Software Engineering

Informatik II.

Zusammenfassung

– Kapitel 1. – 5. –

Dipl.-Inform. Hartmut Petters

- Mit dem Begriff „Software-Engineering“ werden alle Aktivitäten, Prinzipien, Methoden und Werkzeuge zusammengefasst, die zur Erstellung komplexer Software-Systeme verwendet werden.
- Da Software-Entwicklung und somit auch Software-Engineering eine noch recht „junge“ Disziplin und Wissenschaft ist, sind die grundlegenden Erfahrungen seit ca. Mitte der 60er Jahre gemacht worden und haben nach und nach den inhaltlichen Rahmen des Begriffs „Software-Engineering geprägt.
- Die Vorlesung hat die Zielsetzung die Hintergründe dieser Vorgehensweise zu beleuchten, die Prinzipien, Methoden und Werkzeuge darzustellen und das Bewusstsein bzgl. einer systematischen Software-Entwicklung zu stärken, Vorteile dieses methodischen Vorgehens darzustellen und näher zu bringen.

Vorwort – was ich noch zu sagen hätte …

Basis dieser Vorlesung sind vor allem die folgenden Ausarbeitungen

- Vorlesungsskript „Software Engineering"
  von Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/ses/
- Vorlesungsskript „Informatik II – Software Engineering"
  von Frau Prof. Dr. Kühn FH Karlsruhe FB W
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Das Buch „Software Engineering" 6. Auflage/2001
  von Prof. Ian Sommerville University of Lancaster (UK)
  Addison Wesley **ISBN 3-8273-7001-9**

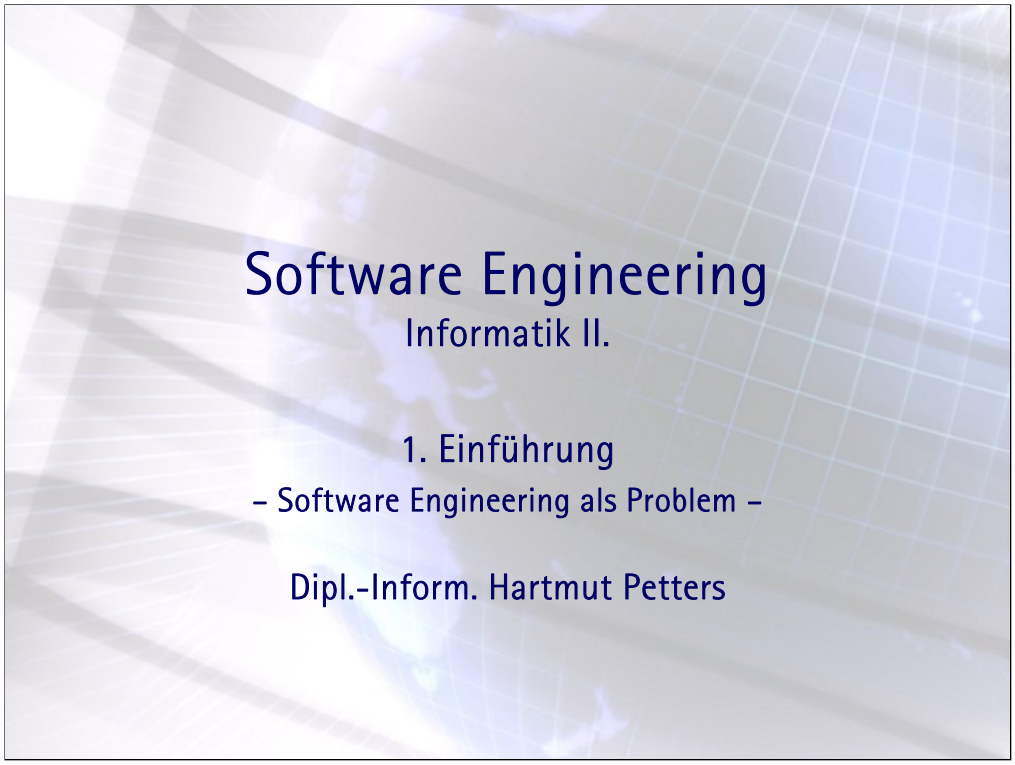
Konkret entnommene Beiträge sind i.d.R. mit einem Quellen-Verweis gekennzeichnet – sollte dieser fehlen bitte ich um Nachsicht.

Den „roten Faden" durch die Vorlesung habe ich dem Skript der Vorlesung von Prof. Dr. Martin Glinz entnommen und um eigene Beiträge erweitert bzw. aus den beiden anderen Quellen ergänzt.

Für die Möglichkeit der Verwendung der wesentlichen Inhalte möchte ich mich an dieser Stelle bei den Autoren herzlich bedanken.

Da diese Vorlesung innerhalb kürzester Zeit augebaut wurde, war mir dies nur möglich, indem ich auf bereits bewährtes Material zurückgreifen konnte und die oben genannten Autoren ihre Ausarbeitungen großzügiger Weise zur Verfügung gestellt haben.

Hierfür möchte ich mich ausdrücklich bedanken.



Im 1. Kapitel habe ich die Grundproblematik dargestellt, die sich mit der Entwicklung komplexer Software-Systeme ergibt.

Was ist Software

- Software (Definition nach Roger S. Pressman)
  1. Instruktionen, die bei der Ausführung auf einem Computer gewünschte Funktionen und Leistungen hervorbringen
  2. Datenstrukturen, die eine adäquate Informationsverarbeitung durch die Programme ermöglichen
  3. Dokumente, die Operationen und Benutzung der Programme beschreiben

Was ist Software?

- Software als Wirtschaftsgut ist heute fester Bestandteil unseres Lebens, allerdings ist es aufgrund seiner immateriellen Struktur kaum fassbar und nur sehr weitschweifig zu erläutern.
- Eine grundsätzliche Definition liefert Roger S. Pressman, in dem er darstellt, dass Software im Wesentlichen aus folgenden 3 Elementen besteht

1. Instruktionen
2. Datenstrukturen
3. Dokumenten

Diese Umschreibung ist deswegen notwendig, da die eigentliche Natur von Software nicht direkt, sondern nur indirekt über das nach außen dargestellte Systemverhalten – d.h. beobachtbar – ist.

Software-Entwicklung

- Software-Entwicklung
  Ist die Umsetzung der Bedürfnisse von Benutzern in Software. Umfasst Spezifikation der Anforderungen, Konzept der Lösung, Entwurf und Programmierung der Komponenten, Zusammensetzung der Komponenten und ihre Einbindung in vorhandene Software, Inbetriebnahme der Software sowie Überprüfung des Entwickelten nach jedem Schritt.
- Verständnis der Probleme ist grundlegend wichtig
  - Klein =/= Groß
  - Aufwand steigt überproportional mit Produktgröße
  - Software ist einer Evolution unterworfen
  - Software wird von Menschen gemacht
  - Software kann verschleißen

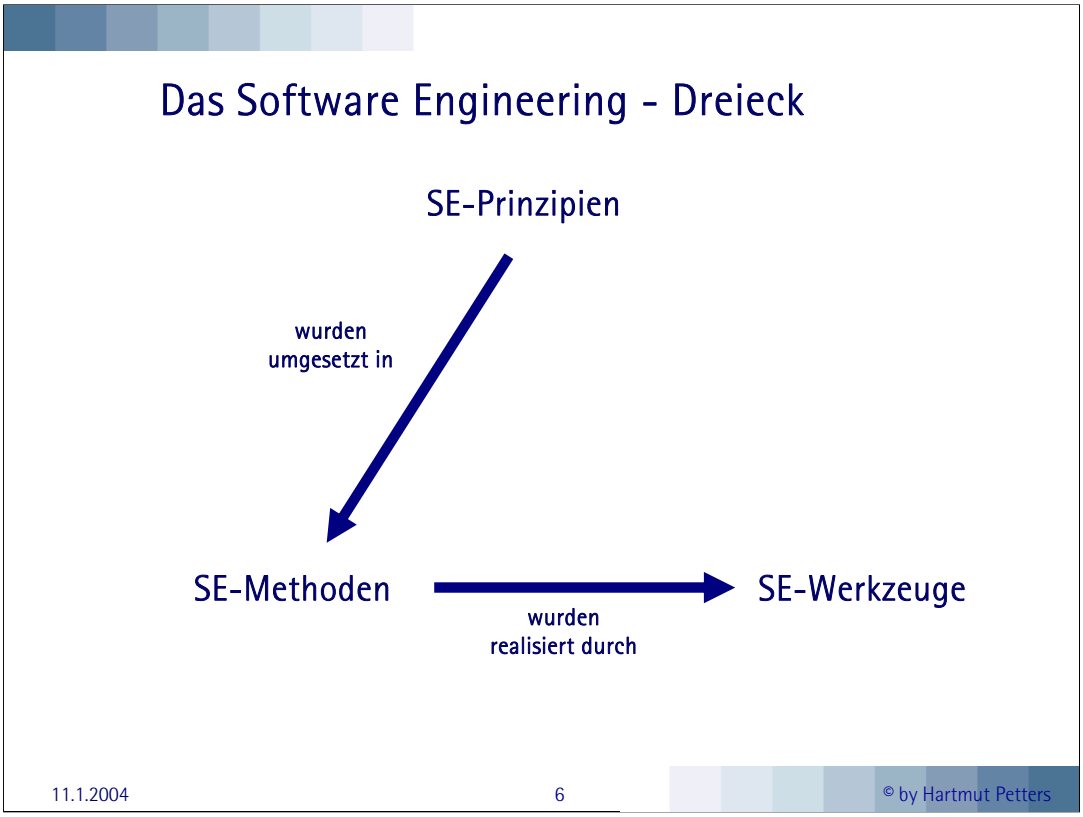
11.1.2004 5

Software-Entwicklung – der Prozess, der letztendlich dazu dient das Produkt „Software“ zu erzeugen – umfasst dabei die Phasen

- Spezifikation der Anforderungen
- Entwurf des Lösungskonzepts
- Detail-Entwurf + Detail-Spezifikation
- Programmierung + Umsetzung
- Test + Integration der Komponenten
- Test + Integration der Komponenten zum System
- Test + Integration des Systems in die bestehende Systemlandschaft / Inbetriebnahme
- Abnahme jeder einzelnen Phase und den damit verbundenen Dokumenten

Grundsätzlich ist zu verstehen, dass große Unterschiede bestehen bei der Entwicklung von

- Kleinen Projekten vs. großen Projekten
- Die Komplexität der Realisierung überproportional steigt
- Software ebenso wie alles andere einer Evolution und damit einer kontinuierlichen Veränderung unterworfen ist und damit auch verschleißt.



Basis des heutigen Software-Engineering ist die Entstehung der Software-Engineering Prinzipien ende der 60er sowie in den 70er Jahren, als man erkannt hat, dass die Entstehung der Software nicht einem willkürlichen Prozess unterworfen sein darf, sondern nach klaren Maßgaben ingenieurmäßig umgesetzt werden muß.

Aufgrund der Software-Engineering-Prinzipien entstanden daraus in den 80er Jahren die daraus abgeleiteten Methoden, die klare Rahmenvorgaben für die Umsetzung der Prinzipien aufzeigten.

Aber erst mit dem Entstehen von mächtigen Werkzeugen zur Realisierung dieser Methoden und den darin manifestierten Prinzipien war es möglich dies auch sinnvoll und wirtschaftlich in den Entwicklungsprozess einzubeziehen.

Generell wurde durch diese Entwicklung ein großer Schritt in Richtung qualitativ hochwertiger Software-Produkte getan, da die bisher eher heuristischen Methoden durch Systematik und nachvollziehbare Schritte ersetzt wurden.

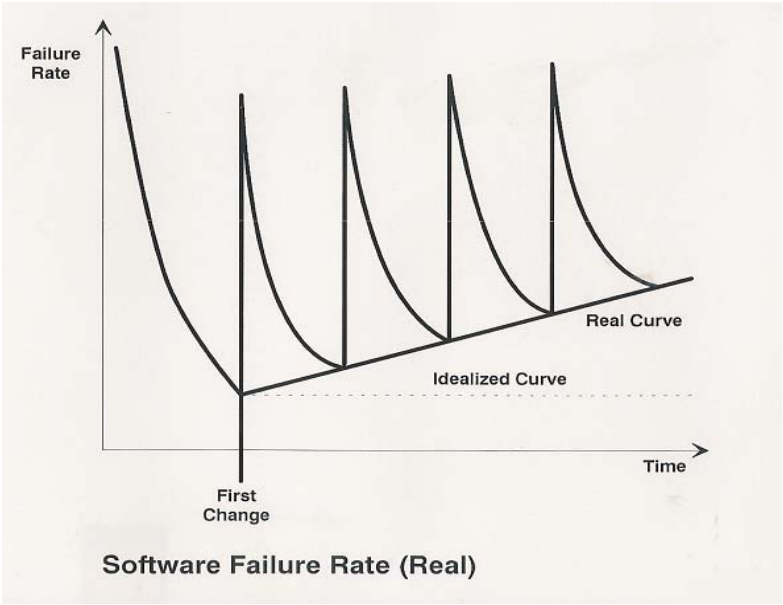
Was gehört zu Software Engineering?

- Idee – Was möchte ich machen?
- Entwurf – Wie soll es gemacht werden?
- Planung – Was und wen brauche ich dazu?
- Planung – Zeit, Budget, Menschen
- Doing – Programmierung / Integration
- Doing – Dokumentation
- Testing – Funktionsprüfung
- Testing – Fehlerbeseitigung
- Testing – Benutzerakzeptanz
- Rollout – Einführung + Schulung
- Pflege – Weiterentwicklung + Anpassung

Auf der Basis der verfügbaren Prinzipien, Methoden und Werkzeuge konnten nun alle Basiselemente in den Software-Entstehungsprozess eingebracht werden, wie

- Die Zusammenfassung der Ideen in der Spezifikation
- Dem Grob- und Detailentwurf der Lösung als Lösungskonzept
- Eine grundlegende Planung der Aktivitäten zum Abgleich
- Berücksichtigung der Vorgaben (Kosten, Zeit, Funktionalität) bei der Planung bis hin zur Nachkalkulation
- Programmierung, Test, Integration und Dokumentation können als integrale Bestandteile des Software-Entwicklung geplant, kontrolliert und gesteuert werden
- Ebenso ist die Systemeinführung, die Rollout-Phase Bestandteil dieses Prozesses
- System-Weiterentwicklung, Pflege und Anpassung an sich verändernde Randbedingungen ist langfristig kalkulierbar.

Software Fehlerrate



Generell sollte man meinen, dass Software einen einmal erreichten Qualitätsstand bis zum „End-of-Life“ beibehält.

Diese Aussage ist an sich richtig, wenn man davon ausgehen kann, dass dies Software an sich nie mehr verändert wird und so, wie sie abgenommen wurde auch bis zum Ende des Lebenszyklus bestehen bleibt.

Entgegen diesem idealisierten Fall ist Software aber aufgrund der sich permanent ändernden Rahmenbedingungen (Hardware, Betriebssystem, Datenbanksystem, grafische Oberfläche, etc.) einer Software-Evolution unterworfen, die dafür sorgt, dass die Software auch bei veränderten Rahmenbedingungen lauffähig bleibt und eventuell sogar zusätzliche Leistungsmerkmale erhält.

Aber gerade durch diesen Evolutionssprozess erfährt Software immer wieder Änderungen, die zusätzliche Fehler und Fehlverhalten einbringen und damit weitere Nachbesserungen nachziehen.

Dies wiederum sorgt dafür, dass der erreichte Qualitätsstand sich über die Zeit verschlechtert und Software somit auch einem Verschleiß-Prozess unterworfen ist, der nach einer längeren zeit es ebenso erforderlich macht das System einem „Redesign“ und einer Neuimplementierung zu unterwerfen.

Was beeinflusst Software Engineering?

- Komplexität
- Anzahl Nutzer
- Art der Nutzer
- Art der Software / der Anwendung
- Integrationsfähigkeit
- Wiederverwendbarkeit
- Systemumgebung (SW + HW)
- Programmiersprache
- Datenvolumen / Datenaufkommen
- Datenspeicherung (Dateien / Datenbanken)
- Fremddaten-Integration (CAD, BDE, ...)
- Anwendungsumgebung (Mission Critical)
- ...

Generellen Einfluß auf das Software-Engineering haben Faktoren wie:

- Komplexität des Systems, da mit steigenden Anforderungen auch die Software-technischen Anforderungen überproportional steigen
- Anzahl + Art der Benutzer stellt ebenso einen wesentlichen Einflußfaktor dar.
- Art der Software, Integrationsfähigkeit sowie die Forderung nach Wiederverwendbarkeit erhöhen ebenso die Komplexität beim Software-Engineering
- Wesentliche Einflüsse gehen auch von der Art der Entwicklung, der Entwicklungsumgebung sowie der Systemumgebung (Software wie Hardware) aus.
- Konkret verändert sich auch die Komplexität aufgrund des Datenaufkommens sowie der Art der Speicherung der Daten
- Insbesondere die Integrationsanforderungen und der automatischen Datenübernahme aus Fremdanwendungen kann den Komplexitätsgrad erheblich beeinflussen.
- Kritische Einsatzgebiete steigern die Komplexität ebenso

Der Begriff – Software Engineering

Software Engineering ist eine „Ingenieur-Wissenschaft“, die sich mit den Prinzipien, Methoden, Werkzeugen und Menschen befasst, die Software unter industriellen Bedingungen planen, entwickeln, anwenden und warten.

A. Schulz

In allen Definitionen wird der Begriff „Software-Engineering“ als Ingenieurs-Wissenschaft dargestellt und durch die Begriffe Prinzipien, Methoden, Werkzeuge charakterisiert, die dazu dienen Software-Erstellung nach industriellen Vorgaben systematisch zu planen, entwickeln, anwenden und pflegen bzw. weiterentwickeln.

Projekt-Charakteristika

- Einmaliges Ereignis (nicht unbedingt erstmalig)
- Hohe Komplexität
- Viele Beteiligte
- Unterschiedliche Disziplinen
- Teamarbeit der Beteiligten
- Zeitlich begrenzt
- Klar definierte Anforderungen
- Festgelegtes Ergebnis / Funktionalität
- Vorgegebenes Budget
- Planbare Abwicklung
- Integration in bestehende Umgebung
- Klare Rahmenbedingungen und Durchführung nach Vorgehensmodell

Im Bereich der professionellen Software-Entwicklung ist davon auszugehen, dass die damit verbundenen Software-Projekte über einige oder alle der folgenden charakteristischen Merkmale verfügt, die die Komplexität der Planung und Umsetzung wesentlich beeinflussen:

- Die projektmäßige Umsetzung ist als einmaliges nicht in gleicher Weise wiederkehrendes Ereignis anzusehen
- Die Komplexität des Projekts ist sehr hoch
- Viele Beteiligte (Entwickler, Kunden, Auftraggeber, Lieferanten) in unterschiedlichen Disziplinen sind involviert
- Zeit und Kosten sind streng limitiert, die Anforderungen fest vorgegeben
- Weitere, nicht systemspezifische Anforderungen wie vorgegebene Entwicklungsumgebung u.a. kann die Komplexität zusätzlich beeinflussen
- Meist ist eine Integration in bestehende Systeme unerlässlich
- Vorgehensmodell und Umsetzung sind ebenso oft Bestandteil der Vorgaben.

Vom Programm zum Produkt

- Einfache Programmentwicklung
  - *Faktor 1*
- Wiederverwendbare Programme/Funktionen
  - *Faktor 10*
- Software-Produkte
  - *Faktor 100*

Der Weg:

vom einfachen Programm zum fertigen Produkt

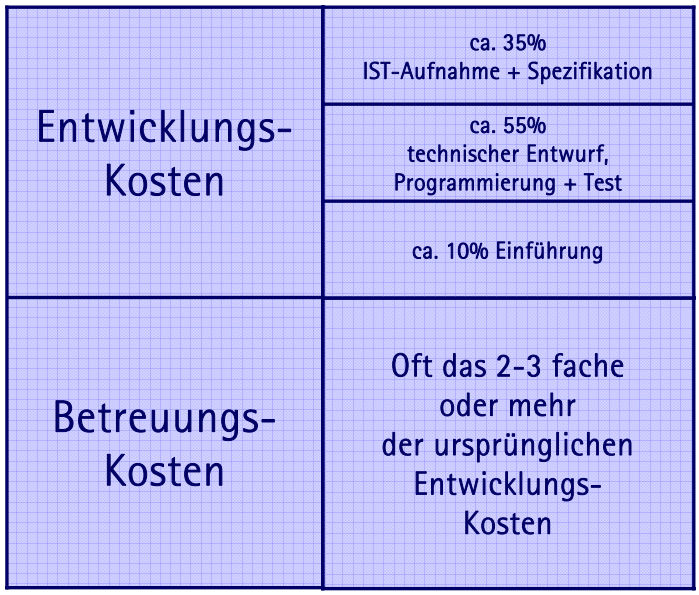
ist mit erheblichem Aufwand und damit auch mit entsprechenden Kosten beaufschlagt.

Diese Kostenfaktoren kommen dadurch zustande, dass das einfache Programm durch das in der Spezifikation festgelegte Verhalten 100%ig festgelegt ist und nur diese Rahmenvorgaben zu erfüllen sind.

Soll dieses Programm aber in der Gesamtheit und/oder in Teilen bereits anderen zur Wiederverwendung zur Verfügung gestellt werden, ist ein Faktor 10 des Aufwands zu treiben, um die entsprechende Funktionsneutralität und Dokumentation sicherzustellen.

Dieser Aufwand erhöht sich nochmals um den Faktor 10, sollte aus dieser wieder verwendbaren Software ein „Produkt“ generiert werden, da dadurch sowohl der Test wie auch der Dokumentationsaufwand weiter steigt und möglichst alle Eventualitäten einer Installation durch Fremdpersonal berücksichtigt und getestet sein müssen.

Kostenstruktur – Software-Produkt



Die Kostenstruktur der Entwicklung von Software ist nicht nur durch die eigentliche Entwicklung geprägt, da dies von den Entwicklungskosten nur ca. 50-55% ausmacht.

Ein wesentlicher Teil der Kosten mit ca. 40% fließt in die Aufnahme der Anforderungen und die Spezifikation dieser im Detail.

Die eigentliche Einführung und Auslieferung generiert dagegen nur Kosten von ca. 10%

Sollte die Software als System über die Zeit gepflegt und angepasst werden, so werden i.d.R. nochmals ca. das 2-3 fache der ursprünglichen Kosten für diese Aktivitäten erforderlich, um das System am Leben zu erhalten und an die sich schnell verändernden Randbedingungen anzupassen.

Ziele des Software Engineering

- Steigerung der *Produktivität*
- Verbesserung der *Qualität*
- Verbesserte *Führbarkeit* der Projekte

Generell können die Ziele des Software-Engineering mit den drei Begriffen:

1. Produktivität
2. Qualität
3. Führbarkeit

charakterisiert werden.

Software-Maße

- Software-Maße werden „Metriken“ genannt
- Software-Maße werden per Definition vorgegeben
- Beispiele für Software-Metriken
  - Anzahl von Programmzeilen je Routine
  - Anzahl von Schleifen in einer Routine
- Beispiel für eine Software-Metrik
  - Programm-Größe
  - Messkriterium: „Lines of Code“
  - Skala: Nicht negative ganze Zahlen

Kriterien zur Messung der Qualität von Software – so genannte Metriken – sind nur sehr schwer festzulegen.

Oft ist dies nur durch die Einbeziehung indirekter Indikatoren möglich, die Aufschluss über das System geben.

Direkte und Indirekte Maße

- Direkte Maße
  - Direkt messbare Merkmale
  - Beispiele: Kosten, Durchlaufzeit
- Indirekte Maße
  - Kein direktes Maß vorhanden oder Messung zu aufwendig / zu teuer
  - Messbare Indikatoren bestimmen
  - Indikatoren müssen in dem zu messenden Merkmal korreliert sein
  - Indikatormaße bilden zusammen ein „indirektes Maß" für zu messende Merkmale
  - Beispiele: Portabilität, Benutzerfreundlichkeit

Direkte Messgrößen wie Kosten, Durchlaufzeit sind kein Maß für die Qualität und Erfüllung der Anforderungen. Hierzu bedient man sich oft einiger Indikatoren, die über diese Größen indirekt Aufschluss geben.

Maße für Software Engineering Ziele

- Produktziele
  - Funktionserfüllung, Größe, Zuverlässigkeit, Benutzerfreundlichkeit, Wartbarkeit, ...
  - Meist nur indirekte Maße
  - Messung oft schwierig
- Projektziele
  - Aufwand, Durchlaufzeit (Dauer), Arbeitsfortschritt, Entwicklungskosten, Fehlerkosten, ...
  - Meist direkte Maße
  - Messung nicht schwierig, wenn Ziele richtig definiert sind und die richtigen Basisdaten erhoben werden

Generell ist zwischen den Produktzielen und den Projektzielen zu unterscheiden.

Produktziele sind anhand der Anforderungsspezifikation sowie der Systemabnahme oft nur indirekt zu überprüfen, indem die Vorgaben mit dem Systemverhalten verglichen werden. Eine 100%ige Sicherheit der absolut richtigen Umsetzung kann dabei nicht erreicht werden, da eine absolute Überprüfung aller möglichen Systemreaktionen aufgrund der Komplexität nicht sinnvoll umsetzbar ist.

Projektziele sind direkt mit der Umsetzung der Anforderungen gegeben und können z.B. mittels Größen wie Laufzeit, Kosten, Termine direkt überprüft werden.

Zusammenfassung – Software

- **Software / Software-Entwicklung**
  - Software besteht aus Instruktionen, Datenstrukturen und Dokumenten
  - Software kommt heute in fast allen komplexeren Produkten vor
  - Software ist immaterielles, unstetig, leicht zu ändern, schwer abschätzbar
  - Probleme bei der Software-Entwicklung sind
    - Kreativität der Entwickler
    - Keine direkte „Sichtkontrolle“ ob korrekt
    - Fehlerfreiheit ist nicht nachweisbar
  - Software-Entwicklung ist der Prozess, durch den Software entsteht und weiterentwickelt wird
  - Die Komplexität der Software + deren Entwicklung steigt nicht linear sondern exponentiell mit der Zunahme der Anforderungen
  - Software unterliegt ebenso wie andere Produkte einem „Verschleiß“, da durch Weiterentwicklung und Anpassungen weitere Fehler und zu integrierende Module hinzukommen.
  - Konkrete Masse für Software sind nur schwer definierbar und werden oft über Indizien und indirekte Faktoren sichergestellt

Software Engineering

Informatik II.

2. Software-Entwicklung

– Der Software-Prozess –

Dipl.-Inform. Hartmut Petters

Was macht „gute" Software aus?

- Wartbarkeit
  - Weiterentwicklung
  - Portierung
- Zuverlässigkeit
  - Betriebssicherheit
  - Abgesicherte Funktionalität
- Effizienz
  - Nutzung der Systemressourcen
  - Reaktions- und Verarbeitungszeit
- Benutzerfreundlichkeit
  - Selbsterklärend, intuitiv bedienbar
  - Dokumentation + Hilfe-Funktion

Wesentliche Merkmale einer „guten Software" sind

- Wartbarkeit
  im Rahmen der Weiterentwicklung des Systems und der Anpassung des Systems an sich verändernde Rahmenbedingungen sowie die Portierbarkeit auf unterschiedliche Systemplattformen
- Zuverlässigkeit
  im Sinne von einem störungsfreien Betrieb der Software 24x7 (wenig Abstürze mit Neustart), keine Datenverlust sowie einer abgesicherten Funktionalität, die sich in einer umfassenden Dokumentation niederschlägt
- Effizienz
  in dem die optimale Nutzung der verfügbaren Ressourcen (Hardware, System-Software) so sichergestellt wird, dass die System-, Reaktions- und Antwortzeiten in einem akzeptablen Rahmen und zeitgemäß sind .
- Benutzerfreundlich
  im Sinne einer weitestgehend selbsterklärenden Software, die intuitiv bedienbar und durch entsprechende Dokumentationen und integrierten Hilfe-Funktionen gute Unterstützung bietet.

Software-Prozess / Prozess-Arten

- Software-Prozesse beschreiben den Ablauf der Entwicklung + Pflege von Software
- Prozess
  - Folge von Schritten, die zur Erreichung eines gegebenen Zwecks ausgeführt wird
  - Unterscheidung
    - Ad-Hoc-Prozess
      - Spontan
      - Individuell
      - Ungeregelt
    - Systematischer Prozess
      - Geplant
      - gelenkt

Software-Entwicklungs-Prozesse beschreiben den Ablauf und die Umsetzung der Entwicklung und Pflege von Software-Produkten und stellen eine Abfolge von einzelnen Schritten dar, die zweckgerichtet zur Systemerstellung festgelegt werden.

Dabei werden zwei Arten von Prozesstypen unterschieden:

1. Ad-Hoc-Prozess
   als willkürliche und spontane Umsetzung von individuellen Anforderungen in einem ungeregelten Vorgehen, ohne Planung und Kontrolle.
2. Systematischer Prozess
   Als geplantes Instrument zur systematischen Umsetzung der Anforderungen in einem Prozess, der aus den Basis-Schritten: Planung- Überprüfung – Verifikation und Anpassung besteht.

   Komplexe Software-Systeme können nur in einem systematischen Prozess sinnvoll und kalkulierbar umgesetzt werden.

Software-Entwicklung im Ad-Hoc-Prozess

- Entwicklungsprozess im „unprofessionellen Bereich"
- Vage Sachziele +keine Langfristplanung
- Keine Termin- und Kostenvorgaben
- Keine Spezifikationen, bruchstückhafte Entwürfe
- Test nur ansatzweise, Qualität kein Thema
- Entwicklung : „Trial-and-Error"
- Produkte eher klein, kurzlebig + undokumentiert
- I.d.R. nur eine Person involviert
- Pflegemaßnahmen + Entwicklung spontan
- Nur für kleine Anwendergruppe gedacht

Software-Entwicklung im „Ad-Hoc-Prozess" die auch unter der Bezeichnung „Midnight-Hack" zusammengefasst werden kann, wird nur in „unprofessionellen" Bereichen eingesetzt. Einsatz dieser Methoden zur Erzeugung kommerziell vermarkteter Software führt nur zu unüberschaubaren Problemen und Kundenunzufriedenheit, da weder die Funktionalität noch die Qualität gewährleistet werden kann.

Die Charakteristika dieser Methode sind im Wesentlichen

- ungeplantes Vorgehen
- Keine oder nur sehr vage schriftliche Spezifikation
- Keinerlei Vorgaben bzgl. Kosten + Termine
- Keine systematischen Tests zur Qualitätssicherung
- Entwicklung im „Trial-and-Error"-Verfahren
- Nur für kleine Insider-Anwendergruppen gedacht

Professionelle Software-Entwicklung

- „Ad-Hoc-Prozess" wird nur für experimentelle Entwicklungen + Prototypen eingesetzt
- Problem wenn Produkte aus einer „Ad-Hoc-Entwicklung" an Benutzer ausgeliefert wird
- I.d.R. geplante + gelenkte Prozesse
  - Abwicklung wird geplant
  - Klare Termin-, Kosten- und Sachziele
  - Ablauf des Prozesses wird überprüft
  - Lenkungsmaßnahmen bei Störungen im Ablauf
  - Qualität ist sehr wichtig
  - Produkt wird gepflegt und i.d.R. weiterentwickelt

Im Gegensatz zum „Ad-Hoc-Prozess" unterscheidet sich „professionelle Software-Entwicklung" durch eine systematische Planung und einem nicht unerheblichen Aufwand der Spezifikation, Festlegung der Abnahmekriterien sowie der systematischen und nachvollziehbaren Durchführung aufwendiger Tests der Komponenten und des Gesamtsystems.

Nur durch diesen Rahmen ist eine qualitativ hochwertige Software-Entwicklung möglich.

Kontrollmechanismen sind dabei

- Termine, Kosten sowie Funktionalitäten
- Vordefinierte Abläufe und Meilensteine
- Steuerungsmechanismen zur Sicherstellung der Zielerreichung
- Einhaltung vorgegebener Qualitätskriterien
- Langfristig angelegte Planung bzgl. Weiterentwicklung + Anpassung

Systematische Software-Entwicklung

- Entstehungsprozess mit klar definiertem Anfang und Ende
- Ziel ist es, ein Software-Produkt zu entwickeln, das eine vorgegebenen Aufgabe erfüllt
- Mögliche Optionen
  - Entwicklung neuer Software
  - Beschaffung, Konfiguration und/oder Anpassung

Systematische Software-Entwicklung ist durch eine klare Festlegung des Anfangs und des Endes der Entwicklung gekennzeichnet.

Ziel ist es ein Produkt zu erstellen, das einer vorgegebenen Anforderungsspezifikation entspricht und die darin festgelegte Leistung vollständig erbringt.

Optionen dabei sind:

- Entwicklung ganz neuer Software (ohne bestehende Referenz)
- Weiterentwicklung eines bestehenden Systems
- Erstellung eines Systems aufgrund bestehender, vorgegebener Funktionalitäten (Referenzsystem) als Ablösung
- Beschaffung, Konfiguration und/oder Anpassung im Rahmen der Wartung (Einführung komplexer Anwendungssysteme wie z.B. SAP R/3)

Systematische Software-Pflege (Wartung)

- Kontinuierlicher Prozess
- Ziel: Erhaltung der Gebrauchstauglichkeit
- Regelt Reaktionen auf auftretende Probleme
- Plant + lenkt
  - Weiterentwicklung der Software
    - Zusätzliche Funktionalitäten
    - Änderung bestehender Funktionalitäten (Ablauf)
  - Laufende Anpassung an sich verändernde Umwelt
    - Technische Umgebung (O/S, DB, Netzwerk, ...)
    - Organisatorische Umgebung (funktionale Ergebnisse)

Systematische Software-Pflege ist ein kontinuierlicher Prozess aufgrund der sich ändernden Rahmenbedingungen in den Bereichen

- Fachlich (Inhalte haben sich verändert / z.B. geänderte Lohn-/Gehaltsabrechnung aufgrund veränderter Gesetze)
- Systemtechnisch (neues Betriebssystem / Datenbanksystem / Netzwerkschnittstellen oder Schnittstellen bestehender Fremdsysteme, in die sich die Anwendung zu integrieren hat)

Generelles Ziel ist es die Gebrauchstauglichkeit über einen längeren Zeitraum hinweg sicherzustellen.

Drei Software-Klassen nach Lehmann

- Lehmann teilt Software in 3 Klassen ein
  - *S-Typ*
    vollständig durch formale Spezifikation beschreibbar
    Erfolgskriterium: Spezifikation nachweisbar erfüllt
  - *P-Typ*
    Löst ein abgegrenztes Problem
    Erfolgskriterium: Problem zufriedenstellend gelöst
  - *E-Typ*
    Realisiert eine eingebettete Anwendung
    Erfolgskriterium: Anwender zufrieden
  - Software vom *S-Typ* ist stabil
  - Software vom *P-Typ* und *E-Typ* ist einer Evolution unterworfen

Nach Lehmann kann Software in drei Klassen eingeteilt werden, wobei davon nur eine einzige – Software vom S-Typ – vollständig beschreibbar und damit durch eine formale Spezifikation und überprüfbar bzgl. der Funktionalität ist.

Die beiden Typen „P“ und „E“ sind der üblichen Software-Evolution unterworfen und nicht formal beschreibbar, so dass dies erfolgreiche Umsetzung nur aufgrund indirekter Messkriterien überprüfbar ist.

Software-Evolution

„Wandel der Software durch Veränderung des Umfeldes und der Randbedingungen"

- Jedes technische Produkt unterliegt einer Evolution
- Besonderheiten der Informatik
  - Evolution läuft hier besonders schnell ab
  - Teilweise ist die Evolution selbststeuernd

Software-Evolution umfasst alle Veränderungen, die eine nachträgliche und kontinuierliche Anpassung der Software erforderlich machen, damit diese weiterhin gebrauchsfähig ist.

Dies sind insbesondere Anpassungen aufgrund veränderter Rahmenbedingungen bzgl. der Funktionalität wie z.B. geänderte Gesetze, Abrechnungsverfahren und ähnliches sowie alles Weiterentwicklungen und Veränderungen der Systemumgebung im Bereich der System-Hardware und System-Software durch Innovation und Pflege.

Konsequenzen für Software vom P- / + E-Typ

- *P-Typ* + *E-Typ*-Software ist nie über mehrere Jahre hinweg gebrauchstauglich ohne Veränderung
  - Pflege ist kein „Unfall" sondern unvermeidlich
  - Überlegungen zur Software sollten sich immer auf die gesamte Lebensdauer der Software beziehen
- Längere Software-Projekte:
  - Änderungen der Anforderungen während der Entwicklung sind wahrscheinlich
  - Sich verändernde Ziele sind kein „Unfall" sondern liegen in der Natur der Software-Entwicklung

Software vom Lehmann´schen Typ „P" und „E" unterliegt generell der fortschreitenden Veränderung der Systemumgebung und bedarf einer kontinuierlichen Anpassung an diese um weiterhin die daran gestellte Aufgabe zu erfüllen.

Meilensteine sind Kontrollpunkte

- Meilenstein
  - eine Stelle in einem Prozess, an dem ein geplantes Ergebnis vorliegt und überprüft werden kann.
  - Das Mittel zur
    - Prozess- und Projekt-Strukturierung
    - Kontrolle des Projektfortschritts
    - Planung durch Festlegung von Ergebnissen
  - Wurde erreicht, wenn das geplante Ergebnis vorliegt
  - Kontrollpunkt zum SOLL-IST-Vergleich
    - Funktionalität
    - Kosten
    - Zeit
  - Nur sinnvoll, wenn messbar / überprüfbar

In der Software-Entwicklung wie auch bei anderen Projekten spielen „Meilensteine“ eine wesentliche Rolle, da sie Fixpunkte im Projekt sind, an denen fest vorgegebene Entwicklungsergebnisse überprüfbar sind.

Deswegen sind Meilensteine das adäquate Mittel für die Prozess- und Projekt-Strukturierung und dienen der Kontrolle des Projektfortschritts.

Meilensteine sind bereits am Anfang der Projektplanung klar festzulegen und anhand konkreter und überprüfbarer Messkriterien zur Ergebniskontrolle einzuplanen.

Insbesondere sind Meilensteine einerseits zur Ergebnis-Sicherung unerlässlich, aber haben zudem den Nebeneffekt, dass in Abhängigkeit der vertraglichen Regelungen mit Erreichen eines Meilensteins gegebenenfalls Teilzahlungen möglich sind, die bei einer längeren Projektlaufzeit eine Finanzierung der Entwicklungstätigkeiten sicherstellen.

Was ist ein Software-Prozess?

- Tätigkeiten, durch die Software entsteht
- Vier grundlegende Prozess-Aktivitäten
  - Software-Spezifikation
  - Software-Entwicklung
  - Software-Validierung
  - Software-Evolution
- Eigenschaften des Software-Prozesses
  - systematisch
  - geplant
  - gesteuert

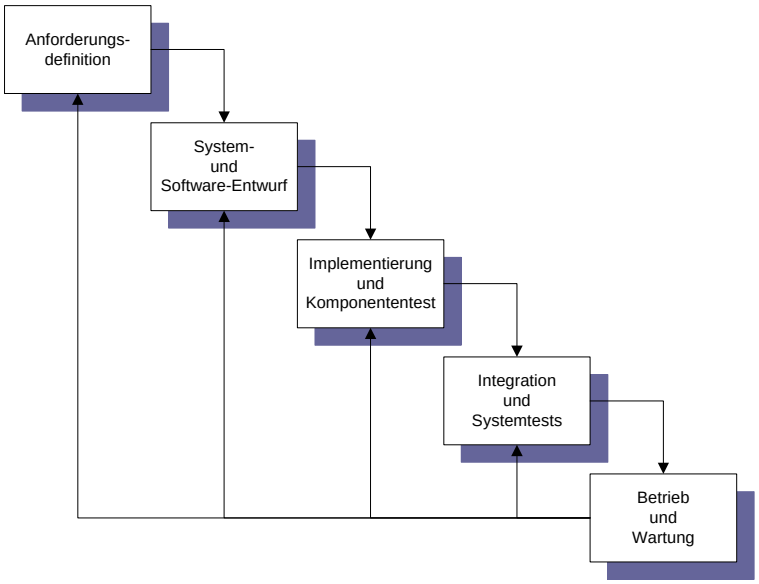
Der Software-Prozess umfasst alle Tätigkeiten, die zur Entsehung eines Software-Produkts erforderlich sind. Insbesondere sind dies die Aktivitäten der Phasen:

- Anforderungsspezifikation der Software
- System-Entwicklung / Software-Erstellung
- Software-Validierung – Überprüfung der Zielerreichung
- Weiterentwicklung aufgrund der Evolution

Grundlegende Eigenschaften dieser (Teil-)Prozesse sind

- Systematische Umsetzung
- Planung der erforderlichen Aktivitäten, Termine, Kosten, Ressourcen, Meilensteine
- Steuerung dieser Aktivitäten und der Folgeplanung durch Anpassung an veränderte Rahmenbedingungen

Der Software-Lebenszyklus



Der Lebenszyklus von Software wird durch die obige Darstellung charakterisiert.

Ausgewählte Prozess-Modelle

- **Wasserfallmodell**
  - Gliederung des Projekts in eine Folge von Aktivitäten
- **Ergebnisorientiertes Phasenmodell**
  - Gliederung in eine Folge von Zeitabschnitten
    Phasenabschluss durch überprüfbare Meilensteine
- **Wachstumsmodell**
  - Gliederung in eine Folge von Lieferschritten
- **Spiral-Modell**
  - Gliederung in eine zyklische, am Risiko orientierte Folge von Entwicklungsschritten
- **Prototypen**
  - Einsatz in verschiedenen Modellen möglich
    hauptsächlich zur Risikoabschätzung und –Steuerung
- **Agile Software-Entwicklung**
  - Schlanke Software-Entwicklungs-Prozesse

Ausgewählte Prozessmodelle sind:

- Wasserfallmodell
  bei dem das Projekt in einer Folge von Aktivitäten geplant wird
- Ergebnisorientiertes Phasenmodell
  als eine Variante des Wasserfallmodells, das sich aber nicht an den Aktivitäten, sondern an den Ergebnissen konkreter Meilensteine und der Planung der zugrunde liegenden Zeitabschnitte orientiert.
- Wachstums- oder Evolutionsmodell
  bei dem die Planung an konkreten funktionsfähigen Lieferungen fest gemacht wird und somit eine Folge von Lieferschritten entsteht
- Spiral-Modell
  als Weiterentwicklung des Phasenmodells in Form zyklischer, am Risiko orientierter Entwicklungsschritte
- Prototypen
  die in verschiedenen Entwicklungsmodellen Einsatz finden zur Erprobung und Risikoeingrenzung
- Agile Software-Entwicklung
  als modernes Modell zur Umsetzung schlanker + effizienter Entwicklungs-Prozesse

Wasserfallmodell - Eigenschaften

- Ältestes systematisches Prozess-Modell
- Entwicklung als Sequenz aufeinanderfolgender Entwicklungs- und Prüfschritte
- Jede (Haupt-)Tätigkeit bildet eine Phase
- Phasen-Übergang erfolgt, wenn geprüftes und akzeptiertes Ergebnis vorliegt
- Lokale Iterationen sind möglich
- Es gibt eine Vielzahl ähnlich konstruierter Modelle

Hauptproblem:

- Nicht-Lokale Iterationen (Management erschwert)
- In Ursprungsform besser nicht einsetzen!

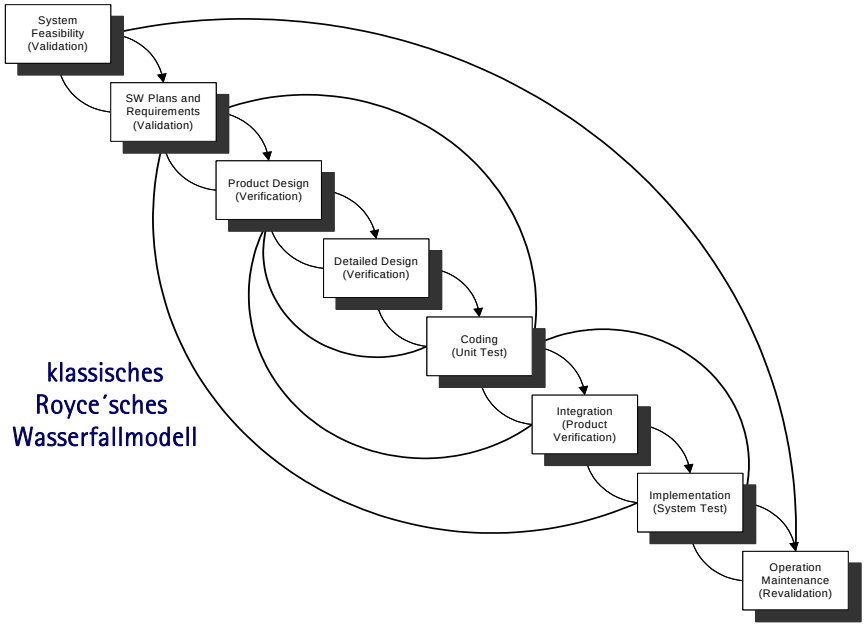
Das Wasserfallmodell ist das älteste systematische Modell zur Software-Entwicklung, in dem jede Haupttätigkeit als eigene Phase geplant wird. Phasenübergänge erfolgen nach erfolgreicher Prüfung und Abschluss der aktuellen Phase.

Spätere Veränderungen führen dabei zu einer nachträglichen Überprüfung und Iteration zwischen den einzelnen Phasen, bis das endgültige Ergebnis vorliegt.

Viele Modelle sind als Varianten vom Wasserfallmodell abgeleitet worden.

Für komplexe Projekte ist das Wasserfall in der Urform wegen der nur lokal möglichen Iterationen weniger geeignet und hat für die aktuellen Entwicklungen kaum Bedeutung.

Wasserfallmodell – „Nicht-lokale Iterationen“



Das um nichtlokale Iterationen erweiterte Wasserfallmodell ist geeignet auch weitergehende Projekte zu begleiten.

Ergebnisorientiertes Phasenmodell –Prinzip

- Orientiert sich am Software-Lebenslauf
- Grundidee:
  Entwicklung wird in eine Sequenz aufeinanderfolgender Zeitabschnitte unterteilt
- Je Phase wird ein Teil des Ergebnisses erarbeitet
- Phase wird nach Ergebnis / Haupttätigkeit benannt
- Keine Iterationen
- Jeder Phasenabschluss ist ein Meilenstein
- Es gibt eine Vielzahl von Varianten und Namen

Das ergebnisorientierte Phasenmodell orientiert sich am Software-Lebenslauf.

Ergebnisorientiertes Phasenmodell + / -

- Vorteile
  - \+ Leicht verständlich
  - \+ Folgt dem „natürlichen" Lebenslauf
  - \+ Geeignet für Projekt-Management
  - \+ Fördert planvolles Vorgehen
  - \+ Trägt zur frühzeitigen Fehlererkennung bei
- Nachteile
  - – Ignoriert die Software-Evolution
  - – Lauffähige Systeme entstehen erst sehr spät
    - – Lange „Papier-Durststrecke"
    - – Technische Risiken
    - – Adäquatheitsrisiken
  - – System wird auf einen Schlag in Betrieb genommen (Big Bang)

Als geeignetes Modell für das Projektmanagment wird das ergebnisorientierte Phasenmodell auch heute noch eingestzt.

Ergebnisorientiertes Phasenmodell - Eignung

- Ergebnisorientierte Phasenmodelle sind geeignet
  - Management nicht zu großer Projekte
  - Projekte mit genau definierten Anforderungen
  - Wenn die Entwickler über das erforderliche „Know-How" verfügen
  - Wenn das Entwicklungsrisiko eher gering ist

Evolutionäre Entwicklung - Wachstumsmodell

- Leitbild: Software-Evolution
- System wird nicht konstruiert, sondern wächst
- Verschiedene Bezeichnungen
  - Versionen-Modell
  - Evolutionäres Modell
  - Inkrementelles Modell
- Grundidee:
  Gliederung der zu liefernden Software in aufeinander aufbauende, betriebsfähige Lieferungen
- Unterteilt Entwicklung in Folgen von Iterationen
- Pro Iteration: Vollständiges Teilergebnis mit betriebsfähiger Software
- Lieferung sind als autonome Teilprojekte organisiert

Das Wachstumsmodell orientiert sich an den Anforderungen, die an die einzelnen Lieferungen gekoppelt sind, wobei das System nicht in seiner Gesamtheit auf einmal konstruiert wird, sondern in einzelne lauffähige Lieferungen unterteilt wird, sodass der Anwender zu jedem Zeitpunkt eine lauffähige Variante des Systems testen und validieren kann.

Die Festlegung der einzelnen Lieferungen kann dabei unter verschiedenen Gesichtspunkten erfolgen.

Evolutionäre Entwicklung - Planung

- Umfang und Reihenfolge der Lieferungen
- Planung bis zum gewünschten Endzustand
- Schrittweiser Ausbau nach verschiedenen Merkmalen ist möglich
  - Nach Funktionalität
  - Nach Bedienungskomfort
  - Nach Leistung + Performance
  - Nach Verwendbarkeit

Die Planung der Entwicklung nach dem Wachstumsmodel erfolgt dabei in einzelnen Versionen, wobei bereits von Beginn an der Umfang und die Reihenfolge der Lieferungen bis hin zum endgültigen Endzustand festgelegt wird.

Der schrittweise Ausbau des Systems kann nach unterschiedlichen Gesichtspunkten erfolgen wie

- Nach Funktionalität
- Nach Bedienkomfort
- Nach Leistung + Performance
- Nach Verwendbarkeit

Dabei können bei den einzelnen Lieferungen die Ziele der Lieferung variieren, so dass die vier genannten Merkmale wechselweise befriedigt und als Schwerpunkt für eine Lieferung herangezogen werden.

Evolutionäre Entwicklung – Integration

- Normales Wachstumsmodell
  - Betriebsfähige Version des Systems nach jeder Teillieferung
- Kontinuierliche Integration
  - Ständig eine betriebsfähige Referenz-Version mit dem aktuellen Entwicklungsstand verfügbar
- Referenzversion wird in kurzen Abständen (täglich, wöchentlich, ...) neu erzeugt („daily build")
- Verkürzt die Rückkopplungszyklen und senkt damit die Fehlerkosten
- Funktioniert gut, wenn
  - Aufgabe in viele kleine, autonome Teilaufgaben teilbar
  - Aufwand für Neuerzeugung der Referenzversion gering ist
- Gefahr, dass notwendige Restrukturierungen unterbleiben

Beim normalen Wachstumsmodell steht jederzeit ein betriebsfähiges System zur Verfügung, das zu Systemtests und zum Test der Gebrauchstauglichkeit herangezogen werden kann.

Durch diese Vorgehensweise sind kurze Rückkopplungsphasen sichergestellt, so dass Fehlentwicklungen kurzfristig korrigierbar sind und Anwenderbedürfnisse direkt einbezogen werden können.

Eine Anwendung dieses Modells ist möglich, wenn die Anforderungen in kleine, unabhängige Aufgaben untergliedert werden können, die getrennt realisierbar sind und jeweils als Teillieferung installiert werden können.

Voraussetzung ist ebenfalls, dass sich der zusätzliche Aufwand für die Erstellung der einzelnen Lieferungen in akzeptablen Grenzen hält.

Aufgrund der evolutionär wachsenden Software besteht dabei die Gefahr, dass die Struktur der Software, die zu Anfang festgelegt werden muss, den letztendlichen Anforderungen nicht genügt und eine Restrukturierung erforderlich wäre, um eine langfristige Pflege der Software sicherzustellen. Oft werden diese notwendigen Restrukturierungen der Software aus Kosten- und Termingründen unterlassen

Evolutionäre Entwicklung + / -

- Vorteile
  - + Modellieren das natürliche Verhalten von Systemen
  - + Sehr gut geeignet für Projekt-Management
  - + Frühe Verfügbarkeit lauffähiger Teilsysteme
    - + Fördert die Motivation der Beteiligten
    - + Lindert größte Nöte durch System-Verfügbarkeit
  - + Verkürzte Rückkopplungszyklen
  - + Schrittweise System-Einführung
- Nachteile
  - – Gefahr der Insel-Bildung
  - – Gefahr der Zerstörung von Strukturen und Konzepten durch den schrittweisen Ausbau des Systems

Bei fähigen Entwicklern und Systemarchitekten durchaus für größere Vorhaben geeignet, wenn die Basiskriterien erfüllt sind. Allerdings erfordert diese Art der Entwicklung sowohl Disziplin wie auch sauber Planung und Dokumentation.

Evolutionäre Entwicklung - Eignung

- Evolutionäre Entwicklung ist gut geeignet bei
  - großen Systemen mit langen Entwicklungszeiten
  - Systemen, bei denen die volle Funktionalität nicht sofort verfügbar sein muß
  - Vorhaben, die vorab nicht vollständig spezifiziert sein müssen
  - Ein erstes Ergebnis und Basis-System schnell verfügbar sein soll

Die iterative Vorgehensweise bildet die Basis für den Einsatz bei Systemen, die nur schwer spezifizierbar sind. Wenn die Entwicklung auf der Basis von „Time-and-Material“, d.h. als Abrechnungsbasis auf Stundensatz-Abrechnung erfolgt, ist dabei keinerlei Risiko im Kostenbereich. Ansonsten ist durch die ungenügende Spezifikation das Risiko der nicht kalkulierbaren Kosten damit verbunden.

Spiralmodell von Boehm (1988)

- Weiterentwicklung des Wasserfallmodells
- Vor allem für die Entwicklung großer und risikoreicher Projekte / Systeme gedacht
- Zweigeteilter Zyklus (Entwickeln / Prüfen) des Wasserfallmodells wird durch vierteiligen Zyklus ersetzt
  - Planung
  - Zielsetzung / Steuerung / Zielkorrektur
  - Risikountersuchung / Risikobewertung
  - Entwicklung + Prüfung
- Manchmal wird das Spiralmodell fälschlicherweise als Wachstumsmodell bezeichnet

Da Spiral-Modell wurde von Boehm vor allem für die Umsetzung großer und risikoreicher Projekte entwickelt und erfolgreich eingesetzt.

Der bisherige Zyklus „Entwickeln – Prüfen“ des Wasserfallmodells wird durch die vier Stufen

1. Planung der Phase
2. Zielsetzung / Steuerung / Korrektur
3. Risikoanalyse / Risiko-Bewertung
4. Entwicklung + Prüfung

ersetzt.

Agile Software-Entwicklung - Idee

- Idee
  Die Verfahren für die Entwicklung von Klein-Software so auf große Software-Projekte übertragen, dass
  - Es erfolgreich funktioniert
  - Die Prozesse möglichst einfach und wenig reglementiert sind (Zeitersparnis, weniger Prozess-Bürokratie)
  - Entwickelt in den späten „90er" Jahren, ab 2000 zunehmend populär

Die Agile Software-Entwicklung setzt die Verfahren, die für die Entwicklung kleinerer Projekte eingesetzt werden für die Realisierung großer Projekte ein, indem das Großprojekt als der Zusammenspiel vieler Kleinprojekte betrachtet wird und in der Umsetzung diese vielen Kleinprojekte entsprechend realisiert werden.

Agile Software-Entwicklung - Prinzipien

- Grundlegende Prinzipien
  - Je kleiner die Teilaufgabe und je kürzer die Rückkopplungszyklen, desto besser
  - Der Kunde gestaltet das Projekt während seiner Entstehung
    - Kundenvertreter ist/sind aktiv am Projekt beteiligt
  - Je freier und konzentrierter die Entwickler arbeiten können, desto besser
  - Die Qualität wird an der Quelle sichergestellt
    - „pair programming"
    - Rigorose Komponententests
  - Keine Arbeit auf Vorrat

Die Agile Software-Entwicklung erfordert es, dass die Ergebnisse der Teilentwicklungen durch einen Kundenvertreter direkt kontrolliert werden, der einerseits über die entsprechenden Kenntnisse der Anforderungen verfügt und andererseits über die Entscheidungsbefugnisse, um kurzfristig Entscheidungen bzgl. der weiteren Entwicklung treffen kann.

Die Entwicklung selbst erfolgt im Team, in dem einer programmiert und der andere die Funktionalität und Fehlerfreiheit überprüft.

Aufgaben werden vorzu erledigt, so wie sie anstehen, um den Systemfortschritt zu gewährleisten. Eine Arbeit auf Vorrat von Programm-Modulen, die erst später benötigt werden erfolgt nicht, da nur die Ergebnisse sofort und Konsequent überprüft werden können, die sofort integrierbar sind.

Agile Software-Entwicklung - Erfahrungen

- **Erste Erfahrungen**
  Agile Software-Entwicklung funktioniert, wenn
  - Auftraggebervertreter verfügbar, kompetent und entscheidungsbefugt sind
  - Das Problem in hinreichend viele kleine Teilaufgaben unterteilbar ist
  - Ein kleines Entwicklungs-Team (oder mehrere kleine parallele Teams)
  - Ein kompetenter System-/Software-Architekt
  - Kontinuierliche Integration mit geringem Aufwand möglich ist
  - Konsequente Qualitätssicherung an der Quelle erfolgt
- **Sonst Absturz ins Chaos und in Ad-Hoc-Entwicklung**
- **Vor- und Nachteile wie bei Wachstumsmodell, aber verstärkt**

Erste Erfahrungen liegen vor und zeigen, dass bei einer ´konsequenten Umsetzung der zugrunde liegenden Prinzipien dies überaus ein sinnvolles Realisierungsmodell ist.

Allerdings erfordert dies sowohl hohes Expertentum wie auch absolute Disziplin.

Anwendung dieses Modells ohne entsprechende Erfahrung und Basis führt zu einem nicht kalkulierbaren Ergebnis.

Prototyp und Prototyping

- Zentrales Mittel zur frühzeitigen Erkennung und Lösung von Problemen
- Prototyp
  Lauffähiges Stück Software, welches kritische Teile eines zu entwickelnden Systems vorab realisiert
- Prototyping
  Prozess zur Erstellung und Nutzung von Prototypen
- Drei Arten von Prototyping
  - Explorativ
  - Experimentell
  - Evolutionär

Prototypen sind kein eigenes Modell, sondern werden im Rahmen der Umsetzung zur Erprobung und zur Machbarkeitsstudie herangezogen.

In den meisten Fällen handelt es sich bei den erstellten Prototypen um exemplarische Realisierungen, die nicht nach den Regeln der Kunst bzgl. Software-Engineering entwickelt wurden und nur die Machbarkeit zeigen sollte. Diese dadurch entstandenen Prototypen sind so genannte Wegwerf-Entwicklungen und sollten nie als Basis für die weitere Entwicklung herangezogen werden, da dies sonst zu einer nicht kalkulierbaren Basisentwicklung führt.

Evolutionäres Prototyping

- Prototyp ist Pilotsystem
- Bildet den Kern des zu entwickelnden Systems
- Kein Wegwerf-Prototyp
- Muss nach den Regeln der Kunst entwickelt werden
  - Planung + Steuerung
  - Überprüfung / Reviews
  - Dokumentation
- Evolutionäres Prototyping entspricht einem Wachstumsmodell

Abweichend von den übrigen Prototypen ist der evolutionäre Prototyp, der als Basis der weiteren Entwicklung erstellt wurde und als Pilotsystem dient weiterhin einsetzbar. Dies setzt allerdings voraus, dass die Entwicklung dieses Prototyps nach den Regeln der Kunst vorgenommen und entsprechend dokumentiert wurde.

Evolutionäre Prototypen werden sehr oft im Rahmen des evolutionären Entwicklungs-Modells als Entwicklungsbasis eingesetzt. Werkzeuge, die hierfür Anwendung finden werden als „Rapid Prototyping Tools“ bezeichnet und sind oft als Ergänzungswerkzeuge bei Datenbanken angesiedelt.

Prototyping und Prozessmodelle

- Prototyping ist kein eigenständiges Prozessmodell
  - Demonstrations-Prototypen
    - Unterstützen Akquisition und Aufsetzen von Projekten
    - Geben einen ersten Eindruck des Systems
  - Prototypen im engeren Sinn + Labormuster
    - Dienen der Risiko-Minderung
    - Sind in jedem Prozessmodell jederzeit einsetzbar
  - Entwicklung von Pilotsystemen
    - Eine Form des Wachstumsmodells

Prototypen werden oft als Akquisitionswerkzeug eingesetzt, da die Prototypen dem zukünftigen Anwender ein erstes Gefühl für die Anwendung vermitteln und das „Look&Feel“ der Anwendung bereits vor der eigentlichen Realisierung bereits sichtbar ist.

Wiederverwendung + Beschaffung

- Ziel
  Kosten senken für Software durch
  - Entwicklungs-Produktivität steigern
  - Möglichkeiten begrenzen
  - Stückzahl erhöhen
    - Das Geheimnis der billigen Hardware
    - Bei Software durch
      - Wiederverwendung
      - Beschaffung

Ein wichtiger Bestandteil der Software-Entwicklung ist der Einsatz fertiger Software-Module und/oder Werkzeuge, die vorgegebene Funktionalitäten in Form einer Bibliothek verfügbar machen.

Dies ist insbesondere im Bereich der Benutzerschnittstelle, der Schnittstelle zu Fremd-Softwaresystemen sowie im Bereich der Datenhaltung (Datenbanken) sinnvoll.

Der Einsatz bereits fertiger Software unterstützt den Erstellungsprozess, indem hier auf getestete Module zugegriffen wird und die damit verbundenen Aufwende wegfallen.

Durch Einsatz vorgefertigter professioneller Software-Module kann die Produktivität und die Fehlerfreiheit nachhaltig verbessert werden

Wiederverwendung

- Wiederverwendbare Software muss erst einmal geschaffen werden
  - Komponenten bilden, entwickeln + dokumentieren
- Entwicklung wiederverwendbarer Software ist teurer als die zweckgebundene Software
  - Explizite Anreize für Entwicklung wiederverwendbarer Software
- Wiederauffinden von Wiederverwendbarem
  - Kataloge + Beschreibungen
- Pflege von wiederverwendbarer Software
  - Pflege- / Wartungsverträge
- Förderung der Wiederverwendung
  - Handel mit Komponenten
  - Informationsstelle für Wiederverwendung

Wieder verwendbare Software ist nicht per Definition gegeben, sondern muss gezielt erstellt werden. Dies bedeutet, dass einerseits eine umfangreiche Dokumentation dafür zur Verfügung gestellt wird, die Software selbst nach professionellen Methode3n erstellt und getestet wird und die damit verfügbaren Funktionen durch eine Klassifikation jederzeit auffindbar sind.

Software-Pflege - Wartung

- Pflege (Wartung, „Maintenance“)
  Modifikation und / oder Ergänzung bzw. Erweiterung bestehender Software durch veränderte Software mit dem Ziel, Fehler zu beheben und das Produkt an veränderte Bedürfnisse oder Umweltbedingungen anzupassen oder die bestehende Software um neue Fähigkeiten zu erweitern
  - Systematische Erhaltung der Gebrauchstauglichkeit von Programmen
  - Nicht nur reine Fehlerbehebung
  - Bei Software vom P-Typ und E-Typ (nach Lehmann) zwingend
  - Kontinuierlicher Prozess über die gesamte Lebenszeit der Software

Pflege, Wartung (Maintenance) ist die Aktivität, die sicherstellt, dass Software über einen längeren Zeitraum auch unter veränderten Randbedingungen gebrauchstauglich und lauffähig ist.

Software-Pflege bedarf dabei ebenso der klaren Planung und Umsetzung nach den Regeln der Software-Engineering-Kunst.

Release / Version / Patch

- Sofortige Inbetriebnahme / Auslieferung jeder Änderung ist unmöglich / gefährlich
- Zusammenfassung von Änderungen in „Arbeitspakete" erforderlich
  - „Release" / „Version" oder „Patch"
- Release / Version
  Eine konsistente Menge von Komponenten eines Systems, die gemeinsam zur Benutzung freigegeben werden
- Patch
  Fehlerbehebung durch gezielte Nacharbeit und Austausch fehlerhafter Module eines Systems
- Komponenten
  Programm-Module, Daten, Steuerdateien, Dokumente
  - Identifikation erfolgt durch Nummerierung (z.B. 2.4, 3.7.5)
    - 1. Ziffer Haupt-Version / Major-Release (alle 1-3 Jahre)
    - Folgeziffern Unterversion / Unter-Release (alle paar Monate)
      (gerade: Funktions-Release, ungerade: Fehler-Release)

Bei der Weiterentwicklung und Pflege der Software unterscheidet man begrifflich zwischen folgenden Varianten:

- Release und/oder Version
  bezeichnet eine klar definierte Menge an Änderungen, die den Gebrauch der software sicherstellt und/oder sogar erweitert. Diese Elemente werden dabei gemeinsam freigegeben und verwaltet.
- Patch
  bezeichnet dabei ein Paket von Änderungen, die als schnelle Lösung zur Ausbesserung von Fehlern erforderlich sind
- Komponenten
  sind einzelne Module und Dateien.

  Die Kennzeichnung der Versionen und Releases erfolgt dabei firmenspezifisch nach einem eigenen Nummernsystem.

  Oft wird dabei zwischen Fehler- und Funktions-Releases unterschieden, so dass eine ungerade Endnummer ein Fehler-Release und eine gerade Endnummer ein Funktions-Release kennzeichnet.

Anforderungsanalyse

- Ziele der geplanten Software-Entwicklung klären
  - Management / Auftraggeber
  - Anwender / Betreuer
- Funktionalitäten mit den Anwendern erarbeiten
  - Fragenkatalog
  - Referenz-Systeme
- Abläufe, Datenmodell und Integration
  - Aufnahme der Arbeitsschritte Optimierung
  - Datenmodell erarbeiten
    - Erforderliche Daten und Zusammenhänge
    - Datenmengen und Verfügbarkeit / Datenübernahme
  - Integrationsanforderungen in andere Anwendungen

In der Anforderungsanalyse werden die Funktionen und Anforderungen an das System festgelegt und beschrieben.

Die Anforderungsanalyse kann dabei für verschiedene Anwendergruppen wie Endanwender, Management, Techniker ausgelegt sein und dient der Kommunikation dieser Gruppierungen.

Die einzelnen Anforderungen werden dabei von den „wissenden“ Anwendern erfragt, dies erfolgt mittels Interviews und/oder Fragebögen bzw. schriftliche Ausarbeitung durch die Anwender.

Zum besseren Verständnis können logische Abläufe in Form grafischer Ablaufdiagramme ausgearbeitet werden.

Phasen der Anforderungsanalyse

- **Durchführbarkeitsstudie**
  - Abschätzung, ob Bedürfnisse mit aktueller SW / HW in vertretbarem Rahmen erfüllbar sind
  - Kurzfristige + schnelle Entscheidungsgrundlage
- **Bestimmung + Analyse der Anforderungen**
  - Beobachtung bestehender Referenzsysteme
  - Diskussion + Befragung der Anwender (evtl. Prototyp)
- **Spezifikation der Anforderungen**
  - Umsetzung der Anforderungen in klare Beschreibungen
  - Erstellung des Pflichtenhefts + Detailbeschreibung
- **Validierung der Anforderungen**
  - Prüfung ob Anforderungen realistisch, konsistent + vollständig
  - Ergänzung der Anforderung bis zur Abnahme

Die Anforderungsanalyse umfasst dabei folgende Phasen:

- Machbarkeits-/Durchführbarkeitsstudie
- Analyse + Festlegung der Anforderungen
- Spezifikation der Funktionalitäten der Anforderungen
- Validierung der Anforderungen + Abnahme / Genehmigung

Generell sollte die Anforderungsspezifikation durch Unterschrift der Projektleiter abgesichert werden, so dass dadurch eine Planungssicherheit gewährleistet ist.

Entwurfsprozess

- Mehrere Modelle auf unterschiedlichen Abstraktionsebenen
  - Fehler + Lücken im bestehenden Entwurf beseitigen
  - Frühzeitige Überprüfung + Verbesserung des Modells
  - Phasen miteinander verknüpft und beeinflussen sich gegenseitig
- Ergebnis einer Phase ist Spezifikation für Folgephase
- Gesamtergebnis ist eine detaillierte Spezifikation der Funktionen, Datenstrukturen + Algorithmen
- Aktivitäten sind dabei (in Realität vermischt!)
  - Architektur-Entwurf mit Sub-Systemen + Abhängigkeiten
  - Abstrakte Spezifikation mit Funktionen + Randbedingungen
  - Schnittstellen-Entwurf – eindeutig + konkret
  - Komponenten-Entwurf mit Schnittstellen
  - Entwurf der Datenstrukturen und deren Abhängigkeiten
  - Algorithmen-Entwurf im Detail für Funktionen

Im Entwurfsprozess werden die möglichen Lösungskonzepte gegeneinander abgeglichen, um Fehler und Lücken der Spezifikation zu eliminieren.

Generell sind die Ergenisse einer Phase die Anforderungsspezifikation für die Folgephase.

Entwurfsmethoden

- Oft „Ad-Hoc-Prozess"
  - Willkürliche Sammlung von Anforderungen (häufig in natürlicher Sprache)
  - Informeller Entwurf des Systems
  - Basis für die Programmierung (oft ungenügend!)
  - Veränderung der Anforderungs-Spezifikation während der Umsetzung
  - Nach Systemabnahme ist Ursprungs-Spezifikation meist ungültig durch häufige Änderungen
    - Inkorrekt
    - unvollständig

Meist liegt der Anforderungsspezifikation kein systematischer Prozess zugrunde, sondern wird Ad-Hoc durchgeführt. Beschrieben werden diese Anforderungen zumeist in natürlicher Sprache, so dass es dabei oft zu Fehlinterpretationen kommen kann, da die zugrunde liegende Bedeutung der Begriffe nicht immer eindeutig geklärt ist.

Generell muss auch davon ausgegangen werden, dass sich die Anforderungen im Rahmen der Umsetzung verändern. Diese Veränderungen müssen einerseits festgehalten und genehmigt werden und andererseits die daraus resultierenden Folgen in die bestehende Planung eingearbeitet werden.

Änderungen sowohl in den Spezifikationen wie auch in der Planung unterliegen generell der Genehmigungspflicht und sind von den verantwortlichen Projektleitern und/oder Gremien zu unterzeichnen.

Entwurfsmethoden

- Strukturierte Methoden für den Systementwurf
  - Strukturierter Entwurf nach Constantin + Yourdon 1979
  - Strukturierte Systemanalyse nach Gane + Sarson 1979
  - Jackson System Development nach Jackson 1983
  - Methoden der objektorientierten Entwicklung
    z.B. nach Robinson 1992; Booch 1994; Rumbaugh 1991 u.a.
- Erstellen grafischer Systemmodelle + Entwurfsdokumente
- Einsatz von CASE-Werkzeugen für bestimmte Methoden
- Strukturierte Methoden werden erfolgreich eingesetzt
- Reduktion der Kosten weil Standards verwendet werden
- Kaum Unterschiede bei strukturierten Methoden
- Erfolg hängt von Eignung bzgl. des Einsatzes ab

Für den Systementwurf wurden auch einige systematische Methoden entwickelt, die sinnvoll und erfolgreich eingesetzt werden können.

Insbesondere im Zusammenspiel mit „Computer Aided Software Engineering“-Werkzeugen sind diese Methoden von Vorteil.

Strukturierte Methoden

- **Umfang der strukturierten Methoden**
  - Modell des Entwurfsprozesses
  - Notation zur Darstellung des Entwurfs
  - Berichtsformate
  - Regeln + Entwurfsrichtlinien
- **Unterstützung für**
  - Datenfluss-Modell mit Datenumwandlung
  - „Entity-Relationship-Modell"
  - Strukturelles Modell der Komponenten + Interaktionen
  - Objektorientierte Methoden
    - Vererbungsmodell
    - Modell der statischen + dynamischen Beziehungen
    - Modell bezgl. des Zusammenspiels der Objekte

Die Unterstützung des Entwurfsprozesses durch geeignete Methoden beschränkt sich meist auf die Vorgabe festgelegter Systematiken und Formate bzw. auf die Darstellung der Datenobjekte, deren Beziehungen untereinander sowie den Datenfluss zwischen den Objekten.

Strukturierte Methoden

- Zusätzliche Modelle
  - Zustandsdiagramme
  - Zentrales „Data Dictionary"
- Praxis
  - Anleitung eher informell
  - Daher unterschiedliche Entwürfe verschiedener Entwickler
  - Methoden sind in Wirklichkeit „Standard-Notationen" guter Entwicklungspraxis
  - Bei Strukturierung ist Kreativität sehr wichtig
  - Eigentlich nur Anleitung + Richtlinien mit Adaption

Hilfreich zur Darstellung der Abläufe sind dabei auch Datenfluss-Diagramme sowie ein „Zentrales Data-Dictionairy", das die Datenelemente umfassend beschreibt.

Meist beschränken sich dabei die Methoden aber auf die Festlegung von Standards und deren Empfehlung.

Software-Validierung

- Verifikation + Validation (V+V)
  - Zeigt den Erfüllungsgrad der Spezifikation
  - Zeigt die Erfüllung der Kunden-Erwartungen
  - Sieht Überprüfungsprozesse in jeder Phase vor
  - Hauptanteil beim Systemtest / Abnahmetest
- Phasen des Testprozesses
  - Einzeltests der Komponenten
  - Modultest – Test zusammenhängiger Komponenten
  - Subsystem-Tests –zusammenhängender Module
  - Systemtest – Test des Gesamtsystems
  - Abnahmetest – letzte Testphase mit dem Anwender

Durch Validierung wird der Erfüllungsgrad der Anforderungen durch die erstellte Software überprüft. Das Mass dafür ist dabei einerseits die festgeschriebene Spezifikation und andererseits der Anwender, der das System als Pilotsystem auf seine Gebrauchtauglichkeit testet.

Automatisierte Prozess-Unterstützung

- Computer Aided Software Engineering („CASE")
  - Wird benutzt um Software-Prozesse zu unterstützen
    - Anforderungsanalyse
    - Entwurf + Programmierung
    - Integration + Test
  - Für die meisten Routine-Aktivitäten verfügbar
  - Verbesserung der Produktivität + Qualität (ca. 40%)
  - Einschränkende Faktoren
    - Entwurfsaktivität – gesteuert durch Kreativität (Automatisierung der Routine-Aufgaben hilft hier nicht)
    - Software-Engineering ist Team-Arbeit, erfordert viel Kooperation + Kommunikation (kaum Unterstützung durch CASE-Tools)

Die Basis für weit reichende Tests wird durch so genannte „CASE-Tools (Computer-Aided-Software-Engineering) zur Verfügung gestellt.

Dabei wird der reine Entwicklungsprozess – Programm-Implementierung – Test – Verifikation – Fehlersuche – am stärksten unterstützt, da dieser Prozess sehr stark systematisch betrieben werden kann.

Insbesondere kann ein geeignetes Werkzeug dabei die Nachvollziehbarkeit der Korrektur durch automatisch ablaufende und systematische Tests und Code-Interpretation sicherstellen.

Durchgeführte Tests werden in einer „Log-Datei" automatisch dokumentiert und dadurch jederzeit wieder exakt wiederholbar.

CASE-Klassifizierung

Werkzeugtyp	*Beispiele*
Planungswerkzeuge	Netzplan-Werkzeug, Werkzeug zur Kosten-/Aufwandsschätzung, Tabellen-Kalkulator
Editoren	Text-Editoren, Diagramm-Editoren, Textverarbeitungsprogramme
Werkzeuge für Änderungs-Management	Werkzeuge zur Verfolgung der Anforderungen, Änderungskontroll-Systeme
Werkzeuge für Konfigurations-Management	Versions-Kontroll-Systeme Werkzeuge zur System-Erstellung
Werkzeuge zum (Rapid-)Prototyping	Abstrakte Spezifikationssprachen Generatoren für Bediener-Oberfläche
Werkzeuge zur Methoden-Unterstützung	Entwurfs-Editoren, Data-Dictionaries, Codegeneratoren, 4GL-Werkzeuge

Die Klassen der durch CASE verfügbaren Werkzeuge umfasst dabei

- Planungswerkzeuge
- Editoren + Testmanager
- Produkt- und Änderungs-Management
- Konfigurations-Management
- Werkzeuge zum Rapid Prototyping
- Methoden-Unterstützung und Code-Generatoren

CASE-Klassifizierung

Werkzeugtyp	*Beispiele*
Sprachverarbeitende Werkzeuge	Compiler, Interpreter
Werkzeuge zur Programm-Analyse	Generatoren für Querverweise, statische + dynamische Analyse-Programme
Test-Werkzeuge	Generatoren für Testdaten, Test-Simulatoren Programme zum Vergleich von Dateien
Werkzeuge zur Fehlerbehebung	Interaktiver Debugger
Dokumentations-Werkzeuge	Seiten-Layout-Programme Bildbearbeitungs-Programme
Re-Engineering-Werkzeuge	Querverweis-Systeme, Systeme zur Neustrukturierung von Programmen

Ebenso werden Werkzeuge zur verfügung gestellt für

- Die Übersetzung in Maschinencode bzw. direkte Ausführung im Interpretations-Mode
- Werkzeuge zur Programm-Analyse, die Schwachstellen und Fehlerquellen in der Programmierung aufzeigen, auf Risikoquellen hinweisen
- Test-Werkzeuge wie Debugger, die eine schrittweise Ausführung der Programme ermöglichen
- Dokumentations-Werkzeuge, die einen entsprechenden Rahmen vorgeben, damit die Dokumentation einen einheitlichen Charakter erhält
- Bis hin zu Re-Engineering-Werkzeugen, die eine Rückwärts-Kompilierung des Maschinencodes in eine höhere Programmiersprache ermöglichen

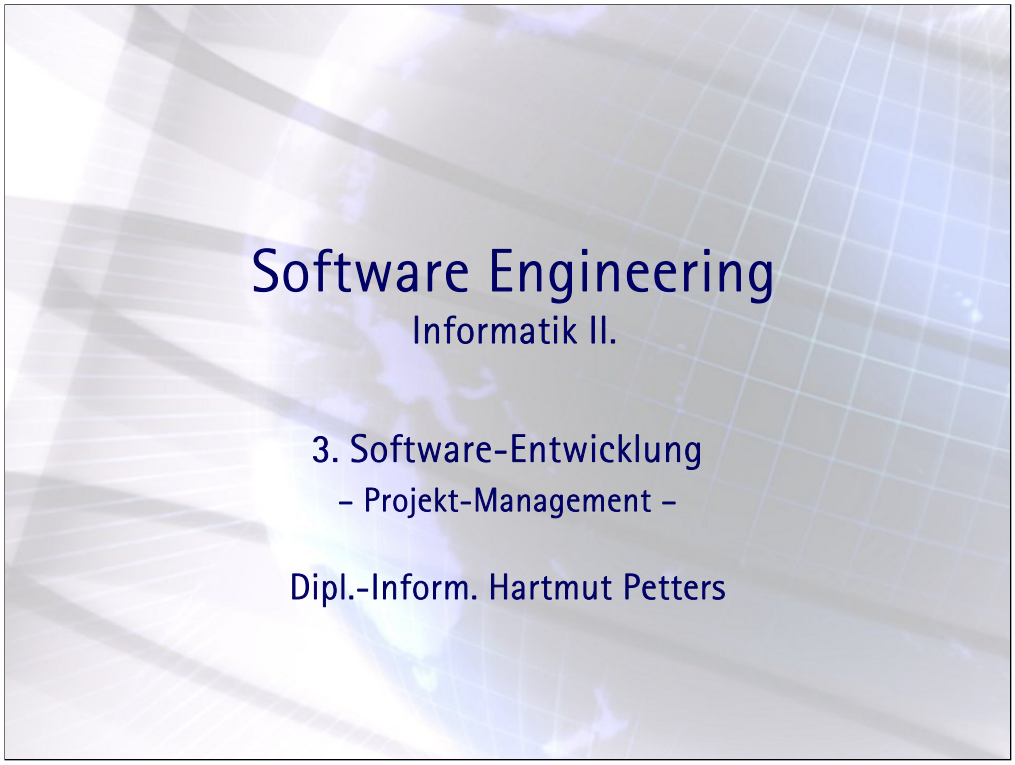
Zusammenfassung – Software-Prozesse

- Aktivitäten zur Software-Herstellung
- Softwareprozesse bestehen aus den Phasen
  - Spezifikation
  - Entwurf
  - Implementierung
  - Validierung
  - Weiterentwicklung
- Vorgehensmodelle beschreiben den Aufbau der Software-Prozesse (z.B. Wasserfall-Modell)
- Iterative Vorgehensmodelle – Zyklus von Aktivitäten
- Anforderungs-Analyse zur Entwicklung der Spezifikation
- Entwurf + Implementierung = Umwandlung in Software
- Software-Validierung = Überprüfung der Ergebnisse
- Weiterentwicklung ist die Evolution von Software
- CASE-Tools zur Unterstützung der einzelnen Phasen

Der Software-Prozess umfasst alle Aktivitäten und Phasen zur Software-Herstellung.

Entsprechende Modelle sind bereits seit vielen Jahren erfolgreich im Einsatz und wurden an die steigende Komplexität der Software adaptiert wie z.B. das Spiral-Modell.

In vielen Bereichen existieren heute bereits mächtige Werkzeuge (CASE-Tools), die einzelne Phasen im Entstehungsprozess sehr effizient unterstützen. Dies insbesondere direkt in der Realisierungs- und Test-Phase.



Da Software ein immaterielles Produkt ist, führt eine ad-hoc-Abwicklung bei Software-Projekten wesentlich schneller ins Desaster als bei konventionellen Projekten.

Daher ist es unabdingbar Software systematisch zu entwickeln und als Software-Projekt systematisch zu planen und zu steuern.

Insbesondere sind die mit dem Software-Projekt verbundenen Risiken zu betrachten und zu bewerten

Besonderheiten bei Software-Projekten

- Was ist bei Software-Projekten anders?
  - Software ist immateriell!

    ↳ Führung durch genaues Hinsehen funktioniert nicht!
    - ↳ Sorgfältige Planung
    - ↳ Ständige Überprüfung + Kontrolle
    - ↳ Permanente Steuerung
    - ↳ Beachtung aller Projektrisiken

11.1.2004 67 © by Hartmut Petters

Projekte, die eine gewisse Komplexität überschreiten, können i.d.R. nur durch ein klares Management der einzelnen Komponenten erfolgreich und sinnvoll abgewickelt werden.

Die Projektplanung schafft dabei die notwendigen Voraussetzungen für das Gelingen eines Projekts. Neben der Definition der zu erreichenden Ziele geht es hier vor allem darum, den Weg zum Ziel und die dabei benötigten Mittel (Ressourcen) festzulegen. Dabei sind im Wesentlichen folgende Aufgaben zu erledigen:

- Festlegung des Prozessmodells + der Organisationsstruktur
- Bestimmung der benötigten Personen + deren Einsatzplanung
- Benötigte Qualifikationen + Beistellungen
- Termin- und Kostenplan
- Festlegung der eingesetzten Methoden und Verfahren
- Konfigurations- und Qualitäts-Management
- Risikoabschätzung
- Kontroll-Organe + Berichtswesen

Arten von Software-Projekten

- Entwicklungsprojekt
  - Interner Auftraggeber (Hersteller)
  - Spezifikation vorgegeben
    - Marktuntersuchung
    - Interne Planungen
  - Software Engineering
    - Planung, Realisierung
    - Neuerstellung, Weiterentwicklung
  - Abnahme, Erfolgskontrolle
    - Interne Abnahmetests, Testkunden
    - Erfüllungsgrad der Spezifikation
  - Ergebnis: Software-Produkt

Im täglichen Einsatz gibt es generell zwei Arten von Software-Projekt-Typen:

Eine davon ist das „Entwicklungsprojekt“, dessen Ziel es ist ein Software-Produkt zu erstellen, das entweder als spezielle Software für genau eine Firma realisiert wird (Auftragsprojekt) oder als Produkt, das mehrfach zum Einsatz kommen soll.

Dementsprechend gelten die angeführten Charakteristika.

Da das Ergebnis sich in Form eines „langlebigen“ Software-Produkts niederschlägt, sollte die Weiterentwicklung durch eine systematische Entwicklung und Dokumentation sichergestellt werden.

Arten von Software-Projekten

- Einführungsprojekt
  - Externer Auftraggeber (Endkunde)
  - Spezifikation abhängig
    - Leistungsfähigkeit des Software-Produkts
    - Kundenumfeld / Geschäftsprozesse
  - Software Engineering
    - Prozess-Analyse, Integration, Einführungskonzept
    - Anpassung, Erweiterung
  - Abnahme, Erfolgskontrolle
    - Pilotanwendern, Endanwender
    - Integration in Kundenumfeld
  - Ergebnis: lauffähige Kundenanwendung

Die 2te Art von Software-Projekt ist das Einführungsprojekt, bei dem ein bestehendes Standard-System wie z.B. SAP R/3 in einem Unternehmen eingeführt wird und die Software einerseits durch Konfiguration (Customizing) an die firmenspezifischen Belange angepasst bzw. durch gezielte Zusatz-Entwicklung noch stärker an die Anforderungen des Unternehmens adaptiert wird.

Durch den Einsatz eines standardisierten Systems sind bereits viele Fehlerquellen ausgeschlossen. Trotzdem sind alle Änderungen, Anpassungen, Erweiterungen und Ergänzungen ausführlich zu entwickeln, zu testen und zu dokumentieren. Dies ist insbesondere deswegen wichtig, weil durch die Weiterentwicklung des Systems und die damit verbundenen Änderungen auch Änderungen an den Anpassungen, Erweiterungen und Ergänzungen erforderlich werden können.

Projektmanagement - Kritische Erfolgsfaktoren

- Commitment und aktive Unterstützung durch das Management
- erfahrene, durchsetzungsstarke und anerkannte Projektleiter
- ausgewogenes, motiviertes Projekt-Team
- fachlich, organisatorisch, wirtschaftlich und terminlich abgesicherte Projektdefinition
- präzise und realistische Planung

Projektmanagement ist eine der anspruchvollsten Aufgaben im Umfeld der Software-Entwicklung.

Die erfolgreiche Umsetzung von Projekten benötigt deswegen auch eine optimale Unterstützung und eine optimale Basis. Diese Basis ist nur gegeben, wenn

- Ein klares Commitment und eine aktive Unterstützung des Top-Managements vorhanden ist
- Der Projektmanager über hinreichend Erfahrung verfügt und durchsetzungsstark ist, Dazu gehört auch die Akzeptanz innerhalb der Projekt- und Firmen-Organisation.
- Ein ausgewogenes, motiviertes und kompetentes Team die Realisierung macht
- Alle Rahmenbedingungen fachlich, organisatorisch, wirtschaftlich und terminlich abgesichert sind
- Eine präzise, korrekte und realistische Projektplanung gemacht wurde.

Planung bei Software-Projekten

- Was muß geplant werden?
  - Der Prozess + das Prozess-Modell
  - Die Organisationsstruktur
  - Personal + Personaleinsatz
  - Termine + Kosten
  - Dokumente + Verfahren
- Wie wird geplant?
  - Sach- + zielorientiert
  - Anspruchsvoll, aber realistisch
  - Aufgaben + Ressourcen im Gleichgewicht

Für Software-Projekte müssen insbesondere folgende Rahmenbedingungen geklärt sein:

- Der gesamte Entwicklungs-Prozess sowie das einzusetzende Entwicklungsmodell, nachdem die Entwicklung gesteuert werden soll
- Die Organisationsstruktur und deren Einbindung
- Das verfügbare Personal, die vorhandenen Leistungsprofile sowie deren Einsatz über die Zeit
- Termine und Kosten
- Dokumente, Methoden und Verfahren

Die Planung selbst erfolgt

- sach- und zielorientiert im Hinblick auf die Anforderungen und Vorgaben
- Anspruchsvoll aber realistisch, so dass auch bei auftretenden Problemen nicht sofort die gesamte Planung in Frage gestellt ist.
- Aufgaben und Ressourcen im Gleichgewicht, so dass eine durchgängige und gleichmäßige Auslastung aller Team-Mitglieder sichergestellt ist.

Der Projektplan

- Dokumentiert
  - Alle Ergebnisse der Planung
  - Die Plan-Fortschreibung
- Gibt Antwort auf 6 W-Fragen
  - Warum? Veranlassung + Projektziele
  - Was? Zu liefernde Resultate (Produktziele)
  - Wann? Geplante Termine
  - Wer? Personen + Verantwortlichkeiten
  - Womit? Verfügbare Mittel (Geld, Geräte, ...)
  - Wie? Vorgehensweise + Maßnahmen zur Sicherstellung des Projekterfolgs

11.1.2004 72

Der Projektplan ist die Basis für die Umsetzung und das Werkzeug des Projektmanagers zur Steuerung des gesamten Projekts. Der Projektplan dokumentiert

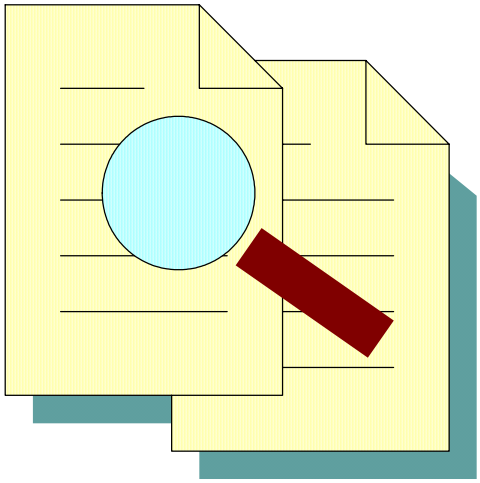
- Alle Ergebnisse der Planung (SOLL-Planung) sowie
- Den Projektfortschritt

Der Projektplan beantwortet sie Fragen:

- Warum wird das Projekt realisiert + was sind die Ziele
- Welche Resultate werden wann erwartet
- Wie sind die Termine über die gesamte Projektlaufzeit
- Wann können Zwischenrechnungen gestellt werden
- Welche Personen sind wie und wann eingeplant
- Welche Mittel stehen zur Verfügung und können wie eingesetzt werden
- Vorgehensweisen + Maßnahmen, Aktivitätenplan für die nächsten Tage, Wochen und Monate

Mögliche Projektplan-Gliederung

1. Einleitung
2. Ziele
3. Arbeitsplan
   (1) Arbeitspakete
   (2) Lieferungen + Abnahmen
   (3) Risiken
4. Terminplan
5. Personalplan
   (1) Projekt-Organigramm
   (2) Personal-Einsatzplan
6. Kostenplan
7. Sonstige Ressourcen
8. Vorgehen



Der Projektplan sollte neben den reinen terminlichen Festlegungen auch alle Informationen enthalten über

- Ziele der Entwicklung
- Aufteilung des Projekts in Arbeitspakete und deren Zuordnung zu Ressourcen
- Liefer- + Abnahme-Termine
- Einen umfassenden Personalplan einschließlich erforderlicher Schulungsmaßnahmen
- Kostenkalkulation einschließlich einer mitlaufenden Kalkulation
- Sowie alle Aktionen, die zur Erfüllung der Aufgabe notwendig sind

Projektkontrolle + Projektlenkung

- Fortschrittskontrolle ist unabdingbar!
- Ergebnisse müssen messbar sein
  - 90%-Syndrom!
- Bei Abweichung
  - Lenkung!
- Rahmen für Kontrolle + Lenkung
  - Termin-Verfolgung
  - Sachziel-Verfolgung
  - Kosten-Verfolgung
  - Risiko-Verfolgung

Ergebnis-Kontrolle ist unabdingbar und sollte auf der Basis der in der Spezifikation festgelegten Kriterien erfolgen

Sachziel- + Terminverfolgung

- Mit Hilfe definierter Meilensteine
- Planung
  - Meilensteine strukturieren die Sachziele in Teilziele
  - Festlegung der SOLL-Termine für alle Meilensteine
- Meilenstein erreicht
  - Zwischenziel nachweislich erreicht
  - Gesicherte quantitative Aussage der Termin-Situation durch SOLL-IST-Vergleich
- Meilenstein am geplanten Termin nicht erreicht
  - Schätzung der Termin-Situation durch Schätzung des verbleibenden Aufwands bis Zielerreichung

Meilensteine sind das Mittel zur Steuerung der Projekte

Meilensteine sollten in Abhängigkeit der Sachziele und den sich daraus abgeleiteten Termine festgelegt werden.

Nur konkrete Messkriterien, die überprüfbar sind, sind geeignet um den Erfüllungsgrad der Anforderungen festzustellen und zu klären, ob der vorgesehene Meilenstein erreicht wurde.

Abweichungen sind entsprechend zu dokumentieren und in die Planung einzuarbeiten. Dabei sind alle Folgereaktionen und daraus resultierenden Abweichungen zu berücksichtigen.

Jegliche Änderung der Planung ist möglichst frühzeitig festzustellen und mit dem Kontroll-Gremium abzustimmen.

Kostenverfolgung

- Scheinbar einfach
  - Aufzeichnen budgetierter + tatsächlicher Kosten über die Zeit
  - „De Facto" unbrauchbar!
- Alternativen
  - Kosten + fiktive Erträge auf einer Zeitachse
  - Kosten auf einer Meilensteinachse
    - SOLL-Kosten am geplanten Meilenstein-Termin mit IST-Kosten bei Erreichung des Meilensteins vergleichen

Auch wenn die Kostenverfolgung aufgrund der klaren Messgröße einfach scheint, so liegt es in der Natur der Projekte, dass ein einfacher Abgleich der aufgebrauchten Mittel mit den Zielvorgaben nicht hinreichend ist.

Da nicht der Mittelverbrauch das Maß für den Erfüllungsgrad der Anforderungen ist, sondern einzig und allein die bereits erstellten Funktionalitäten, ist der Mittelverbrauch im Kontext der erstellten Funktionalität zu betrachten. Dies bedeutet, dass bei 50%igem Mittelverbrauch auch 50% der Funktionalität wie geplant verfügbar sein sollte, da ansonsten Abweichungen in der Projektkalkulation die Folge sind.

Das bedeutet andererseits, dass die kosten immer nur direkt im Abgleich mit den erstellten Funktionen betrachtet werden können.

Verantwortlichkeiten + Berichtswesen

- Verantwortlichkeiten + Kompetenzen aller Beteiligten klar geregelt
- Keine Übertragung von Verantwortung ohne die dafür notwendigen Kompetenzen + Ressourcen
- Stufengerechtes Umgehen mit Problemen
- Berichtswesen
  - Keine bürokratische Schikane ...
  - ... sondern Frühwarnfunktion
- Arbeitspaket-Ordner als Organisations-Hilfsmittel

Innerhalb der Projektorganisation sind alle Verantwortungen und Kompetenzen klar zu regeln, so dass jeder die durch seine Aufgabe erforderlichen Rechte und Pflichten klar zugeordnet hat.

Alle relevanten Entscheidungen und Festlegungen sind durch den Verantwortlichen zu dokumentieren und zu kommunizieren, so dass der Projektmanager jederzeit über alle relevanten Ereignisse informiert und auskunftsfähig ist.

Maßnahmen bei Abweichungen

- Abweichungen
  - ↳ Gegenmaßnahmen
- Beispiel: Terminüberschreitung
  - Bereitstellung zusätzlicher Ressourcen
  - Befreiung der Projektmitarbeiter von anderen Verpflichtungen
  - Anordnung von Überstunden / Urlaubsstreichung
  - Vergabe von Teilaufträgen an Dritte
  - Abstriche bei den geplanten Funktionalitäten
  - Neuplanung der Leistungsziele (Wachstumsmodell)
- Falls Gegenmaßnahmen versagen oder nicht möglich
  - ↳ Planung anpassen

Abweichungen gegenüber Planung sind kurzfristig festzustellen und durch geeignete Gegenmaßnahmen zu korrigieren, im Sinne des festgelegten Projektplans und der Einhaltung dieser Rahmenplanung

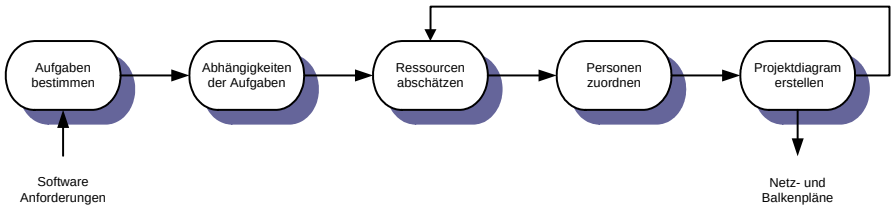
Projektmanagement - Softwareprojekte

- Management-Aufgaben
  - Erstellung + Kalkulation des Angebots
  - Projekt- und Zeitplan
  - Aufgabenpakete + Meilensteine
  - Lieferungen + Abnahme
  - Projektkostenkalkulation
  - Aufgaben-Planung + Zuteilung
  - Ressourcen-Management + Qualifizierung
  - Kostenkontrolle + Kosten-Management
  - Projektüberwachung + Reviews
  - Auswahl und Beurteilung des Personals
  - Erstellen + Präsentation von Berichten

Die Aufgaben des Projektmanager umfassen dabei alle Maßnahmen zur zielgerichteten Projektplanung, -Steuerung und –Kontrolle, wie

- Projektkalkulation, Aufwandsabschätzung und Angebotserstellung bis hin zur Aufteilung der Anforderungen in Arbeitspakete und deren terminliche Planung
- Festlegung der Meilensteine sowie deren Grundlage wie zugrunde liegende Prüfkriterien und terminliche Einplanung
- Ressourcen.Management bis hin zur Ausbildungsplanung des Teams
- Kostenplanung und Kontrolle
- Projektüberwachung und Projekt-Reviews bis hin zur Qualitätskontrolle

Projektplanung - Softwareprojekte

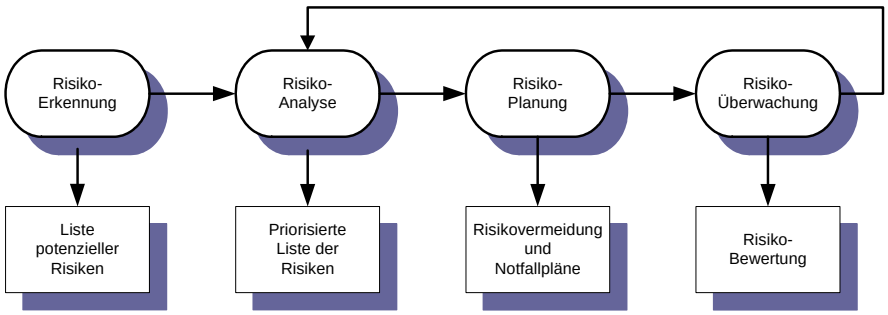


- Mögliche Probleme + Risiken vorhersehen
- Lösungen erarbeiten + präsentieren
- Aufgaben definieren + zuordnen
- Pflege + Weiterentwicklung der Planung
- Risikoanalyse + Risikominimierung

Bereits in der Planungsphase werden mögliche Risiken diskutiert, abgeschätzt und mögliche Alternativ-Konzepte erörtert.

Alle Risiken werden im Hinblick auf die Auswirkung auf das Projekt betrachtet und Notfallpläne hierfür erarbeitet.

Softwareprojekt - Risikomanagement



- Risiko-Management ist Erkennen + Minimierung möglicher Projekt-Risiken + Ergreifen geeignete Maßnahmen
- Generell Risiken vorhersehen, die sich auswirken auf
  - Projektablauf / Zeitplan
  - Qualität des Ergebnisses
  - Kosten des Projekts

Risiko-Management ist eine der wichtigsten Aufgaben des Projektmanagements.

Ziel ist es alle möglichen Risiken so einzuplanen, dass im Falle des Eintritts deswegen das Projekt nicht gefährdet wird.

Risiken bei Software-Projekten

- **Personal** (Projekt)
  - Erfahrenes Personal verlässt das Projekt vor Projektende
- **Management** (Projekt)
  - Management-Wechsel bringt Änderung in den Prioritäten
- **Hardware-Verfügbarkeit** Projekt)
  - Unverzichtbare Hardware ist nicht verfügbar
- **Veränderung der Anforderungen** (Projekt / Produkt)
  - Gravierende Änderungen der Anforderungen stellen die Planung in Frage
- **Verzögerungen bei der Spezifikation** (Projekt / Produkt)
  - Spezifikation von Funktionen / Schnittstellen erheblich verspätet
- **Fehleinschätzungen des Arbeitsumfangs** Projekt / Produkt)
  - Aufwand für Module / Funktionen / Schnittstellen / etc. werden unterschätzt
- **Mangelhafte CASE-Werkzeuge** (Produkt)
  - Produktivität leidet unter der schlechten Qualität der Werkzeuge
- **Technologieveränderungen** (Wirtschaftlichkeit)
  - Zugrundeliegende Technologie wird durch neue verdrängt
- **Produktkonkurrenz** (Wirtschaftlichkeit)
  - Konkurrenzprodukte sind vor dem eigenen Produkt verfügbar

Entsprechend der Aufstellung werden alle projektspezifischen Risiken diskutiert und entsprechende Lösungen erarbeitet, die im Notfall sofort umgesetzt werden können, so dass sich die aus dem Risiko ergebenden Folgen nur in einer vordefinierten weise in der Projektabwicklung auswirken.

Die 10 häufigsten Software-Risiken

1. Zu wenig Leute
2. Unrealistische Kosten- und Terminpläne
3. Falsche Funktionalitäten
4. Falsche Benutzerschnittstellen
5. Vergoldung (überflüssiger Luxus)
6. Sich ständig verändernde Anforderungen
7. Probleme mit zugekauften Komponenten
8. Probleme mit extern vergebenen Aufträgen
9. Nichterreichung der verlangten Leistungen (z.B. Reaktionszeit)
10. Überforderung der Mitarbeiter in Bezug auf ihr Software-technisches Können

Quelle: Boehm (1991)

Boehm hat die 10 häufigsten Risiken in Software-Projekten zusammengefasst und schlägt vor diese einzeln im Projekt zu betrachten und entsprechende Notfall-Konzepte präventiv zu erarbeiten.

Risiko-Erkennung – 1. Schritt

- Technologische Risiken aus Funktionalität + Verfügbarkeit von Hardware + Software
- Personenbezogene Risiken bzgl. des Einsatzes + der Verfügbarkeit des Entwicklungs-Teams
- Risiken durch „Werkzeug-Einsatz" im Projekt
- Anforderungsrisiken durch Änderungen und/oder Erweiterungen der Funktionalitäten
- Schätzrisiken aus Fehlabschätzungen bzgl. der System-/Projekt-Charakteristik + Ressourcen

Risiko-Erkennung ist der erste Schritt zum Risiko-Management.

Denn nur, wenn ich die Risiken kenne, kann ich frühzeitig entsprechende Gegenmaßnahmen ergreifen

Risiken + Arten von Risiken

Risiko-Art	Mögliche Risiken
Technologisch	DB kann nicht so viele Transaktionen ausführen / Sek. wie erwartet. Zur „Wiederverwendung" vorgesehene Software-Komponenten enthalten Fehler, die die Funktionalität einschränken
Personen-bezogen	Nicht genügend Personal mit benötigten Skills rekrutierbar Schlüsselpersonen sind krank oder nicht verfügbar Erforderliche Schulungen sind nicht durchführbar
Unternehmens-bezogen	Umstrukturierung des Unternehmens, dadurch Wechsel der Zuständigkeit für das Projekt. Finanz-Probleme zwingen zu Kürzungen der Projekt-Budgets
Werkzeuge	Der durch CASE-Werkzeuge generierte Programm-Code ist ineffizient CASE-Werkzeuge können nicht eingebunden werden
Anforderungen	Nachträgliche Änderungen der Anforderungen ziehen erhebliche Nacharbeit des Entwurf nach sich.
Schätzung	Entwicklungszeit wird zu niedrig eingeschätzt Zu geringe Schätzung für Fehlerbehebung + Software-Umfang

Generell sind die Risiken in folgende Klassen einzuteilen:

- Technologie-Risiken durch Einsatz unbekannter Technologien
- Personenbezogene Risiken, die durch Einsatz singulärer Ressourcen vorkommen können
- Unternehmensbezogene Risiken, die aufgrund Umstrukturierung und Finanzunsicherheiten auftreten können
- Risiken durch den Einsatz von Werkzeuge, die nicht erprobt und nicht bekannt sind
- Änderungen der Anforderungen während der Laufzeit sind unvermeidlich und sind deswegen in gewissem Umfang einzuplanen
- Fehleinschätzung bei der Aufwandsabschätzung, so dass die zugrunde liegende Planung gefährdet ist.

Risiko-Analyse – 2. Schritt

- Alle erkannten Risiken betrachten + beurteilen
  - Wahrscheinlichkeit
  - Konsequenzen
- Stark abhängig von der Erfahrung des Managers
- Tabellarische Auflistung der Risiko-Abschätzung
- Basis für Risiko-Analyse
  - Informationen über das Projekt
  - Informationen über das Projekt-Team
  - Informationen über das Unternehmen
  - Informationen über den Markt
  - Informationen über den Wettbewerb
- Änderung der Analyse-Ergebnisse durch zusätzliche Informationen ist jederzeit möglich

Erkannte Risiken sind zu beurteilen + zu klassifizieren nach Wahrscheinlichkeit und den sich daraus ergebenden Konsequenzen.

Das Risiko-Management und der Umgang mit den erkannten Risiken ist sehr stark abhängig von dem bestehenden Erfahrungshintergrund des Projektmanagers.

Generell gilt aber, dass alle Schritte im Bereich Risiko-Management nicht alleine, sondern im Team erfolgen sollten.

Risiko-Planung

- Alle erkannten Haupt-Risiken betrachten + Strategie zum Risiko-Handling entwickeln
- Abhängig von der Erfahrung des Projekt-Managers
- Strategien zum Risiko-Handling
  - Vermeidungs-Strategie
    Ziel: Wahrscheinlichkeit für das Auftreten des Risikos verkleinern
  - Minimierungs-Strategie
    Ziel: Auswirkungen des Risikos zu reduzieren
  - Notfall-Pläne
    Ziel: Einstellen auf den schlimmsten Fall um vorbereitet zu sein und Lösung parat ist

Durch die Betrachtung und Kalkulation aller erkannten Haupt-Risiken können bereits im Vorfeld entsprechende Handlungsmaßnahmen und Abwehrstrategien entwickelt werden, die im Falle des Eintritts kurzfristig umgesetzt werden können

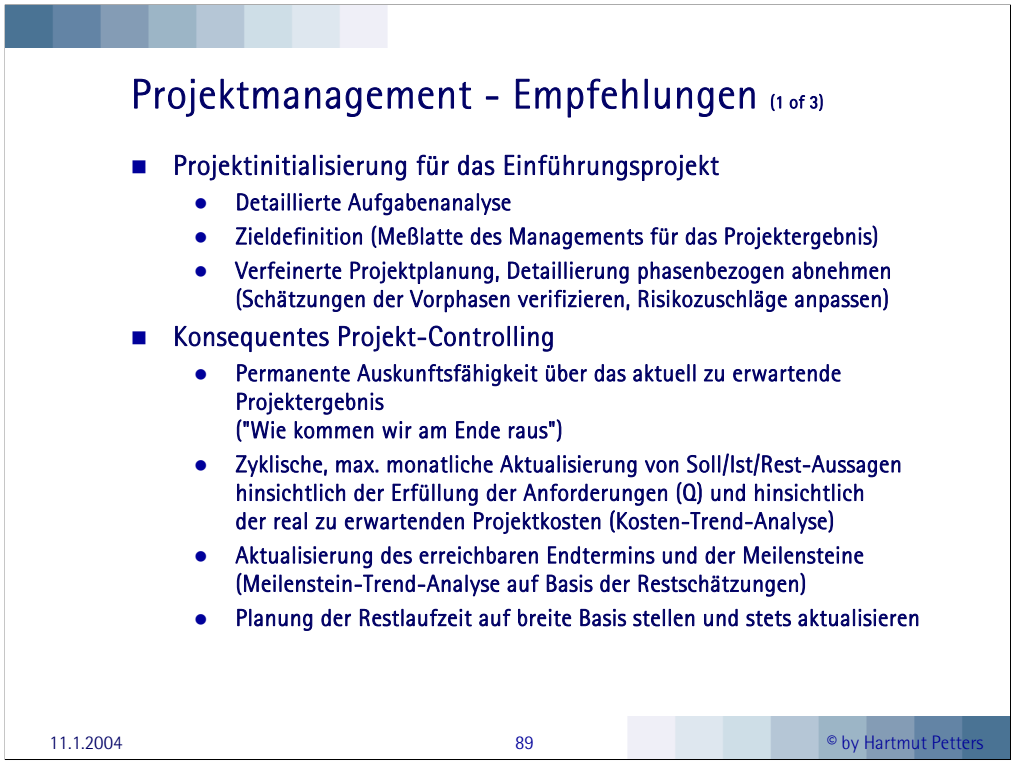
Risikomanagement - Risikoüberwachung

- regelmäßige Beurteilung erkannter Risiken
- Weitere Faktoren zur Beurteilung heranziehen
- Risikoüberwachung ist ein fortlaufender Prozess
- Bei Meilenstein-/Fortschritts-Review Hauptrisiken einzeln betrachten + erörtern
- Beurteilung + Risikoabschätzung gemeinsam mit dem Team durchführen (Konsens-Entscheidung)
- Maßnahmen definieren zur Verbesserung des Risikomangements

Risiko-Management und Risiko-Überwachung sind keine einmalige Aktion sondern begleiten das Projektmanagment während der gesamten Projektlaufzeit.

Abhängig von der Phase, in der sich das Projekt befindet, können dabei Risiken verschwinden oder neue hinzukommen, die sofort in diese Risikobetrachtung einzubeziehen sind.

Das Risiko-Management sowie die Darstellung der möglichen Risiken im rahmen der Kontrollausschüsse gehört zu den Pflichten des Projektmanagers.



- Als Basis für das Projektmanagement muss unbedingt eine detaillierte Aufgabenanalyse erfolgen, die neben der Spezifikation auch eine klare Zieldefinition und die zugehörigen Messkriterien enthält
- Nur durch konsequentes und kontinuierliches Projekt-Controlling kann der Projekterfolg sichergestellt werden.

Projektmanagement - Empfehlungen (2 of 3)

- **Nachvollziehbare Projektsteuerung**
  - Steuernde Eingriffe nicht ohne vorhergehende Standortbestimmung durch das Projekt-Controlling (Was sagt das Konzept? Was steht im Pflichtenheft? Was war geplant? Was gewinnen, was verlieren wir?)
  - nicht: die Anstrengungen verdoppeln, wenn das Ziel aus den Augen verloren wurde!
  - Maßnahmen mit Auswirkung auf das vereinbarte Projektziel müssen vom Lenkungsausschuss abgesegnet werden!

- Projektsteuerung darf nur in kontrollierter Art und Weise erfolgen, d.h. ein Eingriff in die Projektsteuerung muss generell vorher durch eine Analyse und eine Abstimmung mit den Steuergremien abgesichert sein.
- Der Projektmanager trifft keine unternehmerischen Entscheidungen, sondern sorgt nur für die zielgerichtete Umsetzung der Entscheidungen. Die Entscheidungen werden nur durch die dafür vorgesehenen Gremien oder den Auftraggeber getroffen.

Projekt-Management - Empfehlungen (3 of 3)

- Zielführendes Änderungs-Management
  - Änderungen sind unvermeidbar; z.B. aus folgenden Gründen:
    - „Customizing" einzelner Funktionen nicht machbar, aufwendiger
    - Erfolgskritische Anforderungen wurden nicht identifiziert
    - Schnittstellen leisten nicht den erforderlichen Funktionsumfang
    - etc.....
  - Jede Änderung der Projektziele, des Soll-Konzepts oder des Pflichtenhefts, etc. - und sei sie noch so klein - muss vor dem Hintergrund ihrer Auswirkungen auf das Projektergebnis bewertet und bewusst durchgeführt werden.
  - Das Projekt-Team trifft keine unternehmerischen Entscheidungen (bei Projekt-typischen unternehmensübergreifenden Entscheidungen den Lenkungsausschuss und ggf. den Auftraggeber und wesentliche Promotoren einbeziehen.

Änderungs-Management bedeutet, dass vor der Durchführung von Änderungen der aktuelle Stand festgestellt, geeignete Maßnahmen aus der Ist-Situation abgeleitet und entsprechend umgesetzt werden.

Jede noch so kleine Änderung ist im Hinblick auf deren Auswirkungen auf das globale Projektziel hin zu überprüfen und in die gesamte Projektplanung einzuplanen.

Zusammenfassung

- Ein Projekt ist ein außergewöhnliches Vorhaben
- Aufgrund des unternehmensweiten Ansatzes sind die Akzeptanz, die Mitarbeit und die Unterstützung der Betroffenen jeweils ein wesentlicher Erfolgsfaktor.
- Der Aufwand für Kommunikation und Schnittstellen-Management zwischen den Mitwirkenden ist hoch.
- Die Projektplanung, das Projekt-Controlling und die Projektsteuerung sind anspruchsvoll.
- Der Projektmanager hat eine Schlüsselposition

Da Projekte immer einmalige Umstände mit sich bringen und generell mit nicht vorhersehbaren Ereignissen gerechnet werden muss, erfordert dies vom Projektmanager ein hohes Maß an analytischem Denken und Entscheidungsfähigkeit.

Der Projektmanager muss aus diesem Grund jederzeit über alle Aktionen im Projekt und über die laufenden Aktivitäten informiert sein.

Zusammenfassung

- Gutes Management bei Software-Projekten ist zwingend
- Management von Software-Projekt unterscheidet sich von herkömmlichen Projekten, da
  - Immaterielles Gut
  - Einsatz neuer, innovativer Technologien
  - Wenig Erfahrungswerte
  - Wissenschaftlich kaum erforscht
- Software-Manager spielen verschiedene Rollen
  - Projektplanung
  - Projektkalkulation
  - Zeit- / Terminplanung
- Projektmeilensteine durch formellen Bericht dokumentieren
- Grafische Darstellungen zur Projektplanung sind wichtig
- Hauptrisiken erkennen, analysieren + beurteilt werden
- Strategien + Notfallpläne für Projektrisiken entwickeln
- Risiken bei Projekt-Fortschrittssitzungen erörtern

Gutes Projektmanagement ist eine zwingende Voraussetzung für den Projekterfolg.

Da Software generell nicht mit herkömmlichen Gütern und Projekten vergleichbar ist, muss der Projektmanager in der Lae sein aus den auftretenden Ereignissen und Indizien die richtigen Schlüsse zu ziehen.

Software Engineering

Informatik II.

4. Software-Entwicklung

– Software Anforderungen –

Dipl.-Inform. Hartmut Petters

Software-Anforderungen

- Abstrakte + allgemeine Festlegung eines Dienstes, den ein (Software-)System liefern soll
- Klare Trennung zwischen
  - Benutzeranforderung (abstrakte Beschreibung) Lasten- / Pflichtenheft
    - Natürliche Sprache
    - Diagramme bzgl. der Abläufe
    - Randbedingungen
  - Systemanforderung (detaillierte Leistungsbeschreibung)
    - Dienste + Einschränkungen detailliert festgelegt
    - Funktionale Spezifikation (präzise)
    - Grundlage für den Vertrag
  - Spezifikation des Software-Entwurfs
    - Abstrakte Beschreibung des Software-Entwurfs
    - Detail-Spezifikation – Basis für Realisierung

Der Festlegung der Anforderungen sollte sehr genau und konsequent erfolgen, da dies die Basis ist, auf der das gesamte Projekt basiert.

Dabei sollten die Anforderungen getrennt werden nach

- Benutzeranforderungen, die die Funktionalität und die Handhabung des Systems beschreiben
- Systemanforderungen als Leistungsbeschreibung, die den systemtechnischen Rahmen wiedergibt und den leistungsgrad des Systems darstellt (Performance, Einschränkungen, etc.)
- Spezifikation des Systementwurfs, in dem die ersten Festlegungen getroffen werden bzgl. des Lösungskonzepts und der Architektur.

Definition + grundlegende Begriffe (1 of 2)

- Anforderung (Requirements)
  1. Eine *Bedingung* oder *Fähigkeit*, die von einer *Person* zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird
  2. Eine *Bedingung* oder *Fähigkeit*, die eine Software erfüllen oder besitzen muss, um einen Vertrag, eine Norm oder ein anderes, formell bestimmtes Dokument zu erfüllen (IEEE 610.12-1990)
- Anforderungsspezifikation
  - Die *Zusammenstellung aller Anforderungen* an eine Software.
  - Synonyme
    - Anforderungsdokument
    - Software Requirements Specification

Definition + grundlegende Begriffe (2 of 2)

- *„Die Spezifikation"*
  - Im Alltag nicht immer eindeutig
    - *Dokument* oder *Prozess*
    - *Anforderungs-* *Lösungs-* oder *Produkt*-Spezifikation
- Requirements Engineering (Anforderungstechnik)
  1. Das *systematische*, *disziplinierte* und *quantitativ erfassbare* Vorgehen beim Spezifizieren, d.h. Erfassen, Beschreiben und Prüfen von Anforderungen an Software
  2. *Verstehen* und *Beschreiben*, was die Kunden wünschen oder brauchen
- Pflichtenheft (verschiedene Begriffe)
  - Synonym zu Anforderungsspezifikation
  - Spezifikation + Überblick über Lösung
  - Spezifikation + Elemente der Projektabwicklung
  - ↳ „Pflichtenheft" mit Vorsicht verwenden (nicht eindeutig!)

Merkmale einer guten Spezifikation

- *Adäquatheit*
  das beschreiben, was der Kunde will
- *Vollständigkeit*
  alles beschreiben, was der Kunde will
- *Widerspruchsfreiheit*
  da sonst die spezifizierten Anforderungen nicht realisierbar sind
- *Verständlichkeit*
  da dies die Basis für die Kommunikation zwischen Kunden und Informatiker ist und Fehlinterpretationen vermeiden soll
- *Eindeutigkeit*
  damit Fehler durch Fehlinterpretationen vermieden werden
- *Prüfbarkeit*
  damit sicher festgestellt werden kann, ob die realisierten Funktionen den spezifizierten Anforderungen entsprechen

Eine gute Spezifikation sollte zielgerichtet die Anforderungen des geplanten Systems so beschreiben, dass die Spezifikation als Kommunikationsplattform zwischen den verschiedenen Parteien dient und keinerlei Unklarheiten zwischen den Diskussionspartnern aufkommen lässt.

Insgesamt dient die Spezifikation als Basis der Kommunikation zwischen den Endanwendern des Kunden, dem Management des Kunden, dem technischen Personal des Kunden sowie den entsprechenden Entscheidern auf der Auftragnehmerseite und dem eigentlichen Projektteam.

Merkmale des guten Spezifikationsprozesses

- *Kunden*-Orientierung
- *Methodisches* + *zielgerichtetes* Vorgehen
- Verwendung *geeigneter Mittel*
- *Integration* von *Erstellung* + *Prüfung* der Anforderungen

Der Spezifikationsprozess ist vom Projektmanager zu planen und zu steuern und sollte von vornherein darauf ausgelegt sein, mögliche Unklarheiten aufzuzeigen und abzuklären.

Zielgruppen der verschiedenen Festlegungen

- *Benutzeranforderungen* (kein technisches Wissen)
  - Manager des Kunden
  - Endbenutzer des Systems
  - Techniker des Kunden
  - Manager des System-Herstellers
  - Systemarchitekten
- *Systemanforderungen* (Technologie + Abläufe)
  - Endbenutzer des Systems
  - Techniker des Kunden
  - Systemarchitekten
  - Software-Entwickler
- *Spezifikation des Software-Entwurfs* (Programm-orientiert)
  - Techniker des Kunden (möglicherweise)
  - Systemarchitekten
  - Software-Entwickler

Aufgrund der komplexen Systemstruktur und der Aufgabe dies mit den unterschiedlichsten Interessensvertretern zu diskutieren, besteht die Spezifikation an sich aus verschiedenen, Benutzergruppen-spezifischen Bereichen, die die Ansprüche dieser Gruppierungen aus informationstechnischer Sicht angepasst sind. So sind die Inhalte für die Diskussion auf Management-Ebene unterschiedlich zu den Inhalten für die technische Diskussion mit den Programmierern und den technischen Vertretern des Kunden.

Funktionale + nichtfunktionale Anforderungen

- *Funktionale Anforderungen*
  - Aussagen zu den Diensten des Systems
  - Reaktionen des Systems auf spezifische Eingaben
  - Abgrenzung - Was soll das System nicht tun?
- *Nichtfunktionale Anforderungen*
  - Beschränkungen der Systemleistungen / Funktionen
  - Beschränkungen des Entwicklungsprozesses
  - Zeitbeschränkungen, einzuhaltende Standards
- *Problembereichs-Anforderungen*
  - Spezifische Anforderungen des Problembereichs
    - Funktionale Anforderungen
    - Nichtfunktionale Anforderungen

In der Spezifikation selbst sind alle Anforderungen an das System, d.h. auch nicht nur funktionale Belange festzulegen. Insbesondere ist darin herauszuarbeiten, welche Merkmale das System erfüllen und auf jeden Fall nicht erfüllen wird. Die gilt ebenso für die funktionalen Anforderungen wie für die nichtfunktionalen, systembedingten Anforderungen sowie für die Problembereichs-spezifischen Anforderungen wie Staubdichtigkeit und Einsatz unter erschwerten Einsatzbedingungen.

Benutzeranforderungen

- Funktionale + nichtfunktionale Anforderungen
- Kein detailliertes technisches Wissen
- Externe Verhalten des Systems
- Kein Systementwurf
- Beschreibung in natürlicher Sprache
- Einfache Darstellung der Abläufe (Diagramme)
- Probleme
  - Mangel an Genauigkeit
  - Verwirrung bei Anforderungen
  - Verschmelzung von Anforderungen

Generell werden in der Spezifikation die funktionalen und nicht-funktionalen Anforderungen fixiert. Die Beschreibung der Sachverhalte erfolgt ohne tieferes technisches Wissen in einer verständlichen Ausdrucksweise. Besonderer Wert wird dabei auf das externe Systemverhalten und die Benutzerschnittstelle gelegt.

Abläufe werden in einfachen grafischen Darstellungen nachvollziehbar gestaltet.

Systemanforderungen

- Genauere Beschreibung der Benutzeranforderungen
- Basis für Vertrag bzgl. System-Realisierung
- Komplette + widerspruchsfreie Spezifikation
- Ausgangsbasis für Systementwurf
- Verschiedene Systemmodelle (Klassen, Datenfluss)
- Aussage „Was" und nicht „Wie"
- Einschränkung der Interpretations-Freiräume
- Unterstützung der Spezifikation durch „strukturierte natürliche Sprache" (Pseudocode)
- Einbeziehung von Code-Fragmenten + Diagrammen
- Grafische Hilfsmittel zur Darstellung von Abläufen
- Formulare zur Beschreibung der Eingabedaten, Ausgabedaten + Funktionen / Abläufe des software-Systems

Die eigentlichen Systemanforderungen beschreiben das gesamte Systemverhalten, das durch das System zu erfüllen ist.

Notationen für Anforderungs-Spezifikation

Notation	Beschreibung
Strukturierte natürliche Sprache (Pseudocode)	Standardformulare oder Vorlagen zur Spezifikation der Anforderungen
Sprachen zur Entwurfsbeschreibung	Eingeschränkte Sprache ähnlich einer Programmier-Sprache mit abstrakten Funktionen zur Spezifikation der Anforderungen durch Definition eines Einsatz-Modells des Systems
Grafische Notationen	Grafische Sprache / Symbol-Darstellung, ergänzt durch Textvermerke wird zur Definition der funktionalen Anforderungen verwendet (z.B. SADT, Entity-Relationship-Darstellungen, ...)
Mathematische Spezifikationen	Notationen, die auf mathematischen Konzepten aufbauen (z.B. endlicher Zustandsautomat). Dadurch Reduzierung der Streitpunkte zur Systemfunktionalität. Für Ungeübte / Kunden oft schwer verständlich, deswegen Akzeptanzprobleme

Da allen Gruppierungen, die in den Spezifikationsprozess involviert sind die natürliche Sprache gemein ist, wird in den meisten Fällen diese – eventuell in einer abstrahierten Form – für den Spezifikationsprozess herangezogen.

In speziellen Bereiche kann hierfür auch eine grafische Notation oder eine algebraische Notation herangezogen werden, um Unklarheiten von vornherein auszuschließen. Voraussetzung hierfür ist, dass alle Beteiligten diese Notation und die damit ausgedrückten Sachverhalte verstehen.

Das Pflichtenheft

- Aufstellung der *Funktionalitäten*
  - Benutzeranforderungen
  - Systemanforderungen
  - Detaillierte Spezifikation
- 6 Anforderungen nach Heninger (1980)
  - Festlegung des äußeren Systemverhaltens
  - Beschränkungen bezüglich der Implementierung
  - Leicht veränderbar, anpassbar
  - Referenz für die Systemwartung
  - Vorüberlegungen zum Lebenszyklus des Systems
  - Akzeptable Reaktionen auf potenzielle Systemfehler
- Probleme bei der Erstellung
  - Beschreibung des zukünftigen Systemverhaltens
  - Einschränkungen durch bestehende Systeme
  - Kaum Beachtung von Vorüberlegungen zum Lebenszyklus

Das Pflichtenheft umfasst alle Festlegungen des Projektes.

Der Spezifikationsprozess

„Der Spezifikationsprozess" besteht aus mehreren *Teilprozessen*

- Anforderung spezifizieren
  - Gewinnen
  - Dokumentieren
  - Prüfen
- Anforderungen verwalten
  - Freigeben
  - Ändern
  - Rückverfolgen
    (wo kommt eine Anforderung her)
  - Vorwärtsverfolgung
    (wo wird eine Anforderung verwendet)

Der Spezifikationsprozess ist der Vorgang, der die Festlegung des Leistungsumfangs des Projekts sicherstellt. Auch wenn dies durch ein geeignetes Genehmigungsverfahren abgesichert ist, unterliegt das Pflichtenheft während der gesamten Projektlaufzeit Änderungen, die einerseits dokumentiert und andererseits genehmigt und freigegeben werden müssen.

Priorisierung von Anforderungen

- MUSS-Anforderungen
  - *Unverzichtbar*
- SOLL-Anforderungen
  - *Wichtig* – aber bei zu hohen Kosten verzichtbar
- WUNSCH-Anforderungen
  - „*Nice-to-Have*" – aber nicht essenziell
- Nötig bei
  - Harten *Preis-Obergrenzen*
  - *Beschaffung*
- Nützlich bei
  - *Festlegung* von Inhalt + Umfang der *Inkremente* bei inkrementeller Entwicklung
  - *Release-Planung* bei Weiterentwicklung bestehender Systeme

Nachdem alle Anforderungen festgelegt wurden, sind diese miteinander in Beziehung zu setzen und im Rahmen des Projekts mit entsprechenden Prioritäten zu versehen.

Dies bedeutet eine Einteilung der Anforderungen ind die drei Klassen:

- MUSS-Anforderung – unabdingbar
- SOLL-Anforderung – wenn irgend möglich
- WUNSCH-Anforderung – wenn hinreichend Zeit vorhanden ist

Eine Priorisierung der Anforderungen ist erforderlich, da dadurch die Planbarkeit des Projekts unterstützt wird. Insbesondere ist dies erforderlich beim Zukauf von Fremd-Software und der damit abzudeckenden Leistung, sowie bei der späteren Einbeziehung von Zusatzanforderungen, die nicht von vornherein eingeplant wurden, da diese den Projektplan empfindlich stören können. Mittels der Priorisierung ist es möglich später hinzukommende Anforderungen zu Positionieren und eventuell im Austausch gegen weniger hochpriore Anforderungen ins Projekt einzubinden.

Inhalt der Anforderungsspezifikation (1 of 2)

- Darzustellende Aspekte (unabhängig von den verwendeten Gliederungs- und Darstellungsmethoden)
- Funktionaler Aspekt
  - *Daten*
    - Struktur + Verwendung
    - Erzeugung + Speicherung
    - Übertragung + Veränderung
  - *Funktionen*
    - Eingabe
    - Verarbeitung
    - Ausgabe
  - *Verhalten*
    - Sichtbares dynamisches Systemverhalten
    - Zusammenspiel der Funktionen (untereinander + mit Daten)
  - *Fehler*
    - Normalfall
    - Fehlerfälle

In der Anforderungsspezifikation sind alle wesentlichen Aspekte zu betrachten. Dies gilt insbesondere für

- Daten und Datenstrukturen
- Funktionen und die zugrunde liegenden Abläufe
- Systemverhalten
- Fehlerverhalten

Inhalt der Anforderungsspezifikation (2 of 2)

- Leistungs-Aspekt
  - Daten*mengen* (durchschnittlich / im Extremfall)
  - Verarbeitungs- / Reaktions*geschwindigkeit* (durchschnittlich / im Extremfall)
  - Verarbeitungs*zeiten* und Verarbeitungs*intervalle*
  - Wo immer möglich – messbare Angaben!
- Qualitäts-Aspekte
  - Geforderte (*nicht-funktionale*) *Qualitäten* (z.B. Benutzerfreundlichkeit, Zuverlässigkeit)
- Randbedingungs-Aspekte
  - Einzuhaltende / zu verwendende *Schnittstellen*
  - *Normen + Gesetze*
  - *Datenschutz, Datensicherung*
  - *Explizite Vorgaben* des Auftraggebers

11.1.2004 109

Zu berücksichtigen sind auch alle

- Leistungsaspekte, dazu gehören sowohl die Datenvolumen und Mengengerüste sowie die sich daraus ableitende Verarbeitungsgeschwindigkeit
- Qualitätsaspekte aufgrund spezieller Kundenanforderungen
- Sonstige Randbedingungen und spezielle Aspekte die einzuhalten sind, wie Normen und Standards oder explizite Vorgaben.

Techniken zur Informationsgewinnung

- *Interviews*
  Vertreter des Kunden werden einzeln oder in Gruppen zu maximal drei Personen befragt
- *Fragebogen*
  Zur Erfassung von Begriffswelt + Bedürfnissen einer größeren Gruppe von Kundenvertretern
- *Gemeinsame Arbeitstagungen*
  Gemeinsame Erarbeitung der Anforderungen durch eine Gruppe ausgewählter Kundenvertreter und Informatiker (möglichst mit Moderation)

Die Informationen, die zur Erstellung der Spezifikation erforderlich sind, wird im Wesentlichen durch drei Grundmechanismen gewonnen

1. Interviews mit den Anwendern evtl. in kleinen Gruppen
2. Fragebögen, die entsprechend vorbereitet werden
3. Gemeinsame Workshops, die zur gemeinsamen Erarbeitung der Anforderungen dienen

Darstellung von Anforderungen

- Die möglichen Darstellungsformen unterscheiden sich konzeptionell vor allem in drei Aspekten:
  - Art der Darstellung
    - Konstruktiv
    - Deskriptiv
  - Formalitätsgrad der Darstellung
  - Verfügbare Gliederungs- und Abstraktionsmittel

Generell ist die Darstellungsform in drei Aspekte unterteilt

- Die Art der Darstellung – entweder konstruktiv oder deskriptiv
- Der Formalitätsgrad der Beschreibung
- Die Gliederungs- und Abstraktionsmittel

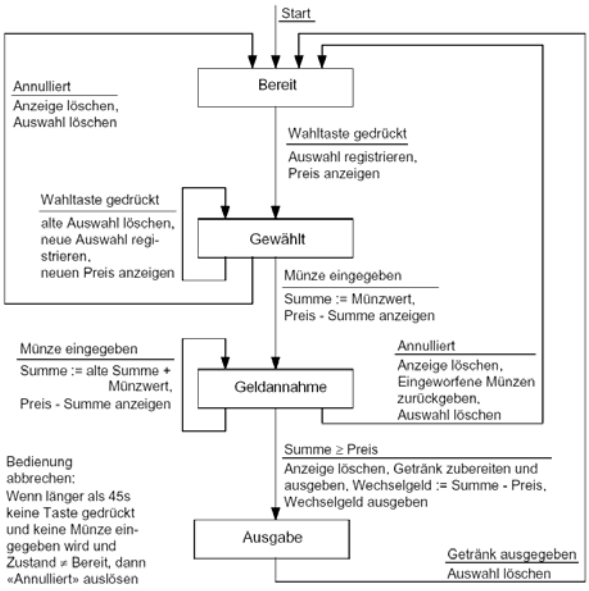
Beispiel einer informalen Spezifikation

„Der Bediener drückt die Wahltaste und bezahlt den geforderten Betrag. Sobald die Summe der eingeworfenen Münzen den geforderten Betrag übersteigt, wird das Getränk zubereitet und ausgegeben. Ferner wird das Wechselgeld berechnet und ausgegeben. Der Bedienvorgang endet, wenn das Getränk entnommen wird und wenn die Bedienung für länger als 45 Sek. Unterbrochen wird. Mit einer Annulliertaste kann der Bedienvorgang jederzeit abgebrochen werden. Nach dem Drücken einer Wahltaste kann entweder bezahlt oder eine andere Wahltaste gedrückt werden. Die zuletzt getätigte Auswahl gilt."

- Unklarheit
  Reihenfolge von „Auswahl und Bezahlung"
- Fehler
  Was ist bei Betrags-Gleichheit
- Mehrdeutigkeit
  Ist „und" oder „oder" gemeint?
- Unklarheit
  Abbruch während der Getränkeausgabe?
- Widerspruch
  Annullierung wird ausgeschlossen

Unklarheiten, Fehler, Mehrdeutigkeiten und Widersprüche sind in der Beschreibung möglichst auszuschließen.

Beispiel einer teilformalen Spezifikation



Grafische und textliche Beschreibung wird kombiniert um eine hinreichend aufschlußreiche Darstellung der Funktionalität sicherzustellen.

Beispiel einer formalen Spezifikation

Sei m_i die i-te eingegebene Münze, $|m_i|$ der Wert dieser Münze, P der geforderte Preis und G die Funktion, welche die Getränkezubereitung auslöst.

Dann muss gelten:

$$\forall\, n \in \mathbb{N}\ \forall\, m_i,\ 1 \le i \le n$$

$$\left[\sum_{i=1}^{n} |m_i| \ge P \ \wedge \ \sum_{i=1}^{n-1} |m_i| < P\right] \leftrightarrow \left[\, G(m_1 \circ m_2 \circ \ldots \circ m_n) \wedge \neg G(m_1 \circ m_2 \circ \ldots \circ m_{n-1}) \,\right]$$

Beschreibung der Funktionalität auf Basis mathematischer und algebraischer Notation, damit für die entsprechenden Fachleute diese Beschreibung eindeutig ist.

Ausgewählte Spezifikationsmethoden

- Natürlichsprachige Spezifikation
- Algebraische Spezifikation
- Datenflussorientierte Spezifikation (Strukturierte Analyse / SADT)
- Verhaltensspezifikation mit Automaten
- Objektorientierte Spezifikation
- Spezifikation mit Anwendungsfällen

Eingesetzt werden für die Spezifikation üblicherweise die hier aufgeführten Methoden.

Natürlichsprachige Spezifikation

- Nur wenig spezifische Methoden hauptsächlich zur Erkennung bzw. Vermeidung von
  - Unvollständigen Sätzen
  - Mehrdeutigkeiten
  - Vagen Aussagen
- Strukturierungshilfsmittel
  - Nummerierung der Anforderungen
  - Gliederungsschemata

Aufgrund der Interpretationsmöglichkeiten der natürlichen Sprache können hier Mehrdeutigkeiten und widersprüche relativ schnell auftreten.

Oft wird zur besseren Strukturierung ein einheitliches Schema eingesetzt.

Algebraische Spezifikation (1 of 3)

- *Deskriptive*, *formale* Methode
- Primär für die formale Spezifikation komplexer *Datentypen*
- *Syntax* durch *Signaturen* der Operationen (Definitions- und Wertebereiche)
- *Semantik* durch *Axiome* (Ausdrücke, die immer wahr sein müssen)
- Problem
  - *Sehr formale* Ausdrucksweise
  - *Nicht allgemeinverständlich*
  - Nur *für kleiner,* abgegrenzte *Aufgaben* geeignet

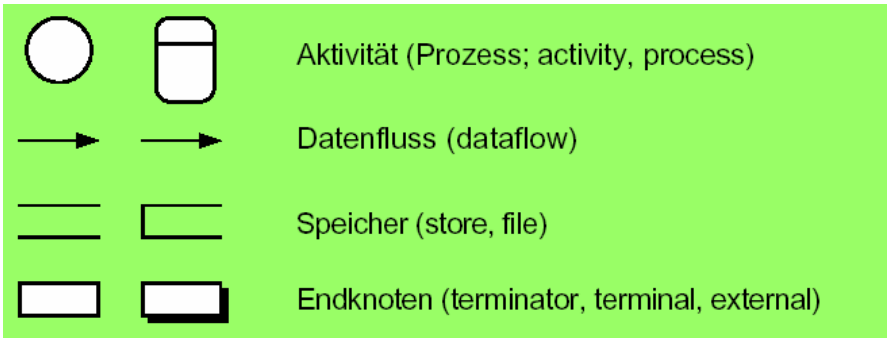
Algebraische Notation für die Spezifikation ist nur in besonderen Fällen geeignet, da die Notation für alle Beteiligten hinreichend verständlich sein muss.

Die algebraische Notation wird überwiegend zur Spezifikation formaler, komplexer Datentypen verwendet.

Strukturierte Analyse

(1 of 3)

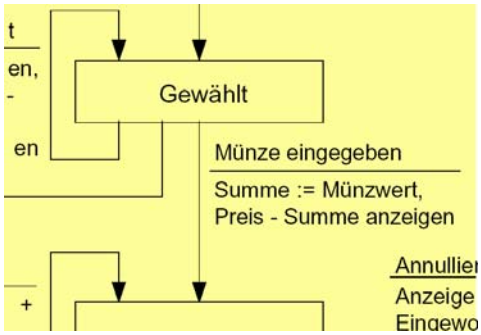
- Teilformale, konstruktive Spezifikationsmethode
- Modellierung durch Hierarchie von *Datenflussdiagrammen*
- Basiert auf dem Prinzip der *datengesteuerten Verarbeitung*



Strukturierte System Analyse und Design (SSAD) wird als teilformale und konstruktive Spezifikationsmethode zur Modellierung der Datenflüsse verwendet und basiert im Wesentlichen auf der Darstellung der Verarbeitung der Daten.

Verhaltensspezifikation mit Automaten

- Verhaltensspezifikation mit Automaten sind *teilformale, konstruktive* Modelle
- Anforderungen werden als eine Menge von *Zuständen, Zustandsübergängen, auslösenden Ereignissen und ausgelösten Aktionen* beschrieben
- Das Modell spezifiziert die Reaktionen des Systems auf eine gegebene *Folge äußerer Ereignisse*



Spezifikation mit Hilfe endlicher Automaten und der unterschiedlichen Stati des Automaten in Abhängigkeit der Eingaben und der Verarbeitungsergebnisse ist ein guten und allgemeinverständliches Hilfsmittel und wird als teilformale und konstruktive Methode oft eingesetzt.

Objektorientierte Spezifikation - Grundlagen

- Grundidee
  - Systembeschreibung durch Objekte
  - Jedes Objekt beschreibt in sich *geschlossen* Teil der *Daten*, der *Funktionalität* und des *Verhaltens* eines Systems
  - Abbildung eines *Ausschnitts der Realität* auf Objekte / Klassen
  - Objekte *kapseln* logisch zusammengehörige Daten, Operationen und Verhaltensweisen („*Information Hidding*")
  - *Gleichartige Objekte* werden als Klasse modelliert
- *Konstruktives*, *teilformales* Verfahren
- Anfänglich eine Vielzahl verschiedener Ansätze (z.B. Booch, Coad und Yourdan, Jacobson, Rumbaugh, Wirfs-Brock)
- *Industriestandard* heute – UML (Unified Modeling Language)

Da die neuen, Windows-basierten Programme üblicher Weise objektorientiert arbeiten werden diese durch eine objektorientierte Spezifikation abgebildet, in dem einerseits die Objekte und die darauf anwendbaren Methoden beschrieben werden.

Die objektorientierte Darstellung ist ebenso ein teilformale und konstruktive Beschreibungsmethode, die sich inzwischen zu einem Industriestandard entwickelt hat.

Objektorientierte Spezifikation -Objekte

- Objekte (object)
  Ein individuell erkennbares, von anderen Objekten eindeutig unterscheidbares Element der Realität
- Beispiel
  Die konkrete Person Hartmut Petters, 50 Jahre alt, Diplom-Informatiker, Manager des Geschäftsbereichs „Engineering Integration (EI)" bei ABB Informatik, verheiratet, 3 Kinder, ...
  - Objekt-Eigenschaften
    - Attribute und Werte dazu
      „Hartmut Petters, *Geschlecht: männlich*"
    - Beziehungen zu anderen Objekten
      „Hartmut Petters *leitet* den Geschäftsbereich „EI"
    - Operationen auf Objekten
      „Hartmut Petters *befördern*"

Ein Objekt im informations-technischen Sinn ist ein eindeutig beschreibbares Element, dass einerseits aus den Daten besteht, die das Objekt beinhaltet und andererseits alle Verfahren und Bearbeitungsmethoden der Daten umfasst, die die zugrunde liegenden Daten manipulieren dürfen.

Basisprinzip für diese Art der Darstellung ist die Betrachtung des Objekts als „Black Box", die Daten beinhaltet und entsprechende Funktionsaufrufe besitzt, die mit einer fest vordefinierten Funktion diese Daten bearbeitet und die entsprechenden Ergebnisse zurück liefert. Ein Zugriff auf die Daten ist nur auf diesen vordefinierten Wegen möglich.

Grundsätzlich entspricht diese Art der Handhabung dem absoluten Geheimnisprinzip.

Objektorientierte Spezifikation - Klassen

- Klassen (class)
  eine eindeutig benannte Einheit, welche eine Menge gleichartiger Objekte beschreibt
- Beschreibt den *Aufbau*, die *Bearbeitungsmöglichkeiten* und das *mögliche Verhalten* von Objekten dieser Klasse
- Beispiel
  Die Klasse *Mitarbeiter* mit den Attributen *„Name, Vorname, Geschlecht, Titel, Zivilstatus, Anzahl Kinder, ..."* und den Beziehungen *„arbeitet in"* und *„leitet"* zur Klasse *„Abteilung"* beschreibt Personen der Art *„Hartmut Petters"*
- Klassendefinition
  - Definition der Attribute der Klasse und der Wertebereiche dazu (lokale Merkmale)
  - Definition der Beziehungen zu anderen Klassen (referenzierte Merkmale)
  - Definition der Operationen (auf Objekten der Klasse / auf der Klasse)

Klassen beschreiben die abstrakte Art der Objekte in Form einer formalen Notation (Pseudo-Programmiersprache oder Programmiersprache wie C, C++)

Die reale Komponente, die sowohl in der Lage ist die Daten aufzunehmen und die entsprechenden Funktionen auszuführen sind die Objekte der Klasse.

Objektorientierte Spezifikation - Objektmodelle

- Alternative oder Ergänzung zu Klassenmodellen
- Modellierung *abstrakter Objekte*
- Stehen als Muster bzw. Repräsentanten für Objekte
- In den meisten heute verwendeten Ansätzen
  - *Nicht (kaum) verwendet* oder
  - Nur *Ergänzung* zu Klassendefinition verwendet (Modellierung des Arbeitskontextes)
- Objektmodelle anstelle von Klassenmodellen hat große *Vorteile*, wenn
  - *Verschiedene Objekte* der *gleichen Klasse* zu modellieren sind
  - Ein Modell *hierarisch* in Komponenten *zerlegt* werden soll
  - Aber
    - ↳ Modellierung von *Assoziationen* und *Vererbungsbeziehungen umständlicher*

Spezifikation mit Anwendungsfällen (1 of 2)

- Klassenmodelle modellieren nicht
  - Den *Systemkontext*
  - Die *Interaktionen* zwischen *System* und *Umgebung*
- Diese sind aber wichtig! Darum
  - ↳ *Ergänzung* durch Anwendungsfall-Modell
- Anwendungsfall (use case)
  Eine, durch genau einen Akteur angestoßene Folge von Systemereignissen, welche für den Akteur ein Ergebnis produziert und an welchem weitere Akteure teilnehmen können.
- Modellieren der einzelnen Anwendungsfälle
  - Informal durch Text, z.T. formatiert durch Schlüsselwörter
  - Teilformal, z.B. mit Zustandsautomaten

Da mit der Spezifikation mittels Klassenmodellen der Systemkontext nicht nachgebildet werden kann, werden hierzu oft Anwendungsfälle herangezogen, die genau eine spezifische Systemsituation, die damit verbundenen Ereignisse und die daraus resultierenden Ergebnisse beschreibt.

Prüfen von Anforderungen - Prinzipien

- *Abweichungen* von der geforderten Qualität der Spezifikation feststellen
- Möglichst viele *Fehler, Lücken, Unklarheiten, Mehrdeutigkeiten* etc. herausfinden und beseitigen
- Konflikte zwischen Wünschen und Forderungen verschiedener beteiligter Personen erkennen und lösen
- Verdeckte *Wünsche / Erwartungen / Befürchtungen* erkennen und thematisieren

Nach der Zusammenfassung aller Anforderungen sind diese entsprechend der zugrunde liegenden Kriterien auf Vollständigkeit, Verständlichkeit und Konsistenz zu prüfen.

Die Anforderungsspezifikation unterliegt dabei ebenso der Qualitätssicherung wie die später erzeugten Programme, da Fehler in der Spezifikation zu erheblichen Kosten und Terminproblemen führen können.

Prüfen von Anforderungen - Organisation

- Beteiligte
  - Informatiker
  - Kunde(n)
  - Beide gemeinsam

  } Je nach Verfahren

- Zeitpunkt(e)
  - *Fortlaufend*
    d.h. begleitend zur Erstellung der Spezifikation, z.B. Autor-Kritiker-Zyklus
  - *Wenn* die Spezifikation *fertig* ist (aber noch genug Zeit bleibt, die gefundenen Mängel zu beheben)

Die Anforderungen und die dadurch beschriebenen Sachverhalte werden durch das Team, durch alle Beteiligte (Anwender, Auftraggeber) gemeinsam vorgenommen. Diese Überprüfung erfolgt mitlaufend und permanent, da auch während der Umsetzungsphase weitere Änderungen und Ergänzungen der Spezifikation notwendig sein werden.

Prüfen von Anforderungen - Prüfverfahren

- Reviews
- Prüf- und Analyse-Hilfsmittel in Werkzeugen
- Simulation / Animation
- Prototypen

- Review
  - Das Mittel der Wahl zur Prüfung von Spezifikationen
  - *Walk-Through*
    Autor(en) führt durch den Review-Prozess
  - *Inspektion*
    Gutachter prüfen die Spezifikation eigenständig,
    tragen in Sitzungen die Befunde zusammen
  - *Autor-Kritiker-Zyklus*
    Kunde liest und kritisiert die Dokumente der Spezifikation
    bespricht die Ergebnisse der Durchsicht mit den Autoren

Prüfverfahren für diesen Bereich sind einerseits Reviews sowie sonstige Prüf- und Analyse-Hilfsmittel, Simulation und Animation bis hin zur Erstellung einfacher Prototypen, die das Systemverhalten und die Benutzeroberfläche darstellen sollen.

Der Review-Prozess erfolgt entweder in Form eines „Walk-through“-Prozesses, in dem der Autor die Beteiligten durch den Review-Prozess begleitet und die einzelnen Passagen erläutert oder in Form einer Inspektion, bei der ein Gutachter die Inhalte überprüft oder alternativ in einem Autoren-Kritiker-Zyklus, in dem der Kunde bzw. die Anwender die Spezifikation lesen und kommentieren bzw. das Verständnis darstellen.

Prüfen von Anforderungen - Prüfverfahren

- Prototyp
  - Beurteilung der Adäquatheit / praktischen Gebrauchstauglichkeit des spezifizierten Systems anhand der, im *Prototyp* realisierten Systemteile
  - *Erprobung* eines Systems in der geplanten Einsatzumgebung
  - Modell für die weitere Entwicklung oder für das zu schaffende Produkt
  - Mächtigstes (aber auch aufwendigstes) Mittel zur Beurteilung der *Adäquatheit*

Zur Prüfung der Anforderung und der Adäquatheit der Beschreibung können vor allem Prototypen gute Dienste leisten, da durch sie auf einfache Art und Weise die Grundfunktionalität und die Benutzerschicht dargestellt werden kann und einen Eindruck des zukünftigen Systems vermitteln kann.

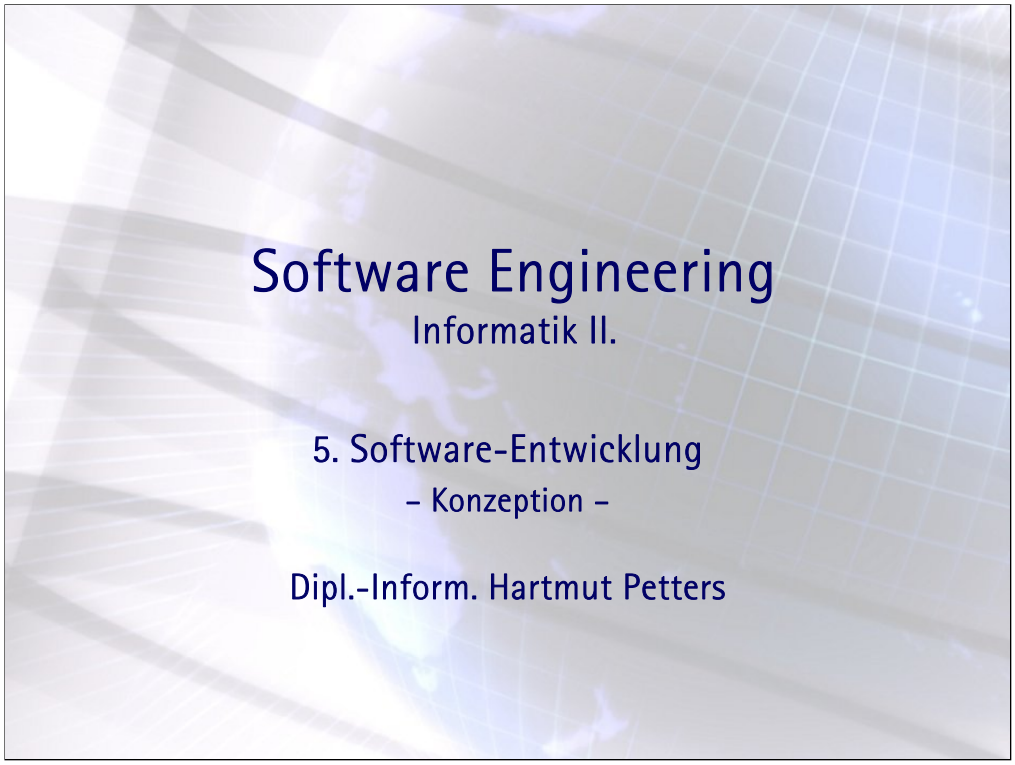
Zusammenfassung – 1te

- **Merkmale einer guten Anforderungsspezifikation**
  - Adäquat + Verständlich
  - Vollständig + Widerspruchsfrei
  - Eindeutig + Überprüfbar
- **Merkmale des Spezifikations-Prozesses**
  - Kundenorientiert + Einsatz geeigneter Mittel
  - Methodisch + zielgerichtet
  - Integration von Erstellung + Prüfung
- **Techniken zur Erstellung der Anforderungen**
  - Interviews
  - Fragebogen
  - Gemeinsame Arbeitssitzungen
- **Lösungsansätze**
  - Deskriptive Lösung
  - Konstruktive Lösung

Die Anforderungsspezifikation ist die Basis für die spätere Implementierung und sollte sehr sorgfältig vorgenommen werden.

Zusammenfassung – 2te

- **Spezifikationsmethoden**
  - Natürlichsprachige Spezifikation
  - Algebraische Spezifikation
  - Datenflussorientierte Spezifikation (Strukturierte Analyse / SADT)
  - Verhaltensspezifikation mit Automaten
  - Objektorientierte Spezifikation
  - Spezifikation mit Anwendungsfällen
- **Prüfen der Anforderungen**
  - Organisation der Anforderungsprüfung
    - Prüfung durch Auftraggeber und Auftragnehmer
    - Kontinuierlich und/oder am Ende
  - Durchführung der Anforderungsprüfung / Prüfverfahren
    - Reviews
    - Prüf- + Analyse-Hilfsmittel in Werkzeugen
    - Simulation / Animation
    - Prototypen



Das Kapitel Konzeption + Aufwandsabschätzung beschäftigt sich mit dem Entwurf und der Konkretisierung der Umsetzung der Anforderungen in eine Lösung.

Die Konzeption selbst erfolgt in einem „Top-Down“-Entwurfsverfahren, in dem der Lösungsansatz zunehmend verfeinert und detaillierter ausgearbeitet wird.

Das Kapitel gibt dabei Hinweise, wie dieser Prozess ablaufen kann und welche Randbedingungen dabei zu beachten sind.

Definition - Begriffsbestimmung

- Konzeption einer Lösung (architectural design)
  Erstellung und *Dokumentation* der *Architektur* oder des Grobentwurfs eines Systems
- Dabei werden die wesentlichen Komponenten der Lösung und die Interaktionen zwischen diesen Komponenten festgelegt.
- Architektur (architecture)
  Die *Organisationsstruktur* eines Systems oder einer Komponente
- Entwurf (design)
  1. Der *Prozess des Definierens* von *Architektur, Komponenten, Schnittstellen* und anderen Charakteristiken eines Systems oder einer Komponente
  2. Das *Ergebnis des Entwurfs-Prozesses* (s. 1.)
- Lösungskonzeption (software architecture, system architecture)
  Das *Dokument*, welches das Konzept der Lösung, d.h. die Architektur der zu erstellenden Software dokumentiert

Die Konzeption einer Lösung beschreibt die wesentlichen Elemente und das Zusammenspiel dieser als Gesamtsystem.

Entwurfsprinzipien – Modularität

- Modularisierung
  ist eine Hauptaufgabe bei der Konzeption
- Modul (module)
  Eine *benannte*, klar *abgrenzbare Komponente* eines Systems
- Gute Module haben folgende Eigenschaften
  - In sich *geschlossene* Einheit
  - *Ohne Kenntnisse* über den inneren Aufbau *verwendbar*
  - *Kommunikation* mit der Umgebung erfolgt ausschließlich über klar *definierte Schnittstellen*
  - Im „Inneren" *rückwirkungsfrei änderbar*
  - *Korrektheit* ohne Kenntnis der Einbettung in das Gesamtsystem *überprüfbar*

Eines der wichtigsten Prinzipien und die Hauptaufgabe bei der Konzeption ist die Modularisierung der Anforderungen und die Zusammenfassung logisch zusammenhängender Elemente zu einem Informationsobjekt.

Aus dieser Modularisierung werden die Hauptkomponenen und die damit zusammenhängenden Funktionen abgeleitet.

Messung der Güte der Modularisierung

- Zwei charakteristische Maße
  - *Kohäsion*
  - *Kopplung*
- Kohäsion (cohesion)
  Ein Maß für die *Stärke des inneren Zusammenhangs* eines Moduls
  - Je höher die Kohäsion, desto besser die Modularisierung
    - *Schlecht* - zufällig, zeitlich
    - *Gut* - funktional, objektbezogen
- Kopplung (coupling)
  Ein Maß für die *Abhängigkeit zwischen zwei Modulen*
  - Je geringer die wechselseitige Kopplung zwischen den Modulen ist, desto besser ist die Modularisierung
    - *Schlecht* - Inhaltskopplung, globale Kopplung
    - *Akzeptabel* - Datenbereichskopplung
    - *Gut* - Datenkopplung

Die Güte der Modularisierung wird an zwei Charakteristikas festgemacht,

- Der Kohäsion, d.h. dem inneren Zusammenhalt der Komponenten
- Der Kopplung, d.h. der Abhängigkeit einzelner Module von anderen.

Die Basis für eine gute Modularisierung ist dann gegeben, wenn der inner Zusammenhalt der Module sehr hoch und die Abhängigkeit von Fremdmodulen so gering wie möglich ist.

Entwurfsprinzipien – Geheimnisprinzip

- Geheimnisprinzip (information hidding)
  Kriterium zur *Gliederung* eines Systems in *Komponenten* (Daten, Abläufe + Algorithmen), so dass
  - Jede Komponente eine *Leistung* (oder eine Gruppe logischer, eng zusammenhängender Leistungen) *vollständig bereitstellt / erbringt*
  - außerhalb der Komponente nur bekannt ist, *was* die Komponente leistet
  - Nach außen *verborgen* bleibt, *wie* sie diese Leistung erbringt

  [Parnas 1972]

  - Fundamentales Prinzip zur *Beherrschung* komplexer Systeme
  - Auch im *täglichen Leben* fortwährend benutzt
  - Liefert *gute Modularisierungen*
  - *Stellt* die *Überprüfbarkeit* der Komponenten *sicher*

Neben der Modularisierung ist das oberste Entwurfsprinzip das Geheimnisprinzip (Information Hidding), das sicherstellt, dass von Außen keinerlei Einfluss auf die Daten genommen werden kann.

Hierbei werden alle Informationen, die ein Objekt betreffen nach außen verschleiert und versteckt, so dass ein Zugriff nur über die vorgegebenen Mechanismen möglich ist.

Nur so ist sichergestellt, dass die einzelnen Module als autonome Einheit betrachtet werden und deren Fehlerfreiheit auch unabhängig von anderen Einflüssen getestet werden kann.

Das Lösungskonzept – 1te

- Dokumentation des Ergebnisses des Architektur-Entwurfs
- Mögliche Gliederung

1. *Einleitung*
   (1) Überblick
   (2) Ziele + Vorgaben
   (3) Einbettung + Abgrenzung
   (4) Lösungsalternativen
2. *Struktur der Lösung*
   (1) Übersicht
   (2) Prozess-Struktur
   (3) Modulare Struktur
   (4) Entwurf der Module
   (5) Physische Struktur

Das Lösungskonzept und die darunter liegende Architektur des Systems unterliegt ebenfalls der Dokumentationspflicht.

In der Einleitung ist ein grundlegender Überblick, die Ziele und die Vorgehensweise sowie die Einbettung und Abgrenzung zusammen mit möglichen Lösungsalternativen darzustellen.

Bei der Darstellung der Lösungsstruktur sollen neben der Übersicht auch die Prozess-Struktur, die Modul-Struktur, der Modul-Entwurf sowie die Physische Systemstruktur dargestellt werden.

Das Lösungskonzept – 2te

3. *Aspektbezogene Teilkonzepte*
   Ein Unterkapitel je interessierendem Aspekt, z.B.
   - Datenhaltungskonzept
   - Mensch-Maschine-Kommunikation
   - Fehlerbehandlungskonzept
   - Fehlertoleranz-Konzept
   - Sicherheitskonzept
   - ...
4. *Voraussetzungen + benötigte Hilfsmittel / Abgrenzungen*
   (1) Benötigte Software
   (2) Benötigte Hardware
   (3) Benötigtes Umfeld
   (4) Was gehört nicht dazu / Was wird nicht gemacht
5. *Quellennachweis*

Die aspektbezogenen Teilkonzepte, zu denen neben dem Datenhaltungskonzept, der Benutzerschnittstelle und der Fehlerbehandlung auch das Sicherheits- und Backup-Konzept gehören, sollten in diesem Abschnitt näher erläutert werden.

Im letzten Abschnitt sind alle Voraussetzungen und sonstigen Randbedingungen sowie die benötigten Hardware- und Software-Komponenten neben dem direkten Umfeld und dem Quellennachweis beschrieben werden.

Der Entwurfprozess

- Erster Schritt der Lösung des Problems
- *Anforderungsspezifikation* als *Vorgabe* notwendig
- Zeitliche + hierarchische *Verzahnung* von Anforderungsspezifikation + Architekturentwurf möglich
- Ergebnisse immer in *separaten Dokumenten*
- Architekturentwurf
  Komponenten, Schnittstellen, Interaktionen, ...
- Detail-Entwurf
  Algorithmen + interne Datenstrukturen

Der Entwurfsprozess stellt den ersten Schritt der realen Lösung dar.

Im Architektur- und Detailentwurfwerden die einzelnen Komponenten, die Schnittstellen sowie die zugrunde liegenden Algorithmen und Datenstrukturen zusammengefasst.

Lösungsvarianten

- Ganzen Lösungsraum betrachten
- Lösungs*varianten*
  - *Erkennen*
  - *Verfolgen*
  - *Abwägen*
    - Was kostet es, das Optimum zu verfehlen?
    - Was kostet die Untersuchung (Geld, Zeit, Ressourcen)?
  - *Entscheiden*
  - *Dokumentieren*
- *Kosten*
  - Nicht nur Entwicklungskosten der Varianten betrachten!
  - Auch Betriebskosten, Pflegekosten, Folgekosten (auch anderswo)
- Beschaffungsvarianten generell immer betrachten!

Zur Vollständigkeit und zur Überprüfung des ausgewählten Lösungs-Konzepts ist es erforderlich auch die möglichen Alternativkonzepte kurz darzustellen und gegeneinander abzuwägen.

Neben der eigenen Umsetzung sind die Möglichkeiten einer Beschaffung und Anpassung existierender Lösungen in den Mittelpunkt zu stellen und dediziert zu betrachten.

Beschaffung + Wiederverwendung

1. *Anforderungen* analysieren
2. *Hauptkriterien* definieren („KO"-Kriterien)
3. *Marktübersicht* verschaffen, *Grobauswahl* treffen
4. *System-Kandidaten evaluieren*
5. *Entscheidungsgrundlage* erarbeiten
6. *Entscheidungen forcieren* + dokumentieren

- Zu treibender Aufwand hängt ab von
  - Wichtigkeit / Priorität
  - Preis
  - Nutzungsdauer

Beschaffung + Wiederverwendung sind die wirtschaftlichsten Methoden für die Umsetzung, da sowohl die Fehlerbeseitigung wie auch die Dokumentation und Pflege hinreichend abgedeckt sein sollten.

Architekturstil

- Architekturstil (architectural style)
  Leitlinie für die *Gestaltung* einer Architektur
  - Verwendete *Modularten*
  - Verwendete *Arten von Kooperation* zwischen den Modulen
  - Benutzung passender *Architekturmuster*
  - Orientierung an einer *Architekturmetapher*

  } *Architekturstil*

- Im Folgenden
  - *Vorstellung ausgewählter Stilarten*

Die Systemarchitektur ist die Basis für das Konzept und stellt die Leitlinie der Gestaltung des Systems dar.

Funktionsorientierte Architektur

- Funktionsorientierte Architektur (structured design)
  - Jedes Modul führt eine *Funktion* aus
  - Zur Realisierung einer Funktion können andere, einfachere Funktionen aufgerufen werden
    - ↳ Entwurf
      = *Hierarchie aufeinander aufbauender Funktionen*
  - Häufig gegliedert nach
    - Eingabe
    - Verarbeitung
    - Ausgabe
  - *Ungenügend Abstraktionsmöglichkeiten*
  - Liefert in vielen Fällen *keine gute Modularisierung*

Die funktionsorientierte Architektur stellt die Funktionen des Systems in den Mittelpunkt und stellt sicher, dass jede Funktion durch ein entsprechendes Modul realisiert wird.

Datenorientierte Architektur

- **Datenorientierte Architektur (Entwurf mit Datenabstraktion)**
  - Modularisierung nach dem **Geheimnisprinzip**
  - Datenstrukturen und alle darauf möglichen Operationen sind zusammengefasst: *Datenabstraktion*
    - Notwendig zum Verbergen von Entwurfsentscheidungen
    - Realisierung durch Abstrakte Datentypen
  - System besteht aus einer *Menge aufeinander aufbauender Datenabstraktionen*
    - Jedes Modul *bietet Leistungen an*
    - Module *benutzen die Leistungen* anderer Module zur Realisierung der eigenen Leistungen
      - ↳ *Benutzungshierarchie*
    - Zusammenarbeit durch **Verträge** (Design by Contracts)
  - Zusätzlich häufig Gliederung in *Schichten*

Bei der datenorientierten Architektur sind die Datenstrukturen das Gliederungsmittel, nach dem sich die gesamte Struktur des Systems ausrichtet.

Objektorientierte Architektur 1te

- System besteht aus einer Menge *kooperierender Objekte*
- Module repräsentieren *Objekte des Problembereichs* oder benötigtes *Informations-Element*
- Abstrakte Beschreibung gleichartiger Objekte → Klasse
- Geeignet gebildete Klassen beachten das Geheimnisprinzip
  - *Gute Modularisierung*
- *Systematischer Zusammenhang* zwischen allgemeinen + speziellen Objekten
  - Objekte von Spezialklassen erben alle Strukturen + Operationen der übergeordneten, allgemeinen Klassen
    - Spezialisierungs- (Generalisierungs-) Hierarchie
  - Ermöglicht *problemnahe* Modellierung
  - Analog der *Begriffshierarchie* im menschlichen Denken
  - Schränkt jedoch die Anwendbarkeit des Geheimnisprinzips ein

Die objektorientierte Architektur baut auf der datenorientierten Struktur auf und stellt das System als eine Menge kooperierender Objekte dar, wobei gleichartige Objekte in Klassen zusammengefasst sind.

Objektorientierte Architektur – 2te

- Zwei Kooperationsmechanismen
  - *Benutzung*
  - *Vererbung*
  - ↳ Objektorientierter Entwurf ist anspruchsvoll
- Bei richtiger Anwendung
  - ↳ *Qualitativ hochwertige Modelle*
- Bei falscher Anwendung
  - ↳ Der Alptraum jedes Wartungs-Programmierers

Die zwei grundlegenden Kooperationsmechanismen zwischen den Objekten sind die

- Benutzung – ein Objekt benutzt ein anderes
- Vererbung – ein Objekt stammt von einem anderen ab und übernimmt von diesem einige Basiseigenschaften.

Ein objektorientierter Entwurf kann bei richtiger Anwendung ein qualitativ hochwertiges Systemmodell darstellen.

Prozessorientierte Architektur

- System besteht aus einer Menge *unabhängig arbeitender*, untereinander *kooperierender* (systeminterner) *Akteure*
- Akteure sind typisch als Prozesse realisiert
- Prozesse *kooperieren* durch *Austausch von Nachrichten* oder durch Zugriff auf *gemeinsame Speicherbereiche*
- Prozesse sind die Module der obersten Stufe
- Jeder Prozess ist typisch ein sequentiell ablaufender Systemteil
- Prozesse sind selbst wieder modularisiert, z.B. in objektorientiertem Stil
- Entwurfsaufgaben
  - Prozess-Struktur
  - Kommunikationsarchitektur (oft zugekauft)
  - Interne Architekturen der Prozesse

Bei der prozessorientierten Architektur stehen die Abläufe des Systems im Vordergrund und bestimmen die Strukturierung.

Komponentenorientierte Architektur

- Komponentenorientierte Architektur (verteilte Objekte)
  Komponente (im engeren Sinn)
  Stark gekapselte Menge zusammenhöriger Objekte / Klassen, die eine *gemeinsame Aufgabe lösen*
- System besteht aus einer *Menge von* (möglicherweise geografisch verteilter) *Komponenten*, die über *Makler / Agenten kommunizieren*
- Komponenten
  kennen
  - *Schnittstellen* ihrer Partnerkomponenten

  ... kennen nicht
  - *Art der Realisierung* der Kommunikation
  - Geografische *Lokalisierung*
  - *Implementierung* der Partnerkomponenten
- Typische Vertreter
  - Client-Server-Architektur
  - Middleware-Architektur

Die komponentenorientierte Architektur basiert auf dem objektorientierten Ansatz, bei dem diese Objekte durch definierte Kommunikationsagenten miteinander kommunizieren.

Zusammenfassung - Konzeption

- Anforderungsspezifikation als Vorgabe erforderlich
- Systematischer Entwurf + strukturierter Aufbau wichtig
- Lösungskonzept zwingend erforderlich
- Lösungsvarianten eruieren + betrachten
- Einheitliche Architektur vorgeben
- Architekturfehler sind kostspielig
- Gliederung in Komponenten + Interaktionen
- Hauptaufgaben
  - Modularisierung
  - Geheimnisprinzip einsetzen
  - Schnittstellen festlegen
  - Wiederverwendung praktizieren

Eine klare und konsistente Anforderungsspezifikation ist die Basis für eine gutes Systemkonzept und ist für die durchgängige Strujtur des Systems unabdingbar.