Software Engineering

Informatik II.

Zusammenfassung

– Kapitel 6. – 11. –

Dipl.-Inform. Hartmut Petters

Vorwort – was ich noch zu sagen hätte ...

Basis dieser Vorlesung sind vor allem die folgenden Ausarbeitungen

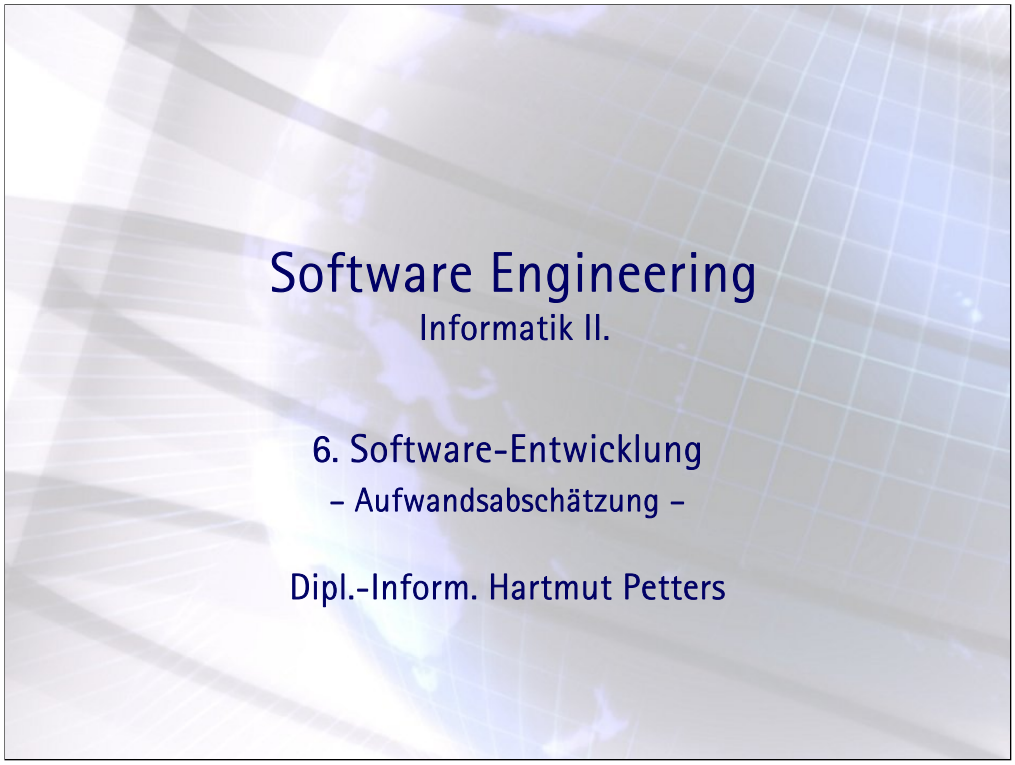
- Vorlesungsskript „Software Engineering"
  von Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/se_I/
- Vorlesungsskript „Informatik II – Software Engineering"
  von Frau Prof. Dr. Kühn FH Karlsruhe FB W
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Das 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.

Für die Möglichkeiten der Übernahme wesentlicher Teile des Vorlesungsskriptes von den genannten Autoren möchte ich mich an dieser Stelle nachmals ausdrücklich bedanken, da es mir ansonsten unmöglich gewesen wäre die Vorlesung in der mir zur Verfügung stehenden kurzen Zeit in dieser Qualität auszuarbeiten und entsprechende Unterlagen zu erstellen.



Das Kapitel 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.

Dabei ist eine der schwierigsten Aufgaben eine sinnvolle und realistische Abschätzung des Aufwands zu machen, da die Umsetzung der Lösungskonzepte in Programm-Code einerseits ein sehr kreativer Prozess ist, der so gut wie nicht steuerbar ist und andererseits die Umsetzung , Integration und Fehlersuche nur wenigen festen Regeln unterliegt.

Eine sinnvolle Aufwandsabschätzung ist deswegen auf die Statistik „großer Zahlen" angewiesen, d.h., dass die Aufwandsabschätzung, wenn sie seriös gemacht werden soll über möglichst viele kleine Teilelemente gemacht werden muss, damit die Statistik greift und sich die Fehleinschätzungen, die sowohl zu Gunsten wie zu Ungunsten passieren sich gegenseitig kompensieren. Je größer die Anzahl der Teilmodule ist, die abgeschätzt und für die Realisierung notwendig sind, umso wahrscheinlicher ist es, dass in der Gesamtheit die Abschätzung realistisch ist.

Aufwandsabschätzung

- Voraussetzung zur Aufwandsabschätzung
  - Genaue Kenntnis der Anforderungen
    - ↳ *Was soll gemacht werden?*
  - Klare Abgrenzung der Funktionalitäten
    - ↳ *Was wird nicht gemacht + was gehört nicht dazu?*
  - Genaue Kenntnis der Anwendungsumgebung
    - ↳ *Welche Integrationen sind erforderlich?*
  - Zusammenfassung der konzeptionellen Ideen
    - ↳ *Wie könnte das Zielsystem aussehen?*
  - Dokumentation der zugrundeliegenden Architektur
    - ↳ *Wie sieht die Architektur aus + warum wurde diese ausgewählt – Pro´s + Con´s?*

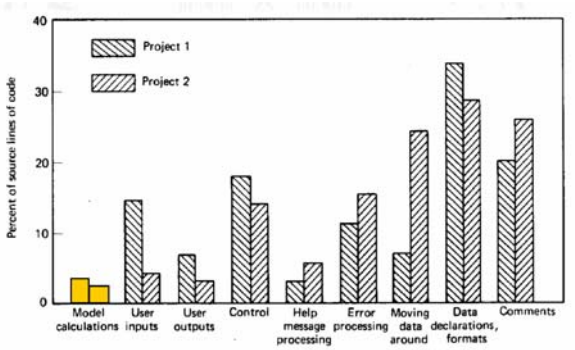
18.1.2004 4

Was benötigen wir für eine realistische Abschätzung des Aufwands?

1. Eine möglichste genaue Darstellung der Anforderungen, so dass die Umsetzung keine Überraschungen bringt.
2. Eine ganz klare Abgrenzung der Funktionalitäten und dabei insbesondere der Dinge, die „nicht" zur Realisierung gehören, dh., von vornherein klar festlegen, was zum Lieferumfang gehört und was nicht! Dies ist deswegen auch wichtig, da es oft viel leichter ist zu sagen was nicht dazu gehört, als was wirklich alles realisiert werden soll.
3. Klare Kenntnisse der Anwendungsumgebung und die Integrationsanforderungen an die Lösung sind ebenso unerlässlich, da gerade Integrationen in Fremdanwendungen (z.B. SAP R/3) sehr aufwendig werden können.
4. Zusammenfassung der Konzeption und Architektur, als Basis für die gesamte Realisierung und Umsetzungs-Richtlinie
5. Dokumentation der einzelnen Schritte und gewonnenen Erkenntnisse – insbesondere auch der Entscheidungskriterien, die zur Festlegung der Architektur und der Umsetzungsstrategie geführt haben (Für + Wider) auch im Vergleich mit den möglichen Alternativen.

Probleme bei der Aufwandsabschätzung

- Kreativität ist nicht kontrollierbar
- Software-Entwicklung ist Kopfarbeit
- Kernfunktionen werden mit dem Produkt verwechselt
- Erfahrungen aus Kleinprojekten werden linear extrapoliert
- Programmierer programmieren nicht zu 100%



Die Umsetzung der Anforderungen in eine Lösung ist ein hochgradiger kreativer Prozess, der durch die Entwickler und System-Designer zu leisten ist. Dieser Prozess kann weder befohlen noch klar gesteuert werden. Die einzige Möglichkeit sinnvoll damit umzugehen ist es, optimale Voraussetzungen für diese Arbeit zu schaffen, in dem diese Phase durch systematische Unterstützung und Gruppenarbeiten einen definierten Rahmen erhalten.

Da Konzeption, Aufwandsabschätzung und Realisierung durch die Kopfarbeit der Entwickler geleistet werden muss, kommt den Rahmenbedingungen eine hohe Bedeutung zu.

Insbesondere Kommunikation und Erfahrungsaustausch sind dabei nicht zu unterschätzen.

Aufwandsabschätzung

- **Projektmanager müssen die Fragen klären**
  - Wieviel Aufwand ist für die Erledigung einer definierten Aufgabe erforderlich?
  - Wieviel Zeit ist für die Erledigung einer definierten Aufgabe erforderlich?
  - Wie hoch sind die anfallenden Gesamtkosten für eine definierte Aufgabe
- **Projektaufwandsschätzung + Zeitplanung erfolgen gemeinsam im Team**
- **Kontinuierliche Aktualisierung der Aufwands-Abschätzungen, sobald mehr Infos vorliegen**

Aufwandsabschätzung bedeutet die Klärung der Fragen:

1. Wie viel Zeit wird für die einzelnen Teilaufgaben, die Zusammenführung der einzelnen Komponenten zu einem (Teil-) System sowie für die Gesamtintegration benötigt.
2. Wie viel Zeit und damit auch der finanzielle Rahmen ist erforderlich
3. Welche Beistellungen und Unterstützung ist notwendig

Die Aufgabe der Aufwandsabschätzung kann umso effizienter und sinnvoller erfolgen, je besser das gesamte Team in diesen Prozess integriert ist und zusammenarbeitet. Die unterschiedlichen Erfahrungen der Teammitglieder ermöglichen es dadurch kritische Bereiche sinnvoll abzuschätzen.

Empirische Schätzverfahren

- Analogieschlüsse basierend auf *Erfahrungen* der Schätzer
- *Experten-Beurteilung*
  - Brauchbar, wenn Erfahrungen mit gleichartigen Projekten vorhanden sind
  - Vorteil: Einfach + billig
  - Nachteil: Krasse Fehler sind möglich
- *Delphi-Methode* (wie Experten-Beurteilung nur iterativ)
  - Schätzung durch mehrere unabhängige Experten
  - Mehrere Runden zur Schätzung
  - Vorteil: Konvergiert (hoffentlich) – eliminiert Ausreißer
  - Nachteil: sehr aufwendig

Generell muss davon ausgegangen werden, dass aufgrund der sehr stark kreativen Komponente der Lösungsfindung und Umsetzung die Aufwandsabschätzung immer mit einem hohen Grad an Unsicherheit belegt ist. Insbesondere auch, da bei der Festlegung der Anforderungen es nicht möglich ist dies absolut zu 100% zu tun und damit immer Interpretationsspielräume vorhanden sind. Diese Unsicherheit kann nur durch die persönlichen Erfahrungen der Schätzer / Experten einigermaßen ausgeglichen werden.

Die üblichen Schätzverfahren erfolgen deswegen auch i.d.R. als Experten-Beurteilung, d.h., Experten machen aufgrund der vorhandenen Kenntnisse Aufwandsabschätzungen, die in der Diskussion verifiziert werden.

Um diese Ergebnisse noch zu verbessern wird im Rahmen der Delphi-Methode dieser Prozess iterativ durchgeführt und die entsprechenden Beurteilungen von den Experten begründet und diskutiert.

Je mehr Unteraufgaben dabei geschätzt werden, umso wahrscheinlicher wird damit auch das Endergebnis, da sich durch die Wahrscheinlichkeit die Fehleinschätzungen i.d.R. – wenigstens teilweise - kompensieren.

Die Delphi-Methode ist aufgrund der gemeinsam erarbeiteten Ergebnisse eine häufig angewandte Methode und liefert bei entsprechend erfahrenen Experten auch gute Ergebnisse..

Algorithmische Schätzverfahren

- Berechnung von *Kosten*- und *Durchlaufzeit-Funktionen*
- *Eingangsvariablen* müssen zutreffend geschätzt werden
- Modell muss *„kalibriert"* werden
- Liefert bei richtiger Kalibrierung die *besten Prognosen*
- Ohne *Maßzahlen* über *abgewickelte Projekte* keine zuverlässigen *Prognosen*!
- Zwei bekannte *Verfahren*
  - *COCOMO*
  - *Function Point*

18.1.2004 8

Um die Unsicherheit der Expertenschätzungen zu eliminieren wurden in der Vergangenheit auch algorithmische Schätzverfahren definiert und in speziellen Situationen auch erfolgreich eingesetzt.

Diese algorithmischen Verfahren basieren dabei aber auf einer Festlegung der Rahmenbedingungen, die dabei üblicherweise geschätzt werden müssen. Damit wird die Heuristik der Schätzung nicht eliminiert, sondern nur in andere Bereiche verlagert. Solange keine verlässlichen Maßzahlen aus bereits abgewickelten Projekten für diese Kalibrierung vorhanden sind, solange ist damit keine zuverlässige Prognose möglich.

Die zwei bekanntesten Verfahren sind dabei

1. COCOMO
2. Function Point

Algorithmische Verfahren

- Stark abhängig von
  - Eingangsgrößen der Kostenfunktion
  - Modell muß kalibriert werden
    - ↳ Dann werden die besten Ergebnisse geliefert
- Probleme
  - Nur einsetzbar nach entsprechender Vorarbeit (Kalibrierung)
  - Stark abhängig von der Genauigkeit der Eingangsgrößen

Algorithmische Verfahren sind stark abhängig von den definierten Eingangsgrößen und der Kalibrierung auf der Basis bestehender Erfahrungen.

Die Vorarbeiten für die Kalibrierung dieser Methoden können dabei sehr aufwendig sein.

COCOMO (Constructive Cost Model)

- Bekanntes algorithmisches Schätzverfahren
- Geht von der Schätzung der Produktgröße aus
- Grundgleichungen:

$$MM = 2{,}4\ KDSI^{1{,}05}$$
$$TDEV = 2{,}5\ MM^{0{,}38}$$

MM:	Man Month (Personen-Monate)
KDSI:	Kilo delivered source instructions (Anzahl der ausgelieferten Codezeilen in 1.000)
TDEV:	Time to Develop (Entwicklungszeit)

- Gilt für *einfache Anwendungs-Software* (organic mode) ...
- ... in einer *stabilen Umgebung*

18.1.2004 10 © by Hartmut Petters

Das Verfahren COCOMO wurde von Boehm (1981) entwickelt und geht von der Schätzung der Produktgröße in „Kilo Lines of Delivered Source Instructions (KDSI)" aus. Aus diesem Grundwert und einer Reihe von Kosten-Multiplikatoren werden die Kosten und die Durchlaufzeit nach der oben aufgeführten Grundgleichung berechnet.

Abhängig von der Komplexität der Lösung wird die Grundgleichung mit unterschiedlichen Faktoren versehen.

COCOMO-Grundgleichung für Programmsysteme

$$MM = 3{,}0\ KDSI^{1{,}12}$$
$$TDEV = 2{,}5\ MM^{0{,}35}$$

COCOMO-Grundgleichung für eingebettete Systeme

$$MM = 3{,}6\ KDSI^{1{,}2}$$
$$TDEV = 2{,}5\ MM^{0{,}32}$$

Die Abschätzung kann noch dadurch verbessert werden, indem der Basisfaktor MM mit dem Produkt einer Reihe von Kostenfaktoren abhängig von der Art der Anwendung und deren Komplexität multipliziert wird.

$$MM_{korr} = \text{Produkt der Kostenfaktoren} \times MM_{nominal}$$

Eine ausführliche Beschreibung dieser Kostenfaktoren findet sich in der Orginalliteratur bei Boehm.

COCOMO - Berechnungsgrundlagen

- Randbedingungen
  - Schätzungen schließen den Aufwand für die Anforderungsdefinition nicht mit ein
  - Gleichungen müssen *unternehmensspezifisch kalibriert* werden

- Grundgleichungen für andere Software
  - *Programmsysteme* (semi-detached mode)
    $MM = 3{,}0\ KDSI^{1{,}12}\ TDEV = 2{,}5\ MM^{0{,}35}$
  - *Eingebettete Systeme* (embedded mode)
    $MM = 3{,}6\ KDSI^{1{,}2}\ TDEV = 2{,}5\ MM^{0{,}32}$

Das „Function-Point-Verfahren"

- *Relatives Maß* zur Bewertung der Funktionalität
- Von A. Albrecht 1979 bei IBM entwickelt
- Falls *Erfahrungszahlen* (Kosten pro „Function Point") vorliegen
  - Kostenschätzung möglich
- Anwendung primär für *Informationssysteme*
- Idee
  - In geeigneter Weise zählen der
    - *Dateneingaben* (External Input)
    - *Datenausgaben* (External Output)
    - *Anfragen* (External Inquiry)
    - *Schnittstellen* zu *externen Datenbeständen* (External Interface File)
    - *Interne Datenbestände* (Logical Internal File)

18.1.2004 12 © by Hartmut Petters

„Function-Points" sind ein relatives Maß zur Bewertung der Funktionalität, d.h. des Leistungsumfangs eines Systems. Verfügt ein Unternehmen über Erfahrungszahlen, wie viel Aufwand pro „Function Point" im Mittel benötigt wird, um Software zu entwickeln bzw. zu pflegen, so können „Function Points" zur Aufwandabschätzung herangezogen werden. Wird in laufenden oder abgeschlossenen Projekten der mittlere Zeitbedarf pro „Function Point" bestimmt, so bekommt man ein Produktivitätsmaß.

Das „Function Point"-Verfahren wurde 1979 von Albrecht bei IBM entwickelt und seither von verschiedenen Autoren bzw. Gremien ergänzt und weiterentwickelt. Das Verfahren basiert auf der Grundidee, die folgenden Größen eines Software-Systems in geeigneter Weise zu zählen und zu bewerten (in Klammer die, von Albrecht verwendete Terminologie):

- Dateneingaben (external input)
- Datenausgaben (external output)
- Anfragen (external inquiry)
- Schnittstellen zu externen Datenbeständen (external interface file)
- Interne Datenbestände (logical internal file)

Dateneingaben, Datenausgaben und Anfragen werden an Hand der logischen Transaktionen, die das untersuchte System ausführen soll, gezählt. Tauchen gleiche Eingabe- bzw. Ausgabedaten in verschiedenen Transaktionen mit unterschiedlicher Verarbeitungslogik auf, werden diese mehrmals gezählt. Die Werte können alle bereits in der Anforderungsspezifikation gezählt werden, sofern diese hinreichend vollständig und detailliert ist.

Zählung + Gewichtung der „Function Points"

- *Eingaben, Ausgaben, Anfragen*
  - Zählen anhand der *logischen Transaktionen* des Systems
  - Gleiche Daten in verschiednen Transaktionen werden mehrmals gezählt
- *Externe / interne Datenbestände*,
  d.h. *logische Dateien* bzw. *Datengruppen in Datenbanken*
  (Gegenstandsgruppen oder Relationen) zählen
- Zählung ist in der *Anforderungsspezifikation* möglich
- Werte werden *gewichtet*
  - Schwierigkeitsgrad pro Element
    - einfach – mittel – komplex
  - Gewichtung der Summe mit dem Wertkorrekturfaktor VAF
- Es gibt unterschiedliche *Zählverfahren* für „Function Points"
  - Hier: Verfahren der
    „International Function Point Users Group (IFPUG)"

Jede Dateneingabe, Datenausgabe, Anfrage etc. wird als „einfach", „mittel" oder „komplex" bewertet und mit einem entsprechenden Gewicht versehen. Entsprechende Werte sind in der Literatur zu finden. Der Wertkorrekturfaktor VAF (Value Adjustment Factor) dient zur Korrektur der „Function Points" aufgrund der Komplexität.

„Function Points“ zur Aufwandschätzung

- *Mittlere Aufwand pro „Function Point“* muss bekannt sein
- *Umrechnungsfaktoren* müssen unternehmensspezifisch *kalibriert* und projektspezifisch angepasst werden
- *Umrechnungstabellen* und *Faustregeln* sind nur mit Vorsicht anzuwenden

- Faustregel von Jones
  - *Durchlaufzeit* [in Monaten] = $FP^{0,4}$
  - Anzahl *Mitarbeiter* = FP / 150
  - *Aufwand* = Durchlaufzeit x Anzahl Mitarbeiter = $FP^{0,4}$ x FP / 150

18.1.2004 14 © by Hartmut Petters

Sollen Function Points zur Aufwandschätzung verwendet werden, so muss der zu erwartende Aufwand pro Function Point bekannt sein. Entsprechende Kurven und Tabellen sind im Umlauf und in der Literatur zu finden. Der Aufwand pro Function Point ist stark abhängig von projektspezifischen und unternehmensspezifischen Faktoren, beispielsweise von dem Können und der Kompetenz der Mitarbeiter im Unternehmen, der eingesetzten Entwicklungsumgebung und dem Stellenwert von Qualität. Zu einer klaren Abschätzung muss zudem geklärt sein, welche Aufgaben in der Aufwandsabschätzung enthalten sind und welche nicht, z.B. ob die Aufwandsabschätzung selbst auch enthalten ist oder nicht.

Das Function Point-Verfahren hat den großen Vorteil, dass die benötigten Eingangsgrößen sich in den frühen Phasen eines Projekts leichter bestimmen lassen als beispielsweise die Größe des erwarteten Resultats bei COCOMO. Zudem sind mit Function Points Produktivitätsmessungen möglich, die weniger leicht verfälschbar sind als Messungen auf der Grundlage der erzeugten Programmzeilen.

Allerdings ist das Verfahren auf Informationssysteme zugeschnitten und daher auf andere Arten von Software (z.B. Prozessautomatisierung) nur beschränkt anwendbar. Ein weiterer Nachteil ist, dass das Function Point-Mass nicht additiv ist, d.h., die Summe der Function Points von n logisch zusammenhängenden Teilsystemen ist größer als die Anzahl der Function Points des Gesamtsystems. Dies liegt daran, dass die Schnittstellen zwischen je zwei Teilsystemen auf beiden Seiten als Schnittstellen zu externen Datenbeständen gezählt werden, während diese Daten bei der Betrachtung des Gesamtsystems als ein interner Datenbestand gezählt werden. Es ist daher nicht möglich, bei einem großen System die Function Points pro Teilsystem zu bestimmen und diese zu addieren.

Schätzung der Wartungskosten

- Kostenverhältnis: *Entwicklung* zu *Pflege*
  etwa 40:60 bis (bestenfalls) 50:50
- Faustregel für die *Kostenverteilung* von Pflegekosten
  - *60 % Verbesserungen*
  - *20 % Anpassungen*
  - *20 % Fehlerbehebung*
- Faustregel von C. Jones für den Pflegeaufwand
  - Benötigtes Pflegepersonal = *FP / 500*
    oder oder = *KDSI / 50*
- KDSI / Personen-Rate aus verschiedenen Projekten sehr unterschiedlich

Aus Messungen über Entwicklungs- und Pflegekosten von Software ergeben sich die beiden folgenden Faustregeln für Pflegekosten:

Regel 1

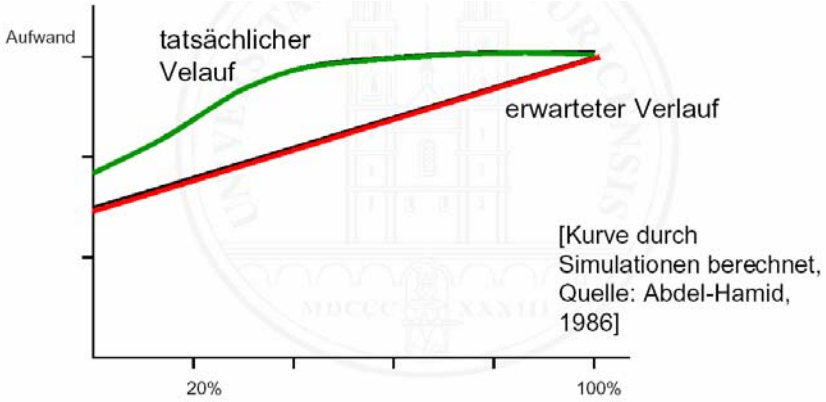
Das Kostenverhältnis zwischen Entwicklung und Pflege eines Software-Produkts liegt im Bereich von 30:70 bis 50:50. Je länger ein Produkt lebt und je schlechter seine Qualität ist, desto höher ist der Kostenanteil für Pflege.

Regel 2

Die Kosten für die Pflege eines Software-Produkts verteilen sich etwa wie folgt:
60% Verbesserung
20% Anpassungen
20% Fehlerbehebung

Einfluss der Schätzung auf den Aufwand

- Schätzung + tatsächlicher Aufwand sind nicht voneinander unabhängig
- Parkinson-Effekt
  - Korrelation von geschätztem + effektivem Aufwand



Schätzung und Aufwand sind nicht unabhängig voneinander. Dies ist durch das Gesetz von Parkinson: „Der Aufwand passt sich der verfügbaren Zeit an" dargestellt.

In Simulationen wurde durch Abdel-Hamid und Madnick eine signifikante Korrelation zwischen der Schätzung und dem tatsächlichen Aufwand festgestellt.

Dieser Effekt tritt vor allem dann auf, wenn in zu knapp kalkulierten Terminplänen Luft geschaffen wird. Dann werden nämlich all die Arbeiten tatsächlich gemacht, die bei ordnungsgemäßer Entwicklung eigentlich gemacht werden sollten (Dokumentation + Test, ...), die aber ansonsten weggelassen werden, damit die Termin überhaupt eingehalten werden können.

Zusammenfassung - Aufwandsabschätzung

- Aufwandsabschätzung ist sehr *schwer* + *heuristisch*
- Stark von der *Erfahrung* des Schätzers abhängig
- Konsens *mehrerer Experten verbessert Ergebnis*
- *Algorithmische Verfahren*
  - können nur *anwendungs- + unternehmensspezifisch* angewendet werden
  - erforderliche **Eckdaten** schwer ermittelbar
  - wenn Eckdaten vorhanden, dann *gute Ergebnisse*
- *„Function Point"-Verfahren*
  - kann auf *Spezifikation* angewendet werden
  - *normierte Zählverfahren* verwenden (Nachvollziehbarkeit)
- *Schätzung + Aufwand* sind *nicht unabhängig* voneinander

18.1.2004 17

Aufwandsabschätzung ist generell sehr schwierig, immer heuristisch und sehr stark abhängig von der Erfahrung der Experten, die die Schätzung vornehmen.

Bessere Ergebnisse werden erreicht durch:

- Teamarbeit, d.h. gemeinsame Abschätzung
- Durch Erfahrungen mit ähnlichen Projekten
- Durch mehrfache Iteration

Software Engineering

Informatik II.

7. Software-Entwicklung

– Realisierung –

Dipl.-Inform. Hartmut Petters

Möglichkeiten der Realisierung

- Optionen
  - *Entwerfen* und *Codieren*
  - *Entwerfen* und *Generieren*
  - *Konfigurieren* vorhandener Komponenten
- Aber immer
  - *Integrieren* und *prüfen*
- Wenn möglich
  - *Vorhandene Komponenten* wiederverwenden

Realisierung, Umsetzung eines Lösungskonzepts in eine funktionierende Lösung kann auf drei arten geschehen:

1. Codierung des Entwurfs und der damit verbundenen Ideen
2. Entwerfen und Generieren, in dem für die Umsetzung ein Code-Generator (4GL) eingesetzt wird
3. Konfiguration vorhandener Komponenten eines fertigen Systems

Detailentwurf + Codierung

- *Entwurf*
  - Entwurf des Programm-Ablaufs
    - Logische Abfolge der Befehle (schrittweise)
    - Logisches Ablaufdiagramm
  - Entwurf der Algorithmen + Datenstrukturen
  - Präzisierung des Lösungskonzepte
- *Codierung*
  - Umsetzung des Detailentwurfs in Programm-Code
  - Aufbau des Testbetts zur Überprüfung
  - Überprüfen + Verifizieren

In allen Fällen müssen die realisierten Einzelkomponenten zu einem System integriert werden und müssen die Komponenten wie auch das gesamte System auf Fehler und Funktionalität überprüft werden. Sowohl beim Entwerfen wie auch beim Codieren sollen wo möglich vorhandene Komponenten + Ideen genutzt werden.

Im klassischen Detailentwurf werden die zu codierenden Algorithmen und Datenstrukturen festgelegt. Dort wo das Lösungskonzept noch zu wenig detailliert ist, werden die nötigen Details erarbeitet. In der Codierung wird dieser Entwurf in eine Programmiersprache transformiert.

Dort wo der Code generiert werden kann bzw. soll, entsteht ein formelles Entwurfsdokument, dessen Syntax den Vorschriften des Generators genügen muss.

Typische Kandidaten für die Generierung sind Datenbanken und einfache Datenbankanwendungen einschließlich der Bildschirmmasken.

Aus hinreichend präzisen teilformalen Entwurfsmodellen können mit geeigneten Werkzeugen Code-Rahmen generiert werden, die dann manuell auszuprogrammieren sind.

Code-Generierung

- Entwurf
  - Erstellung eines *formalen* oder *teilformalen Entwurfsdokuments*
  - *Syntax* muss den Anforderungen des Code-Generators genügen
  - Dient als *Grundlage* für die Generierung
- Beispiele: Generierung von
  - *Datenbank-Schemata* (z.B. durch ARIS-Toolset)
  - *Einfache Anwendungen* (z.B. Symantec Café)
  - *Benutzeroberfläche* (z.B. Rational Rose)
  - *Dialogen*
  - *Codierungs-Rahmen* mit Übergabeparametern

Systematische Programmierung

- Geeignete Wahl von *Algorithmen* und *Datenstrukturen*
- Verwendung geschlossener Ablaufkonstrukte
- Saubere Gliederung der Software in Prozeduren und Module
- Verwendung *symbolischer Konstanten* und *abstrakter Datentypen*
- Verwendung *sprechender Bezeichnungen* + *selbstdokumentierende Namen* für Prozeduren, Variablen, Konstanten und Typen
- *Einhalten* vorgegebener *Programmierrichtlinien* für Formatierung, Namensgebung, Bezeichnungen + Abläufe
- Dokumentation aller Definitionen durch Kommentare
  - Datei-Header mit Beschreibung des logischen Ablaufs sowie der Ein- und Ausgabeparameter
  - Zeilenkommentare für logischen Blöcken + schwer verständliche Zeilen
  - Wirkung der Prozeduren bzw. eingesetzten Methoden
  - Bedeutung der Variablen + Konstanten
- *Defensives* Programmieren – lieber ausführlicher
- Vermeidung von *Trickprogrammierung* und Programmierung mit *Seiteneffekten*

Systematische Programmierung heißt:

- Geeignete Wahl der Algorithmen + Datenstrukturen
- Verwendung geschlossener Ablaufkonstrukte, so dass das Geheimnisprinzip sichergestellt ist und von außen keinerlei Einfluss auf den Ablauf und die Daten möglich ist
- Aufteilung der Software in Module + Komponenten
- Klare Regeln für die Programmierung bzgl. Namensgebung, Formatierung, Kommentierung und Dokumentation, so dass auch sichergestellt ist, dass die Programme zu einem späteren Zeitpunkt lesbar sind.
- Defensive Programmierung, d.h. ohne zu starke Nutzung der Möglichkeiten und damit unverständliche Programmierung

Werkzeuge zur Programmierung

- **Geschlossene Entwicklungsumgebung**
  - Visual-C / C++
  - Visual-Java
  - Visua-Basic,
  - ...
- **Einzelwerkzeuge**

Editoren	Code-Erstellung
Übersetzer / Compiler	Zwischencode-Erzeugung (relativ übersetzt)
Linker	Programm-Erzeugung
Testumgebung	Laufzeit-Umgebung mit Testmöglichkeiten
Make-Utilities + Makefiles	Übersetzungshilfen mit Beschreibung der Abhängigkeiten

Wenn möglich ist eine geschlossene Entwicklungsumgebung vorzuziehen, da die Entwicklungs- und Testwerkzeuge aufeinander abgestimmt sind.

Programm – Funktion - Routine

- Begriffs-Festlegung
  - Programm
    selbständig lauffähiges Software-Produkt, dass alle Teilfunktionen enthält, so dass es direkt gestartet werden kann.
  - Funktion
    Teilprogramm, dass eine *bestimmte Funktion* ausübt und beim Aufruf die erforderlichen *Parameter übergeben* bekommt.
    Nicht ohne Testrahmen lauffähig.
  - Routine
    Teilprogramm, dass eine *definierte Funktion* ausführt, aber *keinerlei Parameter* benötigt, sondern nur Veränderungen in dem Programmkontext vornimmt.
    Nicht ohne Testrahmen lauffähig.
    Programmierung über *Seiteneffekte*!

In der Programmierung wird wie angeführt zwischen dem Programm, den Funktionen und so genannten Routinen unterschieden. Dabei ist ein Programm generell selbst lauffähig, wobei dagegen eine Funktion oder Routine immer einen Testrahmen benötigt, der die lauffähige Umgebung für diese Funktion/Routine sicherstellt.

Basis-Kenntnisse

- *Grundaufbau*
  - Bibliotheken
  - Deklarationsdateien
  - Hauptprogramm
  - Unterprogramme
- *Variable + Konstanten*
  - Lebensdauer
  - Gültigkeit
- *Funktionen / Routinen*
  - Aufgaben
  - Übergabewerte / Rückgabewerte
  - Datenbereiche / Datenstrukturen
  - Testumgebung / Testbett

Programme sind prinzipiell immer gleich aufgebaut und bestehen aus Deklarationsdateien, in denen modulbezogen die Deklarationen, Makros und sonstige Festlegungen zusammengefasst sind, so dass bei Änderungen dieser Elemente nur genau diese Datei modifiziert werden muss.

Im Hauptprogramm wird der Ablaufrahmen zusammengefasst, der die einzelnen Module aufgrund der Ereignisse aufruft.

Bzgl. der Variablen + Konstanten ist generell zu beachten, wie die Lebensdauer und die Gültigkeitsbereiche dieser Elemente definiert ist, da gerade die zu unbeabsichtigten Nebeneffekten führen kann.

Auch wenn zum Aufruf und zur Verwendung von Funktionen nicht generell die entsprechende Deklaration erforderlich ist, so sollte dies trotzdem generell zur Klarheit der Programmierung gemacht werden, damit von vorn herein sichergestellt ist, dass einerseits die Funktion selbst den richtigen Wert liefert und andererseits die Aufrufparameter korrekt bedient werden.

Programmentwurf - Vorgehen

- **Spezifikation der Aufgaben / Detail-Entwurf**
- **Darstellung des Ablaufs**
  - Teilformal (Stichpunkte, Pseudo-Programmcode)
  - Ablaufdiagramme (Nassi-Schneidermann, o.ä.)
  - Datenstrukturen + Informationsfluss (SSAD, Entity-Relationship-Diagramme)
- **Festlegung der Typen**
  - Funktionsdeklarationen (void, int, char *, ...)
  - Übergabe-Parameter
  - Interne Variablen + Konstanten (Lebensdauer, Gültigkeit)
  - Rückgabewerte / Art der Rückgabe
- **Basis-Dokumentation im Programm- / Funktionskopf**
- **Testrahmen**
  - Einbettung in das „Main"-Programm
  - Testfälle (Standard, Sonderfälle, Eingaben, Ergebnisausgaben, ...)

Nachdem die eigentliche Aufgabe im Detail-Entwurf festgelegt wurde, sollte der innere Ablauf der Funktion mit geeigneten Hilfsmitteln (Flußdiagramme, Entity-Relationship-Diagramme, etc.) dargestellt werden.

Allgemeine Richtlinien

- Grundsätzlich bestehende *Programmier-Richtlinien* einhalten
- *Keine Verwendung* des Befehls *„goto"* und Sprungmarken
- *Keine Einsprünge* in Funktionen oder Routinen
- *Keine* Verwendung *„globaler Variablen"*
- Konstante, abstrakte Datentypen und Makros in *„Header-Files"* deklarieren
- Bearbeitungs-*Funktionen* eines *„abstrakten Datentyps" zusammenfassen* (Information Hidding)
- Möglichst nur *einen einzigen Rücksprung* aus einer Funktion
  - Einzige Ausnahme eventuell „Fehlerausgang"
- Möglichst *sprechende Namen* verwenden
- Möglichst den Code einer *Funktion nicht größer ale 1-2 A4-Seiten*
- Besser *ausführlicher programmieren* als komprimiert und unübersichtlich (keine Effizienzsteigerung, da Compiler optimiert)
- *Übergabe-Parameter*, allgemeinen *Ablauf* (Formel, etc.), *Ergebnisübergabe* etc. im Funktionskopf *beschreiben*
- Zeilenkommentare nur wo *sinnvoll* als *Block-Kommentar*

In allen Unternehmen, die professionelle Software-Entwicklung betreiben gibt es allgemeingültige Festlegungen bzgl. der Programmierung.

Dies beginnt mit der Formatierung der Programmzeilen bis hin zur Namensgebung und anderen Programmierkonventionen.

Damit der Programmcode von allen Programmierern gleichermaßen lesbar ist, hat sich jedes Mitglied des Teams an diese Vorschriften zu halten.

Beispiel - Trickprogrammierung

- Ein Unterprogramm in C zur Verkettung zweier Zeichenketten (Strings)

```
void strcat (char *s, char *t)
{
        while (*t++); for (t--;*t++=*s++;        /* t <- (t concat s)  */
}
```

- Ein Unterprogramm zum kopieren einer Zeichenkette

```
void strcpy (char *s, char *t)
{
        for (;*t++=*s++;)
}
```

- Besonderheiten dieser Art der Programmierung
  - Beide Male sind die eigentlichen *Schleifen leer*
  - Fragen hierzu
    - Macht diese Funktion / dieses Programm nichts?
    - Doch! Aber die *Funktionalität* ist in den Seiteneffekten versteckt!
  - *Elegant* + knapp zu *programmieren*
  - Schwer zu verstehen und zu prüfen!

Trickprogrammierung, so wie dies in C und C++ möglich ist, sollte vermieden werden, da dies – außer zur schlechten Lesbarkeit – nichts beiträgt.

Auch voll ausprogrammierte Schleifen und Abfragen werden von den heutigen Compilern so effizient übersetzt, dass dadurch näherungsweise der gleiche Code entsteht, wie wenn dieser in der dargestellten Weise extrem kompakt programmiert wurde.

Da heute die Prozessoren sehr schnell sind und Speicher nur noch eine untergeordnete Rolle spielt, da die Computer ohne große Kosten entsprechend ausgebaut werden können, sollte die Klarheit + Verständlichkeit der Programmierung im Vordergrund stehen.

Test + Integration

- Prüfung jeder Einzelkomponente
  - *Syntaktische* Richtigkeit (durch Compiler)
  - *Semantische / Inhaltliche* Richtigkeit
    - *Selbstinspektion* des Codes durch Autor(in)
    - *Formale Inspektion* durch Dritte
    - *Komponententest*
- Schrittweise Integration der Komponenten
  - *Aufwärts*- oder *Abwärtsintegration*
  - *Integrationstests*
- Systemtest
- Abnahme (Code-Inspektion / Code-Review)

Der Ablauf der Tests und der Integration ist aufgrund der Modularisierung vorgegeben.

Nach erfolgreichem Test der Funktionen + Module sind diese im Zusammenspiel mit den zu integrierenden Komponenten zu testen. Erst wenn jeder Test vollständig erfolgreich durchgeführt wurde, kann die nächste Stufe angegangen werden.

Bei nachträglichen Änderungen sind die davon betroffenen Einzelmodule wieder nach den gleichen Kriterien zu überprüfen, wie ursprünglich und danach ebenso wieder die Integration in die nächste Stufe.

Da nur absolut nachvollziehbare Tests sinnvolle Ergebnisse liefern, sind alle einzelnen Testschritte aufzuzeichnen und in exakt der gleichen Reihenfolge jedes Mal durchzuführen, da andernfalls die Fehlerfreiheit unter den festgelegten Testbedingungen nicht gewährleistet werden können.

Insbesondere bei komplexen Lösungen ist ein entsprechender Test der Einzelmodule und die Aufzeichnung der einzelnen Schritte extrem aufwendig. Hierfür gibt es geeignete Testwerkzeuge, die den Testablauf automatisch aufzeichnen und jederzeit automatisch ablaufen lassen können, damit der Testkandidat jederzeit die exakt gleichen Testbedingungen hat wie vorgegeben.

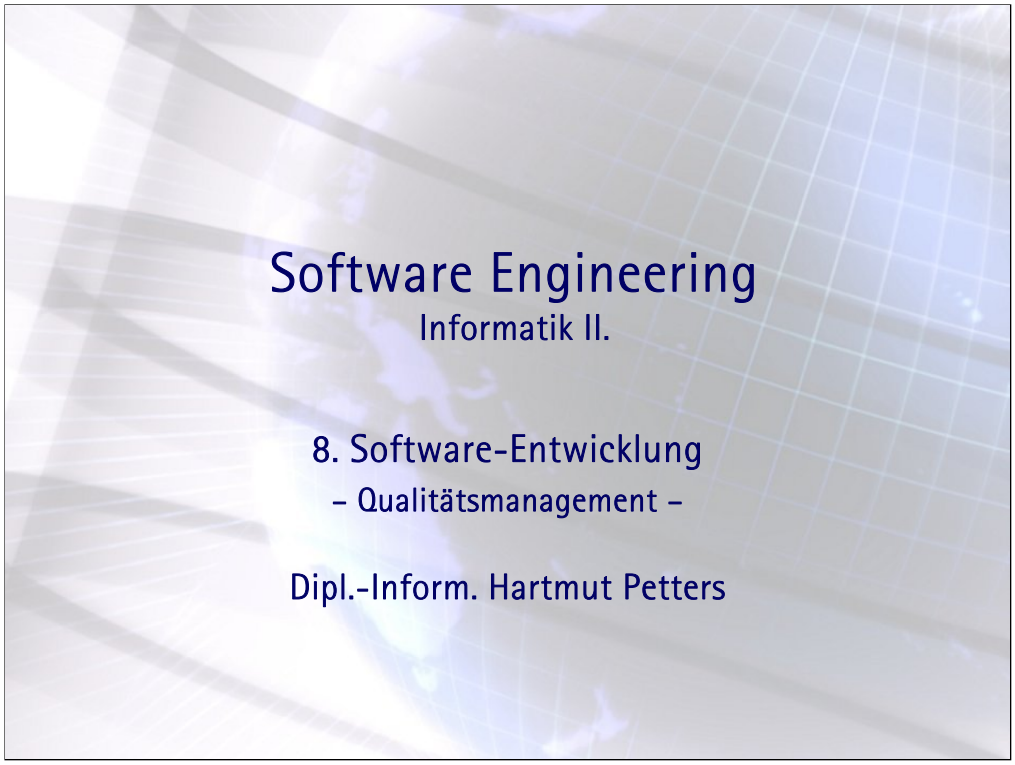
Zusammenfassung

- Nur *kurzer Überblick* bzgl. Realisierung / Umsetzung
- Umsetzung *systematisch* angehen
  - Grundsätzliche *Gedanken zum Lösungsansatz*
  - Ablauf er Funktionen skizzieren / *Ablaufdiagramme*
  - *Übergabewerte + Ergebniswerte* festlegen
  - *Header-Beschreibung* mit Funktionsbeschreibung
  - *Codierung* der Funktionen (evtl. mit Testausgaben)
  - *Testbett* erstellen, um lauffähigen Rahmen zu schaffen
  - Einbettung der Funktion in *reale Modul-Umgebung*
- Jede *Integrations-Stufe* getrennt *testen*
- Endabnahme durch Dritte

Die Umsetzung und Realisierung des Lösungskonzepts in Software unterliegt harten Rahmenbedingungen und entspricht den üblichen Regeln eines „Handwerks".

Die kreative Freiheit des Programmierers kann sich zuvor im Systementwurf- und Design zeigen, aber nicht bei der Umsetzung, da hier aufgrund der Rahmenbedingungen die Regeln einer systematischen Entwicklung unter Einhaltung des Geheimnisprinzips Gültigkeit haben.

Alle Tests sind so durchzuführen und zu dokumentieren, dass diese jederzeit in gleicher Art + Weise wieder durchgeführt werden können, da nur so die Fehlerfreiheit bei Veränderungen gewährleistet werden kann.



Qualitätsmanagement ist eine umfassende Aufgabe, der gerade bei der Software-Entwicklung eine hohe Bedeutung zukommt.

Grundlagen + Definitionen – 3te

- Qualitätsmanagement (quality management)
  aufeinander abgestimmte *Tätigkeiten* zum *Leiten* und *Lenken* einer Organisation bezüglich Qualität.

  Leiten und Lenken bezüglich Qualität umfassen üblicherweise das Festlegen der *Qualitätspolitik* und der *Qualitätsziele*, die *Qualitätsplanung*, die *Qualitätslenkung*, die *Qualitätssicherung* und die *Qualitätsverbesserung* (ISO 9000 ff.)
- Qualitäts-Managementsystem /QM-System (quality management system)
  Management-System zum *Leiten* und *Lenken* einer Organisation bezüglich der Qualität (ISO 9000 ff.)

Da einerseits in den unterschiedlichen Unternehmen eine unterschiedliche Auffassung bzgl. Qualitätsmanagement besteht und andererseits die Abläufe der Unternehmen generell unterschiedlich sind, kann es kein „Standard-System" für Qualitätsmanagement geben.

Unter Qualitätsmanagement werden alle Aktivitäten zusammengefasst, die die Qualität und das Verständnis der Mitarbeiter bzgl. der Erzeugung qualitativ hochwertiger Produkte betrifft.

Was sagt die ISO 9000 ff.

- Was ist ein Qualitätssicherungssystem (QS-System)?
  - Ein QS-System stellt die *Aufbau-* und *Ablauforganisation* eines Unternehmens für den übergreifenden Funktionsbereich QS dar.
  - Es gibt aufgrund der Vielfältigkeit der Unternehmen *kein "normiertes" QS-System*
  - Die *ISO 9000/EN 29000 definiert Modelle*, an denen sich Vertragspartner bei der Ausgestaltung von QS-Systemen orientieren können.

Alle Belange bzgl. Qualitätsmanagement werden in der ISO 9000 ff. festgelegt.

Da es aber generell kein Standard-System geben kann, werden hier die Rahmenbedingungen dargestellt und festgelegt, die für ein Qualitätsmanagement-System relevant sind.

Was regelt die ISO 9000 ff.

- Die *ISO 9000* und ihre Folgenormen 9001/9002/9003

Die *ISO 9000* wird dann angewendet, wenn die *Verträge zwischen zwei Parteien* spezielle Leistungen hinsichtlich der *Spezifikation und Konstruktion von Erzeugnissen* verlangen, und die technischen Daten der Produkte nur in Form von Leistungsanforderungen vorliegen oder erst erarbeitet werden müssen

- ISO 9001 — für die Phasen Spezifikation, Konstruktion, Produktion, Installation und Service
- ISO 9002 — für die Phasen Produktion und Installation
- ISO 9003 — für die Phasen Test und Endabnahme

In der ISO 9000 ff. werden u.a. vor allem die Rahmenbedingungen festgelegt, die für die Phasen

- Spezifikation
- Konstruktion + Entwicklung
- Produktion
- Installation
- Test + Abnahme
- Service + Pflege

Relevanz haben.

Dies sind insbesondere auch die Phasen, die in der Software-Entwicklung von Bedeutung sind.

ISO 9000 ff. – Relevanz

- Die technische *Produktdokumentation*

  Die technische Produktdokumentation ist die Gesamtheit, der während der Lebensphase eines Erzeugnisses erstellten technischen Dokumente.

- Relevanz Software-Engineering
  - ISO 9001 § 4.5.1 *Herausgabe* und Genehmigung von Dokumenten
  - ISO 9001 § 4.5.2 **Änderung / Modifikation** von Dokumenten
  - ISO 9001 § 4.8 Identifikation und *Rückverfolgbarkeit* von Dokumenten
    Überprüfung von Dokumenten

Besonderes Augenmerk wird dabei auf die Produkt- und Projekt-Dokumentation und dem Umgang mit Veränderungen bei der Dokumentation gelegt.

Sechs Grundsätze des Qualitäts-Managements

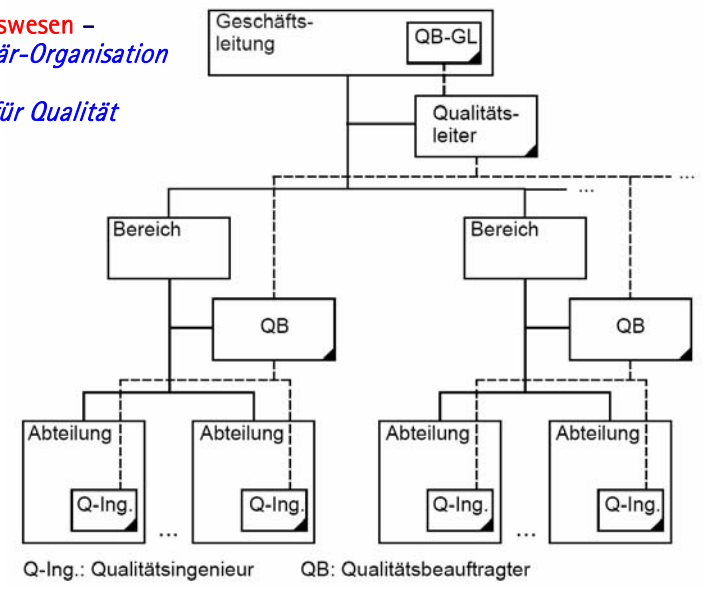
1. Qualität muss *erzeugt* werden, sie kann *nicht erprüft* werden
2. Qualität bezieht sich immer auf *Produkte* und auf *Prozesse*
3. *Qualitätsverantwortung* ist untrennbar verbunden mit Sach-, Termin- und Kostenverantwortung
4. Das Qualitätswesen erbringt *Dienstleistungen* und ist verantwortlich für die *Ermittlung (Messung) der Qualität*
5. Das Qualitätswesen muss einen unabhängigen *Berichtserstattungspfad* haben, der bis zur Geschäftsleitung geht
6. Die Entwickler müssen über die Qualität ihrer Arbeit *orientiert* werden

Die aufgeführten 6 Grundsätze können hier als grundsätzliche Rahmenbedingungen gewertet werden.

Im Vordergrund steht dabei immer, dass Qualität „erzeugt" werden muss und nicht erprüft. D.h. die Geschäftsprozesse müssen von vornherein darauf ausgelegt sein, dass bereits im Entstehungsprozess die Qualität der Erzeugnisse im Vordergrund steht und diese die Messlatte für die Produktion bedeutet. Es kann nicht sein, dass die Qualität generell nur durch entsprechende Endtests nach Produktionsende erfolgen und so die Qualität nachträglich durch beliebige Nachbesserungen sichergestellt wird.

Die Aufbauorganisation

Das Qualitätswesen –
eine *Sekundär-Organisation*
mit den
Fachleuten für Qualität



Die Qualitätssicherung und die Organisation, die diese zu gewährleisten hat, ist eine der vordringlichsten Aufgaben im Unternehmen und darf somit nicht in den Linienfunktionen verankert, sondern muss als parallele „Schattenorganisation" zur Aufbau- und Ablauf-Organisation direkt der Geschäftsleitung unterstellt sein. Nur so kann eine Einflussnahme der Linienfunktionen ausgeschlossen werden.

Die Ablauforganisation

- Das Qualitäts-Management-System regelt alle qualitätsrelevanten
  - *Kompetenzen*
  - *Verantwortlichkeiten*
  - *Beziehungen*
- Qualitätsaufgaben *in* die *Unternehmensprozesse* integrieren
- Möglichst *wenig* Qualitätsaufgaben *separat geregelt*

In der Qualitätsmanagement-Organisation werden alle Belange bzgl. Qualitätsmanagement geregelt und entschieden.

Qualitäts-Management-Verfahren – 1te

Qualitätsplanung

- Definition der Qualitätsziele
  - ↳ Das wollen wir erreichen!

Qualitätslenkung

- Konstruktive Maßnahmen
  - ↳ So müssen wir arbeiten!

Qualitätsprüfung

- Analytische Maßnahmen
  - ↳ Haben wir die Ziele erreicht?
  - ↳ Haben wir richtig gearbeitet?

Das Qualitätsmanagement muss über die Stufen

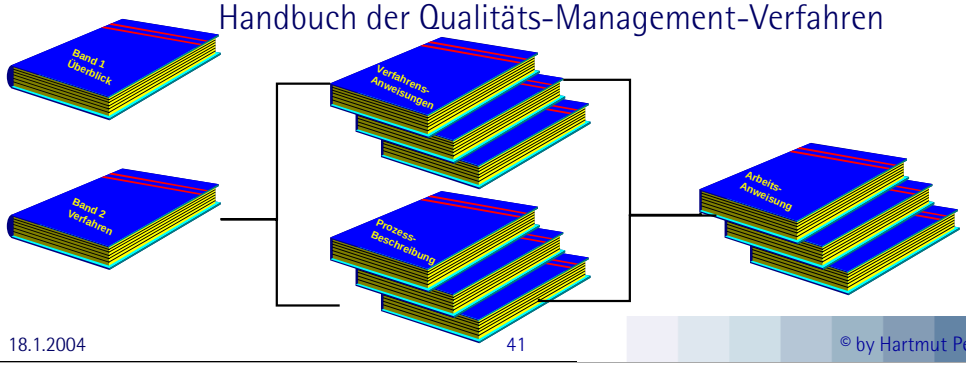
- Qualitätsplanung
  Zieldefinition, Vorgaben-Festlegung
- Qualitätslenkung
  Festlegung und Durchführung konstruktiver Maßnahmen zur Qualitätssicherung bis hin zu Entwicklungsvorgaben
- Qualitätsprüfung
  im Sinne von Bestätigung der Qualität

Qualitäts-Management-Verfahren – 2te

- Qualitätsplanung
  - *Im Allgemeinen*
    Aufbau und Dokumentation des QM-Systems, allgemeine Qualitätsziele
  - *Im Speziellen*
    Festlegung der Qualitätsziele für individuelle Projekte
- Qualitätslenkung
  - *Im Allgemeinen*
    - Methoden, Sprachen, Werkzeuge
    - Ausbildung
    - Vereinheitlichung der Arbeitsweise
  - *Im Speziellen*
    Maßnahmen der Projektführung zur Erreichung der geplanten Qualität
- Qualitätsverfolgung
  - Folgt später

Dokumentation Qualitäts-Managements – 1te

- Qualitäts-Handbuch
  beschreibt das Qualitäts-Management-System
  - *Band 1* (eventuell Offenlegung für Kunden)
    Qualitäts-Management-*Organisation*, *Überblick* über die Qualitäts-Management-Maßnahmen
  - *Band 2* (nur intern / vertraulich)
    Handbuch der Qualitäts-Management-Verfahren



18.1.2004 41 © by Hartmut Petters

Die Dokumentation des Qualitätsmanagement-Systems erfolgt im so genannten Qualitätshandbuch. Dieses Qualitätshandbuch ist in 2 Bereiche unterteilt, beide sollten jedem Mitarbeiter dabei vertraut und jederzeit zugänglich sein.

Im Band 1 werden all die allgemein gültigen Teile des Qualitätsmanagement-Systems beschrieben, die als Überblick und Basisdokumentation nach außen kommuniziert werden können

Im Band 2 sind die internen Verfahren und Vorgaben festgeschrieben, die in den Geschäftsprozessen zur Umsetzung der Qualitätsmanagement-Ziele dienen und die Vorgaben für die Mitarbeiter festlegen.

Dokumentation Qualitäts-Managements – 2te

- **Software-Qualitätsplan**
  Vorgaben betreffend Qualität für die Entwicklung von Produkten
  - In der Regel existiert ein allgemeiner Rahmenplan
  - Rahmenplan wird bei Bedarf projektspezifisch ergänzt
  - Manchmal auch vom Kunden vorgegeben
- **Software-Entwicklungs-Handbuch**
  detaillierte Entwicklungs-Richtlinien und Ausführungsbestimmungen
  - Software-spezifisch
  - Entlastet das Qualitäts-Handbuch von Software-Details

Alle Maßnahmen im Bereich Qualitätsmanagement sind zu dokumentieren.

Qualitätssicherung

Die bloße Existenz eines Qualitäts-Management-Systems genügt nicht – ist nicht hinreichend!

- Qualitätssicherung (quality assurance)
  Darlegung des Qualitäts-Managements, d.h. alle Tätigkeiten zur *Schaffung von Vertrauen*, dass die Qualitätsanforderungen erfüllt werden.

- Überprüfung durch Audits
  - *Internes* Audit
    durch das Qualitätswesen – die Qualitätsbeauftragten
  - *Externes* Audit
    Überprüfung durch eine externe, auf Qualitäts-Management spezialisierte Organisation
    - Auf Veranlassung des Kunden
    - Zur *ISO-9000 Zertifizierung*

18.1.2004 43

Da die reine Existenz eines Qualitätsmanagement-Systems im Unternehmen nicht hinreichend ist für die Qualitätssicherung, wird ein Unternehmen nur zertifiziert, wenn es diesbezüglich von geprüften Auditoren regelmäßig kontrolliert und überwacht wird.

Entsprechende Audits finden alle zwei Jahre durch entsprechende externe Prüfer statt. Bei diesen Audits wird die gesamte Organisation bzgl. der Vorgaben und der Umsetzung im Bezug auf Qualitätsmanagement hin überprüft.

In den dazwischen liegenden Jahren findet i.d.R. generell ein internes Audit statt, bei dem die festgestellten Mängel und Verbesserungspotenziale auf erfolgte Veränderungen hin überprüft werden. Diese internen Audits erfolgen meist durch eigene Qualitätsbeauftragte.

Validierung + Verifikation – 2te

- Validierung (validation)
  Der Prozess der Beurteilung eines Systems oder einer Komponente während oder am Ende des Entwicklungsprozesses, mit dem Ziel, festzustellen, ob die spezifizierten *Anforderungen erfüllt* sind.
- Verifikation (verification)
  1. Der Prozess der Beurteilung eines Systems oder einer Komponente mit dem Ziel, festzustellen, ob die *Resultate* einer gegebenen Entwicklungsphase den Vorgaben für diese Phase *entsprechen.*
  2. Der *formale Beweis der Korrektheit* eines Programms

Validierung bedeutet den Erfüllungsgrad der Anforderungen zu überprüfen, wogegen Verifikation die Anpassung der Lösung an die Anforderungen mit einschließt.

Prüfverfahren

- Review
  Eine formell organisierte Überprüfung eines Arbeitsergebnisses durch eine Gruppe von Gutachtern
- Test
  Die Überprüfung eines Programms, ob dieses bei gegebenen Eingaben die erwarteten Resultate liefert
- Prototyp
  Eine Vorab-Realisierung eines kritischen Systemteils

Seltener:

- Simulation
  Modellierung eines bestimmten Aspekts eines Systems zur Überprüfung seines Verhaltens

In Spezialfällen:

- Formale Verifikation
  Mathematischer Korrektheitsbeweis
- Model Checking
  Überprüfung der Gültigkeit wichtiger Eigenschaften durch systematisches, automatisiertes Erproben

Die üblicherweise eingesetzten Prüfverfahren sind Reviews und Tests. In Ausnahmefällen werden Prototypen zur Darstellung der Machbarkeit als Vorab-Realisierung kritischer Teile herangezogen.

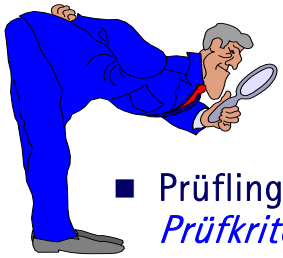
Reviews

- Review
  Eine *formell organisierte* Zusammenkunft von *Personen* zur inhaltlichen oder formellen *Überprüfung* eines Produktteils (Dokument, Programmstück, etc.) nach vorgegebenen Prüfkriterien und Prüflisten.
- Ziele eines Reviews
  - Aufzeigen der *Schwachstellen* und *Mängel* des Prüflings
  - Betätigen der *Stärken* des Prüflings
  - Beurteilung der Qualität
- Abgrenzung
  keine Reviews im hier betrachteten Sinn sind
  - *informelle Prüfungen*, z.B. Durchlesen durch Kollegen
  - *Management-Reviews* zur Überprüfung von Kosten + Terminen

Reviews sind formal organisierte Zusammenkünfte der beteiligten Personen und der Prüfer, um die entsprechenden Dokumente und Programme zu überprüfen und im Hinblick auf festgelegte Ziele (Pflegbarkeit, Erweiterbarkeit, Fehlerfreiheit) zu überprüfen.

Das generelle Ziel ist es Schwachstellen und Mängel frühzeitig herauszufinden und gegebenenfalls zu beseitigen.

Review-Formen – 1te: Inspektion



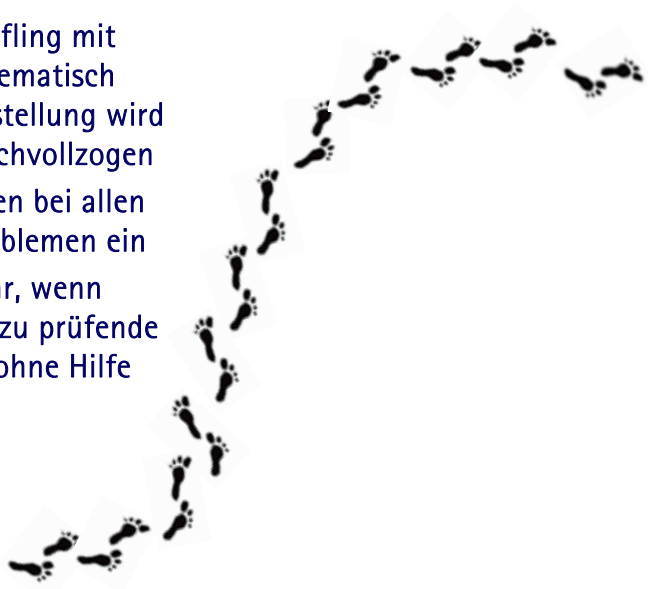
- Prüfling wird von Gutachtern *nach vorgegebenen Prüfkriterien systematisch* Zeile für Zeile inspiziert
- Setzt voraus, dass die Gutachter das zu prüfende Material ohne Hilfe lesen und verstehen können.
- *Wirksamste* Review-Technik
- *Wo immer möglich*
  - ↳ Reviews als *Inspektion* durchführen!
- Nachstehende Ausführungen gelten primär für Inspektionen

Bei Inspektionen als Review wird von einem oder mehreren Gutachtern nach fest vorgegebenen Kriterien das Dokument oder das Programm zeilenweise inspiziert und auf mögliche Fehler und Schwachstellen hin untersucht.

Die Gutachter müssen dabei der Technologie mächtig sein und die Programmiersprache sehr gut verstehen.

Review-Formen – 2te : Walkthrough

- Autor geht Prüfling mit Gutachter systematisch durch, die Darstellung wird gemeinsam nachvollzogen
- Gutachter haken bei allen Mängeln + Problemen ein
- Auch einsetzbar, wenn Gutachter das zu prüfende Produkt nicht ohne Hilfe lesen können



Im „Walkthrough" als weitere Art des Review werden die Dokumente und Programme zusammen mit dem Entwickler systematisch durchgearbeitet und auf Schwachstellen entsprechend den Kriterien untersucht.

Beim „Walkthrough" muss der Gutachter der Programmiersprache nicht absolut mächtig sein, da der Ersteller die Funktionen erklärt und erläutert und die Hinterfründe der Lösung präsentiert.

Warum Reviews?

- Fehler finden
  - Inspektionen *finden* die meisten *Fehler*
  - Reviews sind *billiger* als testen
    - Weniger Vorbereitungs- und Durchführungsaufwand
    - Findet Fehlerursachen, nicht nur Symptome
    - Nicht jede Software-Einheit ist testbar, aber jede ist review-fähig
- Besser werden
  - *Richtiges bestätigen* / beibehalten / *verstärken*
  - Arbeitsweise *vereinheitlichen*
  - Ergebnisse *auswerten*
    - ↳ Prozesse + Qualität *lenken*
  - Aus Schwächen + Stärken *lernen*
    - Wissenstransfer von den guten zu den schlechten Software-Entwicklern
  - Reviews sind *wirtschaftlich*!

Hauptziel der Reviews ist es

Fehler zu finden und Mängel aufzudecken.

Durchführung eines Reviews (Inspektion)

- Planung
  - Termin ein*planen*
  - Aufwand *budgetieren*
  - Teilnehmer *einladen*
  - *Material verteilen*
- Vorbereitung
  - Teilnehmer *bereiten sich individuell vor*
  - *Inspizieren* den Prüfling
  - *Notieren die Befunde / Ergebnisse*
- Sitzung
  - *Moderiertes Zusammentragen* und *Bewerten* der Befunde
  - *Erstellen* des Review-Berichts
- Überarbeiten + Nachkontrolle (nach dem Review-Termin)
  - *Entscheidung* über Änderungen
  - *Durchführung* der Änderungen
  - *Nachkontrolle* durch Projektverantwortlichen oder neues Review

18.1.2004 50

Ein Review läuft nach klaren Regeln + Vorgaben ab:

Planung

bereits in der Projektplanung sind Reviews sowohl zeitlich wie kostenmäßig zu berücksichtigen

Ablauf des Reviews

auch dieser ist als Vorgabe für alle Teilnehmer vorab zu planen, in dem die Teilnehmer rechtzeitig eingeladen und mit dem zu prüfenden Material ausgestattet werden.

Vorbereitung

Da der eigentliche Review effizient ablaufen soll, haben sich alle Teilnehmer entsprechend vorzubereiten. D.h., die Inspektoren haben bereits im Vorfeld die übergebenen Programme und Dokumentie im Hinblick auf die Ziele des Reviews zu prüfen und zu kommentieren, so dass der eigentliche Review nur zur Information und Maßnahmen-Entscheidung dient.

Rollen der Beteiligten (Inspektion)

- Moderator
  - *organisiert*
  - *leitet*
- Gutachter
  - *prüft*
  - *nennt Befunde*
  - *bewertet*
  - verhält sich *positiv* + *kooperativ*
- Schreiber
  - *protokolliert* für alle Beteiligten *sichtbar*
- Autor
  - *hört zu*
  - verhält sich *passiv*

18.1.2004 51

Der Review-Prozess ist generell durch einen Moderator zu steuern, der die Einhaltung der Vorgaben

- Einladung
- Vorbereitung
- Review-Ziele
- Zeitrahmen
- Dokumentation

sicherstellt.

Review-Regeln – 1te

- Material *rechtzeitig* vor der Sitzung *verteilen*
- Alle Teilnehmer *rechtzeitig einladen*
- Alle Gutachter kommen vorbereitet (unverzichtbar bei Inspektionen!)
- *3 bis 7* Beteiligte
- Dauer des Reviews *maximal 2 Stunden*
- Nicht mehr Material verteilen, als in der Besprechung bewältigt werden kann
- Probleme nur *erkennen* – nicht lösen!
- *„Dritte Stunde"* nach der Review-Sitzung zur Diskussion von Problemlösungen

Generell gilt:

- Das entsprechende Material ist frühzeitig zur Verfügung zu stellen, damit die Prüfer sich vorbereiten können
- Klärung der Termine + der Verfügbarkeit der erforderlichen Personen
- Maximal 3-7 Beteiligte, da sonst die Effizienz gefährdet ist
- Maximale Dauer des Review-Meetings mit 2 Stunden kalkulieren

Review-Regeln – 2te

- *Positives + Negatives* nennen
- *Keine Stilfragen* diskutieren
- *Produkt bewerten* – nicht den Ersteller
- Review-Bericht *niemals zur schematischen Bewertung von Mitarbeitern* verwenden
- Anhand von *Standards + Prüflisten* bewerten
- *Fehler in Referenz-Unterlagen* ebenfalls erheben / festellen und in separater Befundliste notieren

Zielorientiert das Meeting durchführen und keine Grundsatzdiskussionen führen.

Das Review-Meeting dient nicht der Mitarbeiter-Bewertung sondern der Sicherstellung der Produkt-Qualität.

Testen

- Test
  Ausführung eines Programms oder eines Software-Systems zwecks *Überprüfung*, ob dieses bei *gegebenen Eingaben* die *erwarteten Resultate* liefert
- Zwei mögliche Testziele
  - Möglichst viele *Fehler finden*
    - Myers (1979):
      Testen ist der Prozess, ein Programm mit der Absicht auszuführen, Fehler zu finden
  - Aus fehlerfreier *Ausführung mit definierten Testdaten* statistisch auf fehlerfreie *Ausführung mit realen Daten* im Einsatz *schließen*

Unter Test verstehen wir die probeweise Ausführung eines Programms zur Überprüfung der Fehlerfreiheit und der Ausführung der geplanten Funktionen.

Testen - Grundlagen

- Jeder Test ist eine Stichprobe
- *Korrektheit / Fehlerfreiheit* kann durch Testen nicht bewiesen werden
  - Beispiele
    Addition von zwei 32-Bit-Zahlen: 2^{64} mögliche Testfälle
    Bearbeitung einer Zeichenkette
    mit 32 Zeichen: 2^{256} mögliche Testfälle
- *Erwartete Ergebnisse* müssen im Voraus *bekannt* sein
  - Testen gegen die Spezifikation
  - Testen gegen vorhandene Ergebnisse (Regressionstest)
- *Unvorbereitete* und *undokumentierte* Tests sind *Zeitverschwendung*
- Testen findet *Fehlersymptome*, keine *Fehlerursachen*
- Nach dem Test
  - Fehlerursachen suchen, finden und beheben (Debugging)

18.1.2004 55

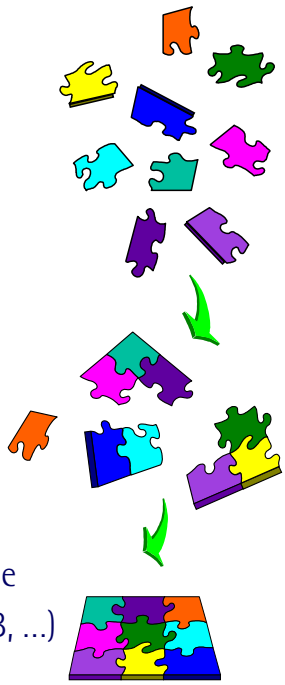
Jeder Test ist nur eine Stichprobe.

Damit die Modifikation und zielorientierte Änderung des Programm-Codes nicht bereits funktionierende Programmteile negativ beeinflusst ist sicherzustellen, dass bei wiederholten Tests die ursprünglichen Tests in gleicher Art und Weise wieder holt werden. Dies kann nur durch eine kontinuierliche Aufzeichnung des gesamten Testablaufs gewährleistet werden.

Üblicherweise wird für diese Wiederholungstest ein so genannter Testmanager eingesetzt, der die Aufzeichnung der Testschritte übernimmt und die Folgetests automatisch durchführt und die Ergebnisse vergleicht, so dass Abweichungen kurzfristig festgestellt werden können.

Testgegenstände – wie testen

- Einzelkomponenten (Modultest)
  - Jede Funktion
  - Jede Routine
- Baugruppen (Integrationstest)
  - Jedes Teilprogramm
  - Jede geschlossene Funktionseinheit
  - Jedes Objekt
- Systeme (Systemtest, Abnahmetest)
  - Alle zusammenhängenden Teilsysteme
  - Komplettsystem + Zusatzsysteme (DB, ...)



Der Testablauf sieht vor, dass von den kleinsten Einheiten ausgegangen und erst nach erfolgreichem Abschluss der Stufe zur nächsten übergegangen wird. Diese Tests gehen von den einzelnen Funktionen und Modulen aus, über die einzelnen Integrationsstufen hinweg, bis hin zum Gesamtsystem, das in die Anwenderumgebung zu integrieren ist.

Testablauf – 1te

- *Planung*
  - Teststrategie
    Was? Wann? Wie? Wie lange?
  - Einbettung des Testens in die Entwicklungsplanung
- *Vorbereitung*
  - Auswahl der Testfälle
  - Bereitstellung der Testumgebung
  - Erstellung der Testvorschriften
- *Durchführung*
  - Testumgebung einrichten
  - Testfälle nach Testvorschrift ausführen
  - Ergebnisse aufzeichnen
  - Prüfkandidaten während des Tests nicht verändern

Auch die Test erfordern eine klare Planung, Vorbereitung und Durchführung bis hin zur Dokumentation des Ablaufs, der Testdaten und Testschritte bis hin zu den Ergebnissen.

Testablauf – 2te

- *Auswertung*
  - Testbefunde zusammentragen + dokumentieren
- *Fehlerbehebung* (ist ein Bestandteil des Tests!)
  - Gefundene Fehler (-Symptome) analysieren
  - Fehlerursachen bestimmen (Debugging)
  - Fehler beheben
  - Test der Korrekturen

Nach erfolgten Tests sind die Ergebnisse auszuwerten und zu dokumentieren

Testverfahren

- Funktionsorientierter Test (Black-Box-Test)
  - *Auswahl* der Testfälle aufgrund der *Spezifikation*
  - Programmstruktur braucht *nicht bekannt* zu sein
  - Geforderte *Eigenschaften* des Prüflings möglichst vollständig *durch Testfälle abdecken*
  - *Fehlerträchtige Situationen* gezielt *provozieren*
- Strukturorientierter Test (White-Box-Test, Glass-Box-Test)
  - Auswahl der Testfälle aufgrund der *Programmstruktur*
  - Spezifikation muss bekannt sein (für erwartete Ergebnisse
  - *Programmstruktur* möglichst vollständig *durch Testfälle abdecken*

Bei den Tests wird generell zwischen zwei grundsätzlichen Arten unterschieden

1. „Black-Box-Test" oder funktionsorientiertem Test
   bei dem die eigentliche Programmierung außer Acht gelassen wird und die Funktion als „Black-Box" betrachtet wird, die eine festgelegte Funktion auszuführen hat. Der Test erfolgt dabei durch Auswahl geeigneter Testfälle und den durch diese Testfälle generierten Ergebnisse durch die getestete Funktion.
2. „White-Box-Test" oder strukturorientierter Test
   bei dem die Testfälle nicht aufgrund der Spezifikation sondern aufgrund der Programmstruktur und den darin enthaltenen Entscheidungszweigen festgelegt werden. Die Spezifikation dient dabei dazu, die Ergebnisse abzusichern. Dabei wird angestrebt, dass alle Programmverzweigungen im Test überprüft werden.

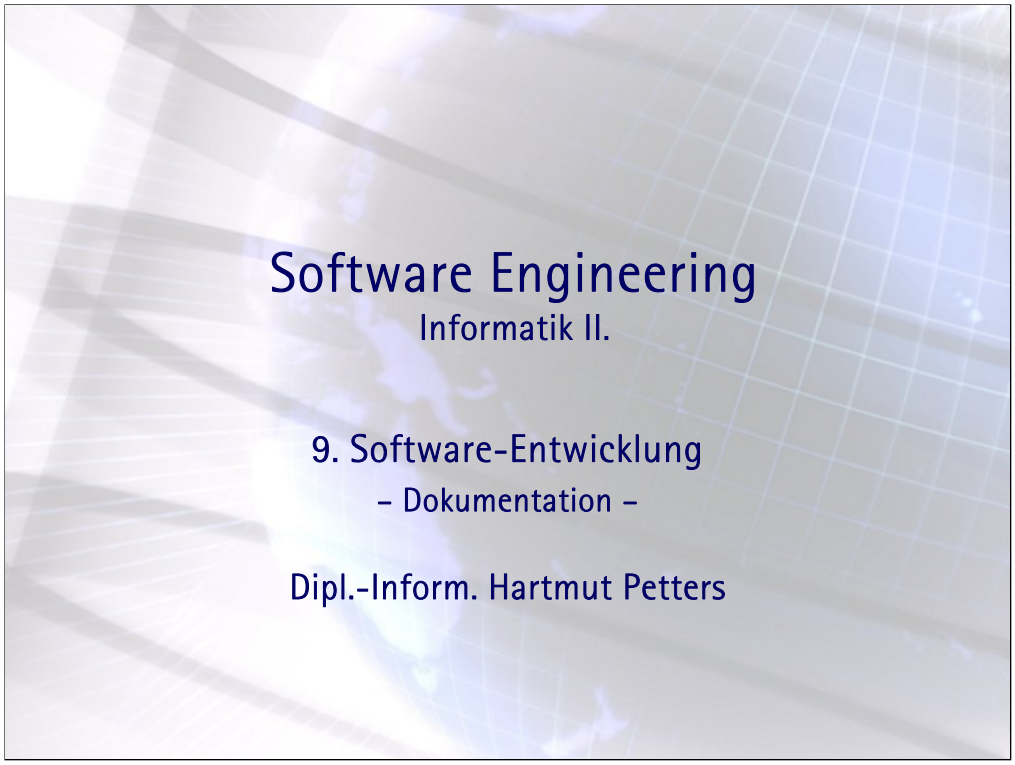
Zusammenfassung - Qualitätsmanagement

- **Qualität ist nicht per Definition gegeben**
  - Qualität muss erzeugt – nicht erprüft werden
  - Betrifft Produkt + Prozesse
  - Qualitätsverantwortung ist fest mit Sach-, Termin- und Kostenverantwortung verbunden
  - Für Qualität sind „alle" verantwortlich – Eigenverantwortung eines jeden
  - Rahmen ist definiert durch ISO 9000 ff.
- **Qualitätsmanagment-System (QM-System)**
  - Empfehlung über Rahmen des QM-Systems + Vorgehen
  - Nie Standard – immer unternehmensspezifisch
  - Eigene Parallel-Organisation zur Ablauforganisation
  - Planung – Lenkung – Prüfung
  - Permanente Aufgabe zur Sicherstellung der Qualität

Qualitätsmanagement ist eine sehr wichtige Grundaufgabe in den Unternehmen und bedarf der akkuraten Planung.

Qualitätsmanagement betrifft insbesondere die Abläufe (Prozesse) im Unternehmen und die dadurch produzierten Produkte

Qualitätsmanagement ist nicht als Standard erhältlich, da die Geschäftsprozesse in den Unternehmen generell individuell sind.



Professionelle Software-Entwicklung setzt eine durchgängige Dokumentation der Abläufe, der Anforderungen, der Planung, der inneren und der äußeren Funktionen voraus.

Nur mit einer vollständigen Dokumentation ist Software als Produkt auch über längere Zeit hinweg pflegbar.

Aufgaben der Dokumentation

- *Wissenssicherung*
  - Informationen aus den Köpfen holen
  - Informationen Anderen zugänglich machen
- *Kommunikation*
  - Reden allein genügt nicht
- *Sichtbar machen des Projektfortschritts*
  - Dokumente sind die greifbaren Resultate des Entwicklungsprozesses

Die Dokumentation dient der

- Wissenssicherung
  damit die Software auch nach längerer Zeit und von unterschiedlichen Personen weiterentwickelt werden kann
- Kommunikation
  um andere Mitarbeiter über die Ziele, die Art der Umsetzung und die Funktionalität zu informieren
- Darstellung des Projektfortschritts und der Steuerung
  um sicherzustellen, dass die Entwicklung den Vorgaben entspricht oder durch entsprechende steuernde Eingriffe zielgerichtete Maßnahmen ergriffen werden können.

Produktdokumentation – 1te

- Die Produktdokumente dokumentieren ...
  - ... die *Anforderungen* an das System
  - ... das *Konzept* der *Lösung*
  - ... die *Einzelheiten / Details* der *Lösung* (Entwürfe + Realisierung)
  - ... die *Montage / Aufbau* der einzelnen Komponenten (Integration + Installation)
  - ... die *Planung* der *Tests* und der *Abnahme*
  - ... die Phasen der *Entwicklung* + *Auslieferung*
  - ... die *Handhabung* des Systems (Benutzerdokumentation)

In der Produktdokumentation werden die wesentlichen Elemente dokumentiert, wie

- Anforderungen und Vorgaben
- Konzept und Lösung
- Umsetzungsschritte, Hintergründe
- Zusammenspiel der Komponenten und Informationsflüsse
- Prinzipieller Aufbau und Basisfunktionen
- ...

Produktdokumentation – 2te

- Anforderungsspezifikation
  Was von dem zu entwickelnden System *verlangt* wird
- Lösungskonzept
  Die *Architektur* der Lösung
  - Gliederung der Lösung in Komponenten
  - Kommunikation zwischen den Komponenten
  - Resourcen-Verteilung
- Detailentwurf + Code
  Lösungsdetails (Algorithmen + Datenstrukturen)
  - Entwürfe sind
    - Entweder separat vom Programm-Code dokumentiert
    - Oder in Form von Kommentaren im Programmcode integriert

In der Anforderungsspezifikation wird gemeinsam im Team zusammen mit Vertretern des Kunden festgelegt, was das System für Funktionen erfüllen muss und wie diese über die Benutzerschnittstelle nach außen geführt werden.

Die Anforderungsspezifikation ist nach Abnahme von Vertretern des Kunden und des Auftragnehmers zu genehmigen und zu unterzeichnen.

Produktdokumentation – 3te

- Testvorschriften
  - Tests für die einzelnen Komponenten
  - Tests für die einzelnen Integrationsschritte
  - Systemtest (nach letztem Integrationsschritt)
- Abnahmevorschriften
  - Abnahme = formaler Abschluss einer Entwicklung
  - Prüfung, ob das System die im Anforderungskatalog festgelegten Anforderungen erfüllt

Bereits im Rahmen der Festlegung der Anforderungen sind die Vorgaben für den Abnahmetest zusammen mit den Testdaten und dem Testrahmen festzuschreiben. Nur so kann sichergestellt werden, dass sich die Rahmenbedingungen nicht während der Realisierungsphase verändern.

Produktdokumentation – 4te

- Integrationsplan
  - Wie die einzeln fertig gestellten Komponenten zu einem, in einer Testumgebung lauffähigen Gesamtsystem integriert werden
- Installationsanleitung
  - Wie ein, auf der Ziel-Hardware lauffähiges System *konfiguriert* ...
  - ... und auf dieser Ziel-Hardware *installiert* wird
  - Welche *zusätzliche Fremd-Software* erforderlich ist (Middleware, Treiber, ...)

Neben den Testbedingungen sind auch die einzelnen Integrationschritte sowie die Rahmenbedingungen für deren Test festzulegen.

Insbesondere ist in der Produktdokumentation festzuhalten, welche Bedingungen das Zielsystem erfüllen muss (CPU. Speicherausbau, Plattenausbau, ...), damit die Anwendung den Benutzeranforderungen entspricht. Dies ist in einer Installationsanleitung, die alle Schritte für die Inbetriebnahme des Systems zu enthalten hat, zusammenzufassen.

Produktdokumentation – 5te

- Benutzerhandbuch
  Die *Bedienungsanleitung* für das System
  - Welche Funktionen stellt das System bereit?
  - Wie startet man das System?
  - Wie bedient man das System / einzelne Funktionen?

  Hinweis:
  Eingebettete Systeme haben kein Benutzerhandbuch, ihre Benutzung wird im übergeordneten System dokumentiert.
- Glossar
  Erklärt die verwendeten *Begriffe + Abkürzungen*
  - Hilfreich für Entwickler + Benutzer
  - Sollte in jedem größeren Entwicklungsprojekt (projektbegleitend) erstellt werden.

Im Benutzerhandbuch sind die einzelnen Handhabungsschritte und die verfügbaren Funktionen allgemeinverständlich zu erläutern.

Der Glossar erläutert die verwendeten Ausdrücke, die den Benutzern nicht von vornherein geläufig sind.

Projektdokumente

- **Projektplan**
  - Dokumentiert den geplanten *Projektablauf*
  - Stellt *SOLL* und *IST* gegenüber
  - Dokumentiert den *Projektablauf* + *Projektabschluß*
- **Qualitätsplan**
  - Projektspezifische *Vorgaben* für die *Qualität*
  - *Durchführungsbestimmungen* für den Ablauf
- **Projekt-Protokoll**
  - *Sammlung* aller *Schriftstücke* + *Berichte* aus dem Projekt, wie
    - Planungsunterlagen
    - Sitzungsprotokolle
    - Review- und Abnahme-Protokolle
    - Änderungsdokumente

Die Projektdokumentation dient der Sicherung der inneren Abläufe im Unternehmen und der Kommunikation gemachter Erfahrungen. Sie umfasst:

- Projektplan
  aufgeteilt in Arbeitspakete, Terminplanung, Ressourcenplanung, Kostenbudgetierung + -Planung bis hin zur Test- und Abnahmeplanung Die Steuerung und Fortschrittsplanung erfolgt auf dieser Grundlage.
- Qualitätsplan
  zur Sicherstellung der Qualitätsvorgaben
- Projekt-Protokoll
  als Sammlung aller Dokumente, die während des Projektablaufs für das Projekt relevant sind.

Dokumenten-Erstellung

- Parallel zur Entwicklung
  - schritthaltend
  - Permanent
- Keine „Hinterher"-Dokumentation
- Eventuell *Überarbeitung + Aktualisierung* der Dokumente bei *Projektabschluss* (Historie)
- Dokumente *genehmigen + unterzeichnen*
  - Ersteller
  - Projekt-Manager
  - Prüfer (Abnahme)

Die Dokumentation muß mit dem Projektfortschritt parallel entstehen, da sonst wesentliche Teile verloren gehen oder aber die Dokumentation nie vollständig gemacht wird.

Da im Laufe des Projekts viele Änderungen erfolgen, insbesondere auch an der Funktionalität und Spezifikation, sind diese Änderungen zeitnah einzuarbeiten und zu kennzeichnen.

Dokumentenverwaltung

- Dokumente unterliegen dem Konfigurations-Management (↳ Kapitel 10)
- Klassifikation
  - *Leichtes Finden* durch *geordnete Ablage*
  - + sinnvolle *Klassifikation*
- Freigabewesen
  - Nur *freigegebene Dokumente* sind *gültig*
  - *Änderungshistorie* + *Änderungsdokumente* mitführen
- Änderungswesen
  - Nur *aktuelle Dokumente* sind hilfreich
  - Aktuelle Dokumente sind *Voraussetzung für Wartbarkeit*

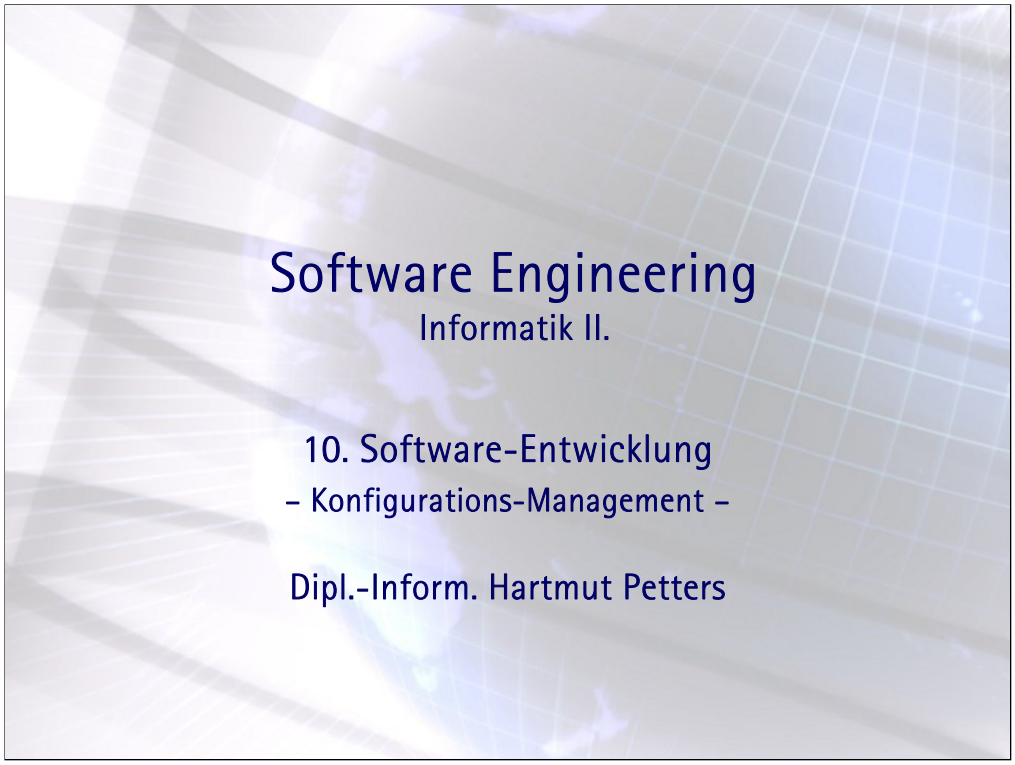
Alle Dokumente, die im Rahmen eines Projekts entstehen und für das Projekt relevant sind, müssen zusammen mit der Software verwaltet werden. Üblicherweise erfolgt dies im Rahmen eines Konfigurations-Management-Systems

Änderungen an den Vorgaben und Anforderungen unterliegen der Projektkontrolle, da durch sie der Ablauf des Projekts nachhaltig gestört werden kann. Deswegen ist für diese Änderungen ein abgesichertes Änderungsmanagement + Freigabewesen relevant. Die Änderungen müssen als Historie mitgeführt werden, so dass die Änderungen jederzeit nachvollziehbar sind.

Zusammenfassung

- **Dokumentation ist wichtig / unabdingbar**
  - Fortschrittskontrolle
  - Maßeinheit für Erreichtes
  - Basis für sinnvolle Kommunikation
  - Wissens-Absicherung
  - Unterliegt dem Konfigurations-Management
- **Dokumentation umfasst**
  - Anforderungspezifikation
  - Lösungskonzept, Detailentwurf + Code
  - Test- und Abnahme-Vorschriften
  - Integrationskonzept + Installationsanleitung
  - Benutzerhandbuch + Glossar
  - Projektplan / Qualitätsplan / Projekt-Protokoll

Dokumentation ist im Projekt extrem wichtig und darf nicht vernachlässigt werden.



Unter Konfigurations-Management wird das Management-System verstanden, dass die Konfiguration des Software-Systems zu jedem Zeitpunkt sicherstellt und so Fehlermeldungen früherer Versionen nachvollziehbar macht.

Definitionen

- Software-Konfigurations-Management
  (software configuration management)
  Die Gesamtheit aller Verfahren + Maßnahmen zur eindeutigen *Kennzeichnung* der Konfiguration eines Software-Systems mit dem Zweck, den *Aufbau* und alle *Änderungen* dieser Konfiguration systematisch zu *überwachen*, die *Konsistenz* des Software-Systems *sicherzustellen* und die Möglichkeit der *Rückverfolgung* anzubieten.
- Software-Konfiguration
  Eine Menge zusammenpassender Software-Einheiten
- Software-Einheit
  (software configuration item)
  Der *kleinste*, im Rahmen des Konfigurations-Managements als *atomar* behandelte *Baustein* einer Konfiguration.
  - Als Ganzes *registriert, freigegeben* oder *geändert*
  - Zum Beispiel Programm-Module und Dokumente

Konfigurations-Management ermöglicht die konsistente Verwaltung aller Elemente eines komplexen Software-Projekts über die Zeit, so dass die jede Version, die freigegeben wurde wieder rekonstruierbar ist.

Dabei umfasst das Konfigurations-Management alle Dokumente des Projekts einschließlich der Produkt- und Projektdokumentation.

Kennzeichnung von Software-Einheiten

- Software-Einheiten haben eine eindeutige Kennzeichnung
- Besteht aus einem *Namen* und einer *Versionsnummer*
- Kann weitere Informationen enthalten, zum Beispiel Name des Systems oder Teilsystems
- Die Identität einer Software-Einheit ist feststellbar, z.B. mit *Prüfsummen*

Für das Konfigurations-Management wird jede einzelne Komponente mit einer eindeutigen Kennzeichnung versehen.

Registrierung + Verwaltung

- Registrierung und Verwaltung der Software-Einheiten durch Software-Bibliothekar
- Pro Einheit mehrere Versionen möglich
- Im einfachsten Fall – aufsteigende Versionsnummern
- Im allgemeinen Fall – Revisionen (aufsteigend) und Varianten (parallel)
- Aktionen zur Verwaltung
  - „Check-In" einer Einheit – Übernahme in die Verwaltung („Electronic Vault")
  - „Check-Out" einer Einheit – Bezug zur Änderung
  - „Code-Freeze" – Festschreibung einer konsistenten Version

Nummer	Name	Typ	Ver	Prüfsumme	Status
...					
LOG 0021	Materialwesen	EntwDok	02	0873451-2	freigegeben
LOG 0027	Stückliste	Prog	03	0372538-1	freigegeben
LOG 0028	Verwendungsnachweis	Prog	02	0576927-6	in Prüfung
...					

Nur registrierte Komponenten können in ein Produkt und/oder Projekt eingebunden werden.

Konfiguration + Release

- Release
  Eine *konsistente* Menge von Software-Einheiten, die *gemeinsam* zur Benutzung *freigegeben* werden
- Dient vor allem
  - Zur *Auslieferung* von Software-Produkten an *Kunden*
  - Zur periodischen *Lieferung* von *Nachträgen* + *Verbesserungen*
- Beantwortet u.a. folgende *Fragen*
  - Welche Software-Einheiten gehören zu einer Konfiguration?
  - Wie hängen die Einheiten voneinander ab?
  - Wie wird ein auslieferbares System generiert?
  - Welche Einheiten gehörten zur Version, die am12. Oktober 1999 ausgeliefert wurde?
  - Welche Änderungen wurden bei einer Einheit aus dieser Version bis jetzt gemacht? Änderungsanträge + Änderungsfreigaben.
  - ... u.v.m.

Durch das Konfigurations-Management kann jedes einzelne Software-Release jederzeit rekonstruiert werden.

Änderungs- und Freigabewesen

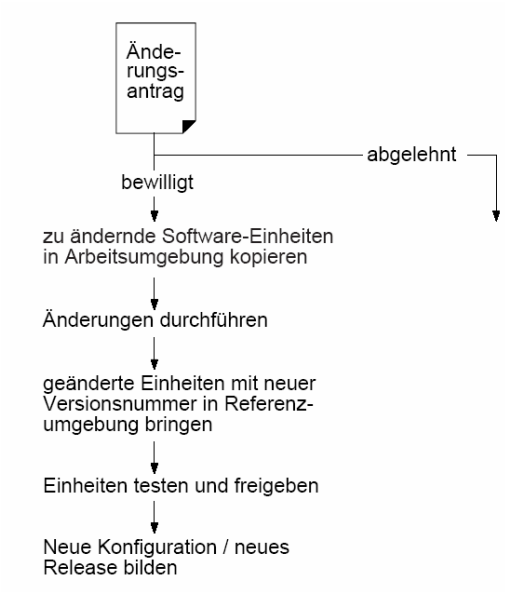
- Getrennte Umgebungen für
  - Entwicklung (*Arbeitsumgebung*)
  - Verwaltung (*Referenzumgebung*)
  - Test (*Testumgebung*)
  - Operativen Einsatz (*Produktionsumgebung*)
- Freie Änderungen nur in der *Arbeitsumgebung*
- *Strikt festgelegte Änderungsprozesse* für Software-Einheiten in der *Referenzumgebung*
- *Änderungen* in der *Produktionsumgebung* sind verboten
- Änderungsstand einer Software-Einheit jederzeit *nachweisbar*
- Definierter *Prozess* für den Ablauf von Änderungen, deren Freigabe und Übernahme in Referenzumgebung

Änderungen an einer freigegebenen Version/Release sind generell nicht möglich.

Sind Änderungen notwendig, so kann dies nur durch die Generierung eines neuen Releases erfolgen, dass auf Kompatibilität zu den anderen Elementen überprüft werden muss.

Alle Änderungen an den Elementen werden in einer Änderungs-Historie mitdokumentiert.

Prozess für den Ablauf einer Änderung



Der Ablauf für die Durchführung von Änderungen ist fest ald „Änderungsprozess" vorgegeben.

Problem- / Fehlererfassung + Dokumentation

- Systematische Behandlung von Kundenproblemen
- Grundlage: organisiertes Problem-Meldewesen
- Problem-/Fehlermelde-Formular
- Geordneter Bearbeitungsablauf (Problembearbeitungsprozess)
  - Registrierung eingegangener Meldungen
  - Analyse der Meldungen
  - Klassifizierung der Meldung (A / B / C)
  - Klasse A + B: „work-around" möglich?
  - Vorläufige Antwort (innerhalb von 24 Stunden)
  - Problembehebung + Dokumentation
  - Abschließende Antwort
  - Abschluss + Ablage der Problemmeldung
  - Auslieferung eines „Patches" oder eines neuen „Releases"

Zur geordneten Abwicklung entsprechender Fehlermeldungen und/oder Verbesserungsvorschlägen sind diese Meldungen an einer zentralen Stelle zu erfassen und an die entsprechenden Entscheidungsgremien weiterzuleiten. Dies erfolgt üblicherweise mit einem so genannten „Trouble-Ticket-System", in dem die Meldungen erfasst, weitergeleitet und bis zur Bearbeitung verfolgt werden.

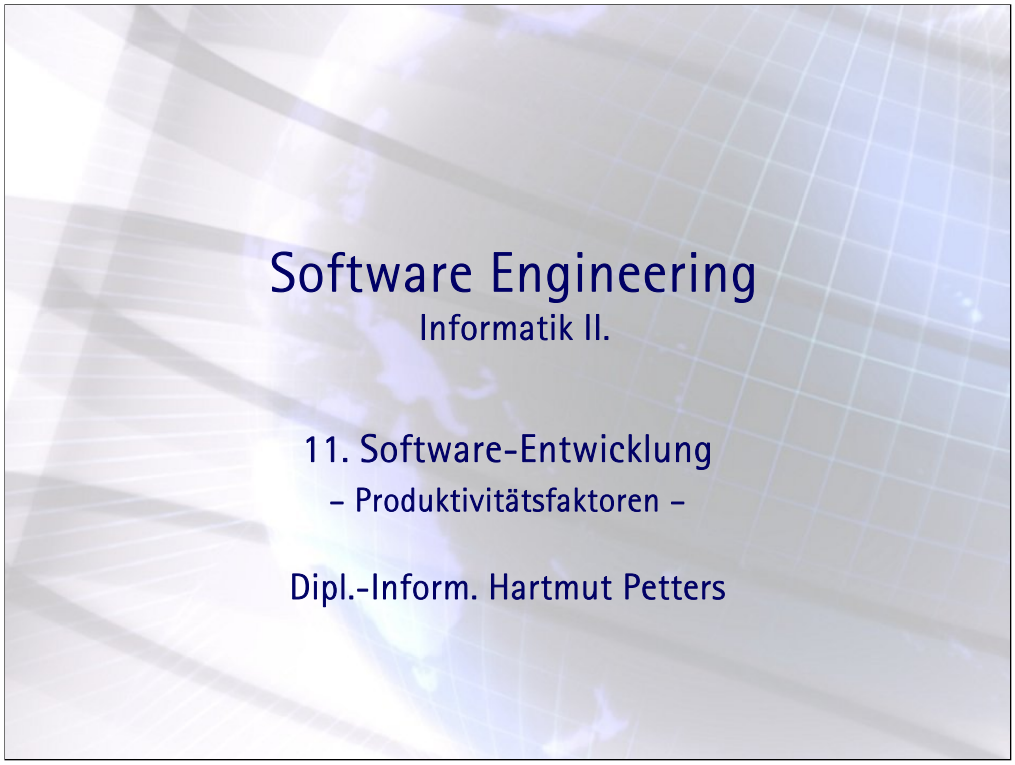
Fehler oder sonstige Verbesserungsvorschläge werden dabei i.d.R. in 3 Klassen (A, B, C) eingeteilt, wobei üblicherweise folgende oder eine ähnliche Bedeutung zugrunde gelegt wird:

A. Fehler, der den Systemeinsatz in Frage stellt und sofort behoben werden muss
B. Fehler, der durch einen "Work Around" kurzfristig umgangen werden kann, eine Fehlerbehebung erfolgt mit dem nächsten Release
C. „Nice-to-Have" Forderung, die keine funktionelle Beeinträchtigung darstellt, sondern nur Wünsche ohne Sysembeeinträchtigung darstellt.

Zusammenfassung

- **Konfigurations-Management ist das konsistente Management der einzelnen Software-Einheiten**
- **Jede Einheit ist eindeutig gekennzeichnet und über dem gesamten Lebenszyklus hinweg nachvollziehbar**
- **Konfigurations-Management beantwortet Fragen wie**
  - Welche Software-Einheiten gehören zu einer Konfiguration?
  - Wie hängen die Einheiten voneinander ab?
  - Wie wird ein auslieferbares System generiert?
  - Welche Einheiten gehörten zur Version, die am12. Oktober 1999 ausgeliefert wurde?
  - Welche Änderungen wurden bei einer Einheit aus dieser Version bis jetzt gemacht? Änderungsanträge + Änderungsfreigaben.
- **Strikt reglementiertes Änderungswesen stellt durch vorgegebene Prozesse die Wartbarkeit der Software sicher, so dass der Änderungsstand jederzeit nachweisbar ist.**

Bei komplexen Software-Systemen ist ein System zum Konfigurations-Management unbedingt einzusetzen, da ansonsten die im Markt vorhandenen Versionen nach einer bestimmten Zeit nicht mehr rekonstruierbar sind und das Änderungsmanagement nicht transparent ist.



Produktivität ist bei einer kommerziellen und professionellen Software-Entwicklung unabdingbar.

Einflussfaktoren

- Produktivität wird durch viele Faktoren beeinflusst
- Hauptfaktoren
  - Werkzeuge
  - Wiederverwendung / Beschaffung
  - Menschen
  - Methoden / Prozesse

Die Produktivität im Software-Erstellungsprozess wird beeinflusst durch

- Die eingesetzten Werkzeuge
- Software. Die zugekauft oder selbst zum Zweck der Wiederverwendung erstellt wurde
- Durch die Kompetenz und die Leistungsfähigkeit der eingesetzten Mitarbeiter
- Durch die zugrunde liegenden Methoden und Abläufe, die zur Software-Erstellung eingesetzt werden.

Was Werkzeuge leisten

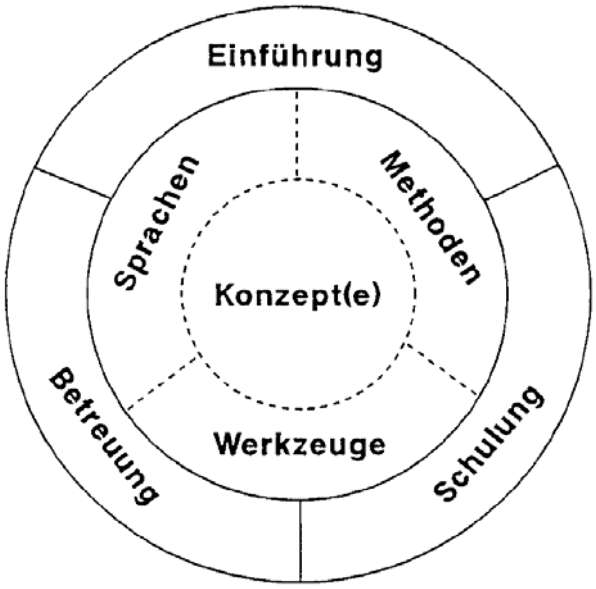
- Entlastung von *Routineaufgaben*
- Bearbeiten von *Sprachen*
- Unterstützen den Einsatz von *Methoden*
- Vereinfachen + Management von *Änderungen*

- Aber
  - Werkzeuge sind *keine Wunderwaffen*
  - *Keine* Produktivitätssteigerung um *Größenordnungen*
  - Ersetzen nicht eigenes *Denken* und *sorgfältiges Arbeiten + Disziplin!*
  - Machen das *Qualitäts-Management* nicht überflüssig

Werkzeuge können im Software-Erstellungsprozess nur für Routineaufgaben eingesetzt werden. Die kreative Phase des Software-Design muss vom Menschen getragen werden. Deswegen sind die Ergebnisse der Erstellung nur so gut, wie die Kompetenz und Leistungsfähigkeit der eingesetzten Mitarbeiter, die einerseits den Software-Entwurf machen und andererseits die Werkzeuge einsetzen.

Generell entlasten die Werkzeuge und der professionelle Einsatz dieser die Mitarbeiter von lästigen Routineaufgaben.

CASE (Computer Aided Software Engineering)



Werkzeuge bei der Software-Erstellung werden unter dem Begriff CASE (Computer Aided Software Engineering) zusammengefasst, die zusammen mit den Methoden und den Programmiersprachen um die prinzipiellen Konzepte angesiedelt sind. Dabei ist zu beachten, dass die Werkzeuge nur dann effizient eingesetzt werden können, wenn diese entsprechend eingeführt und geschult wurden und bei Problemen durch eine kompetente Betreuung unterstütz werden. Der professionelle Einsatz der Werkzeuge ist nur dann möglich, wenn die Mitarbeiter die Methoden und die Funktionen der Werkzeuge verstehen.

Klassifikation von Werkzeugen

- Editoren
- Spezifikations- und Entwurfs-Systeme
- Programm-Entwurfs-Systeme
- Compiler, Browser und Programmierumgebungen
- Programm-Generatoren
- Mess- und Test-Werkzeuge
- Konfigurationsverwaltung

Die Werkzeuge werden abhängig von ihrer Funktion in verschiedene Klassen eingeteilt.

Produktivitätsgewinn durch Werkzeuge

- *Substanzielle Produktivitäts- + Qualitäts-Steigerungen* sind durch Werkzeuge realisierbar!
- Aber
  - Bei der *Einführung* sinkt zuerst die *Produktivität*
    - Schulung
    - Lernkurve / Eingewöhnung
    - Verlagerung von Aufwendungen
- Der Gewinn – die Produktivitätssteigerung kommt erst *mittelfristig*
  - ↳ Werkzeug-Einführung ist eine Investition!

Beim Einsatz hochkomplexer Werkzeuge ist zu beachten, dass aufgrund der Lernkurve in der Einarbeitungszeit die Produktivität stark eingeschränkt sein kann, so dass der Einsatz solcher Werkzeuge generell als Investition in die Zukunft und die Qualität der Produkterstellung betrachtet werden sollte.

Beim Einsatz in einem Projekt ist dies bereits in der Planung zu berücksichtigen, da ansonsten die Planung gefährdet ist.

Planung des Werkzeugeinsatzes

- Was soll unterstützt werden?
- Wie wirtschaftlich ist der Einsatz?
- Welche Entwicklungskonzepte (Methoden, Sprachen) werden eingesetzt?
- Ist die Schulung sichergestellt?
- Wie sieht die Einführungsstrategie aus?
- Ist die Betreuung (im Betrieb) sichergestellt?

Auch der Werkzeugeinsatz sollte geplant werden, damit keine Überraschungen eintreten und die Ziele erreichbar sind.

Rolle der Menschen im Software-Engineering

- Software wird von Menschen gemacht!

- Die Software-Leute sind ein entscheidender Produktivitätsfaktor
  - Können / Kompetenz
  - Motivation
  - Arbeitsumfeld
  - Tagesform

Software wird von den Team-Mitgliedern im Projektteam gemacht und sind somit der Schlüssel für die Qualität des Produkts und die Planungssicherheit.

Werkzeuge können dabei nur unterstützende Funktionen übernehmen.

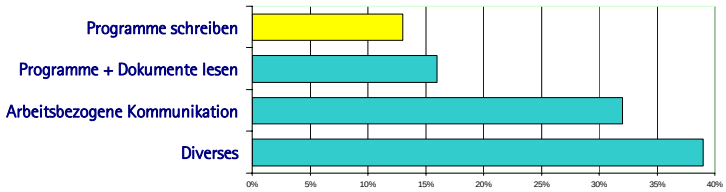
Gesetzmäßigkeiten bei Software-Leuten – 1te

- Produktivität
  - Enorme Schwankungsbandbreiten – bis zu 20:1
  - Selbst bei Gruppen noch Schwankungen bis 4:1
- Disponibilität
  - Personalbestände sind nur langsam auf- und abbaubar
  - Zu jedem Aufwand eine optimale Personenzahl
  - Das Aufstocken des Personalbestands in einem verspäteten Projekt führt zu noch mehr Verzögerungen (Gesetz von Brooks)

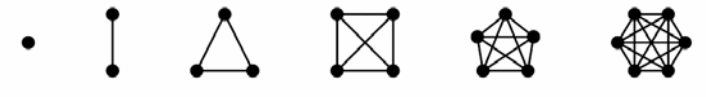
Gerade bei der Software-Entwicklung, die sehr stark auf die Kreativität der Entwickler angewiesen ist, ist aufgrund der Kompetenz und der Tagesform eine hohe Schwankungsbandbreite bis hin zu einem Faktor 20:1 möglich. Selbst bei guten und eingespielten Teams kann es dabei noch eine Schwankungsbandbreite von 4:1 geben.

Gesetzmäßigkeiten bei Software-Leuten – 2te

- Arbeitsverteilung
  - Programmierer schreiben nicht nur Programme



- Gruppengroße
  - Nicht produktiver Aufwand wächst überproportional zur Gruppengröße



18.1.2004 90 © by Hartmut Petters

Wesentliche Einflussfaktoren sind dabei

- Die Arbeitsverteilung und die unproduktiven Arbeiten
- Die Kommunikations-Anforderungen im Team

Der Einfluss der Arbeitsumgebung

- Ausbildung
  - *Gute Leute einstellen*
    - 1er stellen 1er ein
    - 2er stellen 3er ein
  - *Leute besser machen*
    - Fortbildung
    - Geeignete Gruppenprozesse / Team-Bildung
- Arbeitsplätze, an denen gearbeitet werden kann
  - Genug *Platz*
  - *Technisch adäquat* ausgerüstet
  - *Wenig Störungen*
  - *Kommunikations-Kultur*

18.1.2004 91

Wesentlichen Einfluss hat die Kompetenz der Mitarbeiter. Dabei kann davon ausgegangen werden, dass gute Mitarbeiter auch gute Mitarbeiter ins Team aufnehmen, da diese sich gegenseitig ergänzen und der Neidfaktor faktisch nicht vorhanden sein sollte.

Mittelmäßig qualifizierte Mitarbeiter werden beim Teamaufbau darauf achten, dass keine hochqualifizierten Mitarbeiter hinzugezogen werden, da diese die Qualifikations-Unterschiede transparent machen würden.

Bereits in der Planungsphase ist es wichtig, das Team auf die Aufgabe vorzubereiten und entsprechend zu schulen, mit den Methoden vertraut zu machen.

Software-Management / Software-Kultur

- Realistische Planung
- Gegenseitiges Vertrauen
- Informationskultur
- Schaffung einer Software-Kultur
  - Wir sind kompetent, aber wir können nicht alles.
  - Wir sind schnell, aber wir versuchen nicht uns selbst zu überholen oder die Lichtgeschwindigkeit zu ändern
  - Wir tun die Dinge von Anfang an richtig
  - Wir haben Spaß an professioneller Arbeit
  - Entscheidungen so spät wie möglich, aber so früh wie nötig treffen.

Wesentlichen Einfluss auf die erfolgreiche Umsetzung hat vor allem auch eine realistische Planung, die von allen Team-Mitgliedern voll getragen wird.

Zusammenfassung

- Produktivität wird beeinflusst durch
  - Die *Menschen*
  - Die *Arbeitsumgebung*
  - Die *Arbeitskultur*
- Werkzeuge müssen
  - *zweckdienlich* sein
  - *einsetzbar* sein
    - ↳ die Bedienung *bekannt und vertraut*
- Produktivität wird erreicht durch
  - *Kontinuität*
  - *Disziplin*
  - *Professionalität*

Produktivität kann gefördert werden, hängt aber stark von den Mitarbeitern und deren Kompetenz ab.