einer falschen oder unvollständigen Systemspezifikation vermindert und somit eine wesentlich bessere Ausgangsbasis für die folgenden Aktivitäten geschaffen.

5. Lebenszyklusmodelle – neuere Modelle

aktuelle Schlagworte

Der Unified Process

Darunter versteht man die Entwicklung und Standardisierung der Unified Modeling Language (*UML*) für die *objektorientierte Softwareentwicklung*.

Die UML ist eine große Anzahl von verschiedenen Diagrammarten, die im Laufe der Systemerstellung von der Anforderungsanalyse bis zum detaillierten Entwurf, zur Dokumentation des Systems verwendet wird.

Der Unified Process ist ebenfalls ein inkrementeller Prozess. Jeder Zyklus der Inkremente erweitert die Funktionalität des Systems und verbessert seine Qualität.

Im Unified Process *werden die Anforderungen durch Anwendungsfälle* beschrieben.

5. Lebenszyklusmodelle – neuere Modelle

aktuelle Schlagworte

Agile Methoden: Extreme Programming

Agile Methoden zielen darauf hin, dem Kunden schnell einsetzbare Software zu liefern, die *inkrementell in engem Kontakt mit dem Kunden* ausgebaut wird.

Ein Beispiel der agilen Methoden Entwicklungsmethoden ist das Extreme Programming.

Dazu gehören u.a. die folgenden Praktiken:

5. Lebenszyklusmodelle – neuere Modelle

aktuelle Schlagworte

Agile Methoden: Extreme Programming

Permanenter Test durch automatisierte Tests:

Hierfür gibt es für verschiedene Programmiersprachen Werkzeuge (z.B. JUnit für Java).

Testfälle werden bereits vor der Codierung beschrieben, die dann die zu erstellenden Module in ihrer Funktion testen sollen.

Die Implementierung des Moduls gilt als beendet, wenn alle Tests erfolgreich durchlaufen werden.

Kritik:

dabei können durchaus wichtige Testfälle übersehen werden.

Die Tests vor der Implementierung ersetzen keinesfalls die Tests nach der Implementierung

5. Lebenszyklusmodelle – neuere Modelle

aktuelle Schlagworte

Agile Methoden: Extreme Programming

Kunde vor Ort:

Um die Anforderungen des Kunden möglichst genau zu verstehen und hinterfragen zu können, steht ein Vertreter des Kunden vor Ort bei den Entwicklern zu Verfügung.

Kritik:

hohe Kosten

ein einziger Kundenvertreter kann gar nicht alle Anforderungen an ein Produkt vertreten

5. Lebenszyklusmodelle – neuere Modelle

aktuelle Schlagworte

Agile Methoden: Extreme Programming

Anforderungen als Benutzungsbeispiele:

Der Kunde formuliert seine Anforderungen als Benutzungsbeispiele (use-cases). Dadurch kann der Kunde seine Sprache verwenden und die Entwickler gewinnen eine anschauliche Vorstellung für die zu realisierende Funktionalität

Inkrementelles Vorgehen:

Ein Inkrement der Softwareerstellung entspricht der Umsetzung eines Benutzungsbeispiels.

5. Lebenszyklusmodelle – neuere Modelle

aktuelle Schlagworte

Agile Methoden: Extreme Programming

Paar-Programmierung:

Dabei arbeiten zwei Programmierer gemeinsam an einem Bildschirm und mit einer Tastatur. Wechselweise übernimmt einer die Initiative und der andere verfolgt kritisch das Geschehen. Dadurch soll eine besonders intensive Arbeit gewährleistet sein, da keiner der beiden den anderen behindern möchte. Außerdem sei die erste Qualitätssicherung unmittelbar in den Produktionsprozess integriert.

Kritik:

Personalverschwendung

5. Lebenszyklusmodelle – neuere Modelle

aktuelle Schlagworte

Agile Methoden: Extreme Programming

Gemeinsame Verantwortlichkeit:

Das ganze Team ist gemeinsam für den Code verantwortlich. Entdeckt jemand eine Verbesserungsmöglichkeit, so wird sie mit dem Partner besprochen und ggf. direkt realisiert. Durch die automatisierten Tests sieht man sofort, wenn irgendwo Probleme auftreten.

Fortlaufende Integration:

Durch eine fortlaufende Integration mehrmals täglich, gibt es praktisch immer eine aktuelle lauffähige Version.

5. Lebenszyklusmodelle – Zusammenfassung

Zusammenfassend lässt sich feststellen, dass sich der Einsatz von Vorgehensmodellen als Richtlinie zu Software-Entwicklung in der Praxis bewährt hat. Folgende Vorteile ließen sich bislang ableiten:

- Normierung und Vereinheitlichung des Entwicklungsprozesses und der Ergebnisse
- Personenunabhängigkeit
- Anleitung und Hilfestellung zur Lösung komplexer Entwicklungsaufgaben
- Basis für effiziente Programmierung

5. Lebenszyklusmodelle

Werkzeuge

der Softwareentwicklungsprozess soll werkzeugunterstützt betrieben werden

die Klasse von Werkzeugen (Tools), die hierbei eine Rolle spielen, sind die sog. **CASE-Werkzeuge** (Computer Aided Software Engineering)

Idealerweise werden durch ein CASE-Werkzeug die gesamten Arbeitsschritte der Softwareentwicklung unterstützt und insbesondere *Konsistenzprüfungen* zwischen einzelnen Projektdokumenten vorgenommen. Dies bedeutet, dass die einzelnen Entwicklungsdokumente in sich widerspruchsfrei sein müssen.

6. Projektablauf der Pflichtenhefterstellung

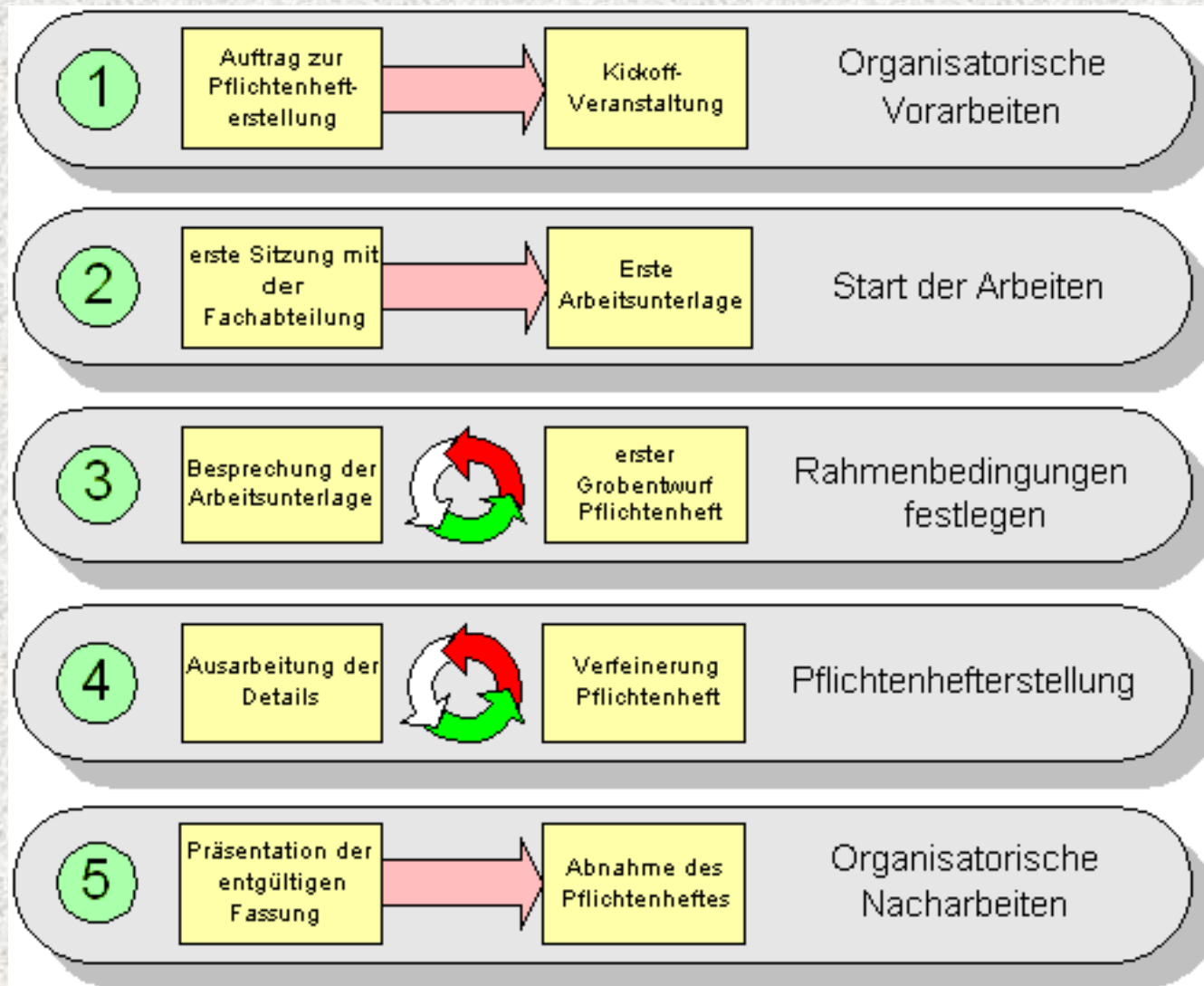
Worum geht es?

Zur Erstellung eines Pflichtenheftes sind neben einer bewährten Methode zur Umsetzung von Anforderungen aus den Fachabteilungen *zusätzliche organisatorische Maßnahmen* notwendig.

Diese Maßnahmen helfen, den Erfolg der Arbeit sicherzustellen und bilden die *Grundlage für die Zusammenarbeit* zwischen Berater und Fachabteilung.

6. Projektablauf der Pflichtenhefterstellung

Die Erstellung eines Pflichtenheftes gliedert sich wie folgt:



6. Projektablauf der Pflichtenhefterstellung

Der Auftrag zur Erstellung eines Pflichtenheftes

Worum geht es?

Das *Pflichtenheft* kann zum einen Teil einer umfassenderen Aufgabe sein. Zum anderen kann die Erstellung auch *ein eigenes Projekt* sein.

Der Autor des Pflichtenheftes sollte seine Arbeit wie ein Projekt organisieren. Dies hat den Vorteil das ihm alle Mechanismen des *Projektmanagements* zur Verfügung stehen.

6. Projektablauf der Pflichtenhefterstellung

Der Auftraggeber initiiert die Erstellung des Pflichtenheftes.

Der *Autor des Pflichtenheftes* ist in diesem Fall wie ein Projektleiter anzusehen. Er übernimmt die *Rolle des Gesamtverantwortlichen*.

Beide Seiten haben während der Erstellung des Pflichtenheftes bestimmte Aufgaben und Erwartungen. Diese gilt es zu dokumentieren.

6. Projektablauf der Pflichtenhefterstellung

Der *Auftrag* ist eine *Vereinbarung* zwischen dem *Auftraggeber* und dem *Autor des Pflichtenheftes*.

Dort werden die Rahmenbedingungen definiert, unter denen das Pflichtenheft erstellt werden soll.

Der Auftrag dient für beide Seiten auch als *Meßplatte* für die *erfolgreiche Erstellung des Pflichtenheftes*. Allein die Fertigstellung des Pflichtenheftes innerhalb eines definierten Zeitraumes ist kein Indiz für die erfolgreiche Fertigstellung. Das Pflichtenheft ist erst dann fertig, *wenn alle Ziele erreicht worden sind*.

6. Projektablauf der Pflichtenhefterstellung

Häufig kommt es vor, dass der *Auftraggeber nicht von selbst auf die Idee* kommt, für die Erstellung eines Pflichtenheftes einen Auftrag zu definieren.

Am einfachsten ist es dann, den *Entwurf des Auftrages vorzubereiten*. Dies hat zwei Vorteile.

Zum einen hat der Autor des Pflichtenheftes so die Möglichkeit, seine Sicht der Dinge zu formulieren.

Zum anderen spart sich der Auftraggeber die Mühe, selbst einen Auftrag zu schreiben.

6. Projektablauf der Pflichtenhefterstellung

Was bringt es?

Der *Auftrag* schafft *Sicherheit auf beiden Seiten*. Dem Auftraggeber ist es auf diese Weise möglich, alles das was er von dem Pflichtenheft wünscht, als Teil des Auftrages zu definieren. So ist schon vor Beginn der Arbeit klar festgelegt, was von dem Autor erwartet wird. Probleme mit Nachforderungen oder Unzufriedenheit nach oder während der Erstellung des Pflichtenheftes sollten so fast ausgeschlossen sein.

Der Autor eines Pflichtenheftes hat mit Hilfe des Auftrages die Möglichkeit, die *Rahmenbedingungen abzustecken*, die als Voraussetzung für einen möglichst reibungslosen Verlauf aus seiner Sicht notwendig sind.

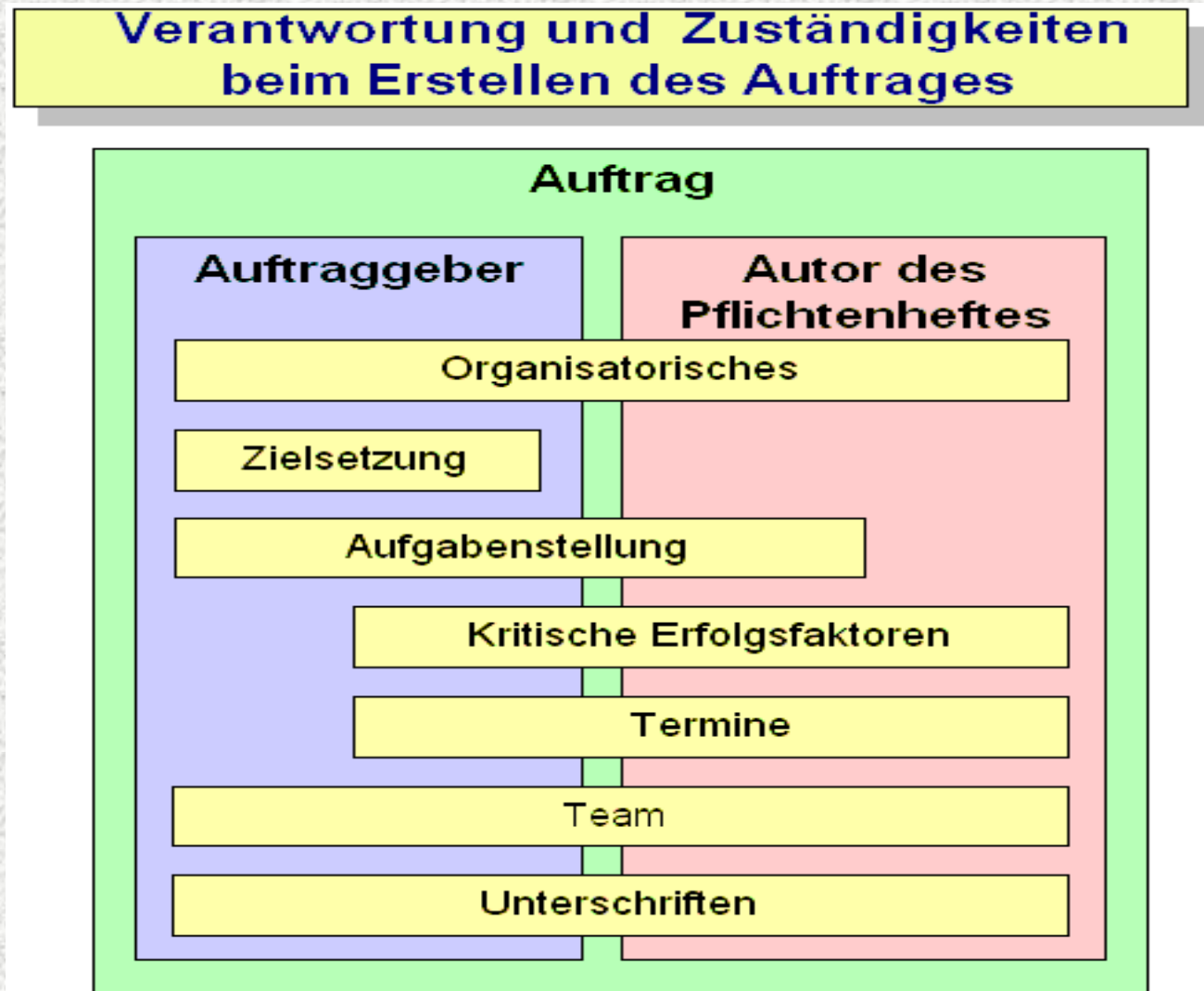
6. Projektablauf der Pflichtenhefterstellung

Dem Autor eines Pflichtenheftes dient der *Auftrag als Leitlinie* während der gesamten Zeit der Pflichtenhefterstellung. Anhand des Auftrages kann der Autor zu jeder Zeit überprüfen, ob die vereinbarten Ziele erreicht werden.

Ziel des Auftrages ist es, *Sicherheit zwischen den beiden Partnern schaffen*. Immer dann, wenn diese Sicherheit nicht mehr gegeben ist, sollte der Auftrag geändert werden. Darunter fällt jedoch nicht jede Kleinigkeit.

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages



6. Projektablauf der Pflichtenhefterstellung

Wie wird es gemacht?

Der Auftrag zur Erstellung eines Pflichtenheftes beginnt mit einem organisatorischen Teil. Wie bei anderen Arten von Verträgen wird zu Beginn der Gegenstand des Vertrages und die Vertragspartner genannt.

Der *Titel* des Pflichtenheftes ist die Arbeitsbezeichnung unter der die gesamte Pflichtenhefterstellung durchgeführt wird. Er sollte den Gegenstand des Pflichtenheftes möglichst genau wiedergeben. Hierbei sollte der Titel nicht zu lang sein. Neben dem Titel ist es sinnvoll für das Pflichtenheft ein *Kürzel* zu vergeben.

6. Projektablauf der Pflichtenhefterstellung

Beispiele:

- Neue Wareneingangssteuerung
- Das Kundenbindungsprogramm
- Spesenabrechnung für den Außendienst
- Automatische Rechnungsprüfung

6. Projektablauf der Pflichtenhefterstellung

Neben dem Gegenstand des Pflichtenheftes müssen unbedingt *die beiden Partner* im Auftrag genannt werden, Auftraggeber und Autor.

Neben den beiden Partnern sind bei vielen Pflichtenheften *zusätzliche Ressourcen* notwendig.

Als Team werden dann entweder eigene Mitarbeiter oder externe Berater benannt. In beiden Fällen ist die Festlegung des Teams, das bei der Erstellung des Pflichtenheftes mitarbeiten soll ein *notwendiger Bestandteil* des Auftrages.

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages - Zielsetzung

Wie wird es gemacht?

In diesem Teil geht es um die Frage, *was eigentlich mit dem Pflichtenheft erreicht werden soll.*

Ergeben sich bei der Zielsetzung Verständnisprobleme, müssen diese gemeinsam mit dem Auftraggeber ausgeräumt werden.

Zudem muss sichergestellt werden, dass die definierten Ziele bestimmten Anforderungen genügen.

Die Anforderungen an die formulierten Ziele sind:

6. Projektablauf der Pflichtenhefterstellung

die formulierten Ziele sind:

Erreichbarkeit :

Die formulierten Ziele müssen erreicht werden können. Dies sollte gemeinsam mit dem Auftraggeber abgeklärt werden.

6. Projektablauf der Pflichtenhefterstellung

die formulierten Ziele sind:

Vollständigkeit

Alle zu erreichenden Ziele müssen niedergelegt werden. Es dürfen später im Pflichtenheft keine neuen Ziele mehr definiert werden, die nicht Teil des Auftrages sind.

6. Projektablauf der Pflichtenhefterstellung

die formulierten Ziele sind:

Verständlichkeit

Eine klare und eindeutige Zielformulierung ist unabdingbar für die zielgerichtete Erstellung eines Pflichtenheftes. Nur wenn die zu erreichenden Ziele verständlich sind, können alle die mitarbeiten, auch helfen, diese zu erreichen.

6. Projektablauf der Pflichtenhefterstellung

die formulierten Ziele sind:

Messbarkeit

Die formulierten Ziele müssen erreicht werden können. Dies sollte gemeinsam mit dem Auftraggeber abgeklärt werden.

6. Projektablauf der Pflichtenhefterstellung

Beispiele :

Falsch - nicht überprüfbar	Richtig - überprüfbar
Die neue Lagerverwaltungs-Software soll effektiv eingesetzt werden.	Die neue Lagerverwaltungs-Software soll eine Reduzierung der Personalkapazität um 10% ermöglichen.
Der gesamte Einkauf soll mit der neuen Software rationalisiert werden .	Der gesamte Einkauf soll mit der neuen Software Einsparungen von mind. 5.000,-- DM pro Jahr realisieren.
Die Erfassung von Aufträgen soll komfortabler werden .	Die Erfassung der Aufträge soll sich auf max. 2 Min. pro Auftrag reduzieren .

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages - Aufgabenstellung

Wie wird es gemacht?

Hier geht es darum, die Mittel und Wege zu definieren, mit denen die festgelegten Ziele erreicht werden können. Neben den Zielen ist dieser Punkt im Auftrag ein wesentlicher Faktor für die Pflichtenhefterstellung.

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages - Kritische Erfolgsfaktoren

Wie wird es gemacht?

Der Autor des Pflichtenheftes hat die Aufgabe, alle aus seiner Sicht erfolgskritischen Faktoren aufzuzählen.

Auf diese Weise kann der Auftraggeber helfen, evtl. auftretende Probleme zu überwinden.

6. Projektablauf der Pflichtenhefterstellung

Beispiele:

termingerechte und ausreichende Mitarbeit der Fachabteilungen:

Dieser Punkt ist der Hauptgrund, warum die Erwartungen an ein Pflichtenheft häufig nicht erfüllt werden.

Teammitglieder, die sich aus Mitarbeitern der Fachabteilungen zusammensetzen sind vielfach mit der zusätzlichen Arbeit an dem Pflichtenheft überlastet. Dies ist immer dann der Fall, wenn diese Arbeit neben dem eigentlichen Tagesgeschäft auf diese Mitarbeiter zukommt.

Der Auftraggeber soll hier helfen, dass die benötigten Mitarbeiter entsprechend freigestellt werden.

6. Projektablauf der Pflichtenhefterstellung

Beispiele:

ausreichende Ausstattung mit Arbeitsmitteln:

Während der Arbeit an einem Pflichtenheft werden permanent bestimmte Arbeitsmittel benötigt.

Alle diese "Kleinigkeiten" können Terminverzögerungen mit sich bringen. So werden z. B. gemeinsame Besprechungen angesetzt und es ist kein Raum frei.

Abläufe sollen der Fachabteilung präsentiert werden, jedoch fehlt leider der Projektor.

Mit Hilfe des Auftrages wird sichergestellt, dass der Autor Zugriff auf alle notwendigen Arbeitsmittel hat, bzw. dass der Auftraggeber die Bereitstellung dieser Mittel sicherstellt.

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages - Termine

Wie wird es gemacht?

Die Terminplanung ist bei der Pflichtenhefterstellung ein schwieriges Thema. Nicht umsonst ist ein Festpreisangebot für eine Pflichtenhefterstellung kaum einzuhalten. Der Grund liegt in der schlechten Abwägbarkeit der Aufwände bei der Pflichtenhefterstellung. Dieser Prozess wird zum einen von der Art der Zusammenarbeit zwischen Fachabteilung und Autor bestimmt, zum anderen von der Komplexität des zu bearbeitenden Themas.

Aus diesem Grunde ist eine exakte Terminplanung für eine Pflichtenhefterstellung nur in Spezialfällen möglich.

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages - Termine

Um überflüssige Diskussionen zu vermeiden, kann man in diesem Teil folgende Struktur vorgeben:

- Start der Pflichtenhefterstellung: TT.MM.JJJJ
- Vorlage eines ersten Grobkonzeptes: TT.MM.JJJJ
- geplante Abnahme durch: TT.MM.JJJJ

Statt genauer Termine kann auch eine Kalenderwoche als Termin genannt werden.

Der Auftrag darf auf keinen Fall eine zu detaillierte Zeitplanung enthalten! Der Zeitpunkt dafür ist bei Auftragserteilung noch viel zu früh.

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages – Das Team

Wie wird es gemacht?

Neben dem Auftraggeber und dem Autor sind meistens bei der Erstellung eines Pflichtenheftes noch weitere Mitarbeiter tätig.

Zur Absicherung dieser notwendigen Ressourcen sollten diese Mitarbeiter Teil des Auftrages sein.

Auf diese Weise wird sichergestellt, dass alle gemeinsam mit dem Auftraggeber benannten Personen fester Bestandteil des Teams werden.

6. Projektablauf der Pflichtenhefterstellung

Die Struktur des Auftrages - Unterschriften

Wie wird es gemacht?

Der Auftrag sollte auf jeden Fall eine Unterschrift enthalten. Ein einfaches OK reicht hierfür nicht aus.

Erst mit den Unterschriften wird der Auftrag besiegelt.

Die Unterschriften sollten daher möglichst zeitnah nach Fertigstellung des Auftrages.

6. Gliederungsstruktur eines Pflichtenhefts

Zielbestimmung

- Musskriterien
- Wunschkriterien
- Abgrenzungskriterien

Produkteinsatz

- Anwendungsbereich
- Zielgruppen
- Betriebsbedingungen

Produkt-Umgebung

- Software
- Hardware
- Orgware
- Produkt-Schnittstellen

Produkt-Funktionen

Produkt-Daten

Produkt-Leistungen

Benutzeroberfläche

Qualitäts-Zielbestimmung

Globale Testszenarien / Testfälle

Entwicklungsumgebung