

Kapitel 1: Prinzipien des Software Engineerings

Boehms 7 Prozessprinzipien:

- 1) Manage using a phased life-cycle plan**
 - Projekt planen
 - Projektplan erstellen
 - Geeignetes Prozessmodell zugrunde legen
- 2) Perform continuous validation**
 - Frühestmöglich Fehler finden
 - Risiken verfolgen
 - Reviews, Tests, Prototypen etc verwenden
- 3) Maintain disciplined product control**
 - Konfigurationsmanagement
 - Identifikation
 - Konfigurationen verwalten
- 4) Use modern programming practises**
 - Programmiersprachen
 - Entwicklungsmethoden
- 5) Maintain clear accountability for results**
 - Aufträge, Verantwortlichkeiten und Ressourcen klar zuweisen, durch ganze Organisation durch Meilensteine
- 6) Use better and fewer people**
 - Fördern und ausbilden
 - Win-win situation anschaffen: Die richtigen Leute am richtigen Ort
- 7) Maintain a commitment to improve the process**
 - Messen
 - Analysieren, Auswerten, Handeln

Regel 8 von Glinz und Schauer

- 8) Reuse or buy Software where you can**
 - Kein falscher Entwicklerstolz ("Not invented here" Syndrom)
 - In alle Prozesse integrieren

Produktprinzipien von Glinz

- Strukturen und Abstraktion
- Modularität
- Information Hiding
- Schnittstellen und Verträge
- Qualität
- Nutzung von Vorhandenem

Kapitel 2: Systematisches Programmieren

Regeln für Namensgebung

- Selbsterklärende Namen!
Beispiele: Variablen: nextState; Methoden: PrintPage etc
- Den gleichen Namen nicht zweimal in einem Gültigkeitsbereich verwenden
- Generell: Sich an Codierrichtlinien halten, Konventionen konsequent durchhalten

Dokumentieren von Programmen

- Dokumentation und Code müssen übereinstimmen
- Kein Nachbeten des Codes
- Falschen Code kann man nicht durch Dokumentation richtig machen

Programmstruktur

Geschlossene Ablaufkonstrukte, GOTO

- Alle sequentiellen Programme können aus den Grundelementen Sequenz, Alternative und Iteration erstellt werden
- Alle Konstrukte haben genau je einen Eintritts- und Ausgangspunkt, sind also in sich geschlossen
- GOTO bricht bei unbedachter Anwendung Übereinstimmung zwischen statischer und dynamischer Struktur (Dijkstra)
→ GOTO nur bei Erhaltung geschlossener Ablaufkonstrukte verwenden
- GOTO kann ein Programm aber auch vereinfachen
→ oft komplizierte Schleifenbedingungen zur Vermeidung von GOTO notwendig

Iteration

- Wiederholte Ausführung einer Gruppe von Anweisungen in einer Schleife
- Vorwärtsberechnung: Resultat wird typisch inkrementell aufgebaut
- Schleife muss explizit gesteuert werden: Initialisierung, Abbruchbedingung, Fortschaltung (letzteres nur bei Zählschleifen)
- Konstruktion Schleife:
 - a) Schleifenkörper

- b) Schleifeninvariante (Prädikat, das nach jeder Prüfung der Abbruchbedingung wahr ist)
 - c) Aus Schleifenvariante ableiten: Initialisierung Abbruchbedingung und Fortschaltung
 - d) Prüfen, ob Schleife terminiert
- Verifikation von Schleifen kann mit Hilfe von Schleifeninvarianten durchgeführt werden.
Es seien

S: Schleife

N: Prädikat, das das erwartete Ergebnis von S beschreibt

b: Abbruchbedingung von S

Inv: Schleifeninvariante von S

Falls (i) $V \rightarrow \text{Inv}$ (ii) $(\text{Inv} + \neg b) \rightarrow N$ (iii) S terminiert ist S korrekt.

- Abweisende vs. Annehmende Schleifen

Abweisende (DO-WHILE) prüfen Abbruchbedingung vor dem Durchlauf durch den Schleifenkörper, also 0 (wenn Bedingung nicht erfüllt ist) bis n mal.

Annehmende (REPEAT-UNTIL) prüfen Abbruchbedingung erst nach dem Durchlauf durch den Schleifenkörper, also mindestens einmal, auch wenn Schleifenbedingung a priori falsch ist.

→ Abweisende sicherer als annehmende

Rekursion

- Mehrfache Ausführung von Anweisungen durch wiederholten Selbstaufufruf
 - Rückwärtsberechnung. Resultat wird durch rekursiven Abstieg gewonnen
 - Steuerung ist implizit: Nur Reduktion und Verankerung müssen sichergestellt werden
 - Jede Rekursion kann in eine Iteration verwandelt werden und umgekehrt
 - Fazit
- ☺ Rekursive Lösungen einfacher und kürzer als iterative
- ☹ Rekursion gedanklich schwieriger nachzuvollziehen als Iteration

Fallunterscheidung

- Vorsicht beim Aufbauen verschachtelter IF-Anweisungen
- ELSE IF besser als THEN IF

Prozeduren und Methoden

- Jede Prozedur / Methode realisiert eine Funktion oder in sich geschlossene Teilfunktion (funktionale Kohäsion)
- **Das** Strukturierungsmittel für Programmierung im Kleinen
- Unterprogramm
 - Benanntes, abgegrenztes Unterprogramm, das unter seinem Namen aufrufbar ist.
 - Bei Aufruf geht die Steuerung ins Unterprogramm und kehrt nach Ausführung dessen an Aufrufstelle zurück.
 - Parameterübergabe mit „call by reference“
- Prozedur
 - Ist ein Unterprogramm mit eigenem Namensraum und Parametersetzung
 - Aufruf als Anweisung
 - Parameterübergabe mit „call by value“, „call by reference“ oder „call by name“
- Methode
 - Wie Prozedur, aber mit dynamischer Bindung an Aufrufer zur Laufzeit
 - Parameterübergabe zum Teil nur mit „call by value“
- Makro
 - Benanntes, abgegrenztes Programmstück, das unter seinem Namen referenzierbar ist
 - Bei Referenzierung wird der Makrokörper vom Übersetzer an der Referenzstelle in den Code kopiert

Konstruktion guter Programme

Guter Entwurf

- Zusammenpassende Algorithmen und Datenstrukturen
- Geschlossene Konstrukte

Effizient und Korrekt

- Entwurf vollständig und korrekt umgesetzt

Lesbar, verifizierbar, änderbar

- Lesbarkeit vor Schreibeffizienz
- Struktur, Namengebung, Kommentar als Elemente der Dokumentierung

Verwendbar

- Schnittstellen definiert und dokumentiert
- Geregelte Zuständigkeiten und Verantwortlichkeiten (Geheimnisprinzip)

Sicher

- Defensives Programmieren
- Alle Voraussetzungen dokumentiert
- Verträge einhalten

Optimierung

Jackson Regeln:

- 1) Tu es nicht! → Code nie auf Verdacht optimieren
- 2) Wenn du es doch tun musst, dann tu es später! → Zuerst messen, Flaschenhalse erkennen, dann ggf. lokal optimieren

Glinz Regel:

- 3) Tu es vorher! (Erst denken, dann codieren) → Wahl guter Datenstrukturen und Algorithmen erspart Optimieren

3 Systematisches Entwerfen

3.1 Modulkonzepte

Modul: Eine benannte, klar abgegrenzte Komponente eines Systems.

3.1.1 Eigenschaften einer guten Modularisierung

- Kapselnde Dekomposition: Jeder Teil kann einzeln und mit möglichst wenig Kenntnisse über das Ganze und das Ganze ohne Detailkenntnisse verstanden werden.
- Verwendbarkeit: Ein Modul bildet eine Einheit bzgl. (Wieder-)Verwendung. Für V. braucht man inneren Aufbau nicht zu kennen. Leistungsangebot für Dritte in Form einer Schnittstelle. Module können durch Dritte zur Komposition von Systemen verwendet werden.
- Geschlossenheit und Lokalität: Jeder Modul ist eine in sich geschlossene Einheit. Änderungen im inneren Aufbau (nicht Schnittstelle) haben keine Rückwirkungen auf das übrige System. Die Korrektheit ist für jeder Modul prüfbar.

3.1.2 Modularisierungsarten

- **Strukturorientierte** Modularisierung: **Modularisierungskriterium:** Namensraum, Übersetzungseinheit; **Bspe.** Subroutinen in FORTRAN; **Güte der Modularisierung:** zufällig
- **Funktionsorientierte** Modularisierung: **M-Krit:** Jeder Modul berechnet eine Funktion; **Bsp.:** Structured Design; **Güte der M.:** gut für rein funktionale, zustandsfreie Probleme, sonst schlecht.
- **Datenorientierte** Modularisierung: **M-Krit:** Modul fasst eine Datenstruktur und alle darauf möglichen Operationen zusammen; **Bsp.:** Abstrakter Datentyp (ADT); **Güte der M.:** gut; **Problem:** ADT sind streng disjunkt, Gemeinsamkeiten können nicht zusammengefasst werden.
- **Objektorientierte** Modularisierung: **M-Krit:** Modul repräsentiert Objekt des Problembereichs; **Bsp:** Klassen im OO-Entwurf; **Güte der M:** gut, wenn Klassen als ADT konzipiert. Schlecht, wenn Klassen offen konzipiert werden. **Vorteil:** extrem flexibel und ausdrucksmächtig
- **Komponentenorientierte** Modularisierung: **M-Krit:** gekapselte Menge zusammengehöriger Elemente, die eine gemeinsame Aufgabe lösen und als Einheit von Dritten verwendet werden; **Bsp.:** Werkzeugsatz zur Bearbeitung von Verbunddokumenten; **Güte der M:** sehr gut; **Vorteil:** Als geschlossene Einheit verwendbar; **Problem:** Schnittstellen müssen gut sein, weniger flexibel als Klassen.

3.1.3 Schnittstelle und Implementierung

- **Verwenbarkeit:** Schnittstelle muss nach aussen sichtbar und dokumentiert sein.
- **Geschlossenheit:** Modul nur über Schnittstelle zugänglich; Implementierung gegen aussen verborgen.
- **Vollständige Trennung von Schnittstelle und Implementierung wäre optimal.** Bsp:Modula(DEFINITION MODULE (Schnittstelle), IMPLEMENTATION MODULE)
- **Keine konsequente Trennung von Schnittstelle und Implementierung bei Klassendefinitionen in OO-Programmiersprachen**

3.1.4 Spezifikation der Leistungen eines Moduls

Notwendiges Min: Namen/Signaturen der verwendbaren Operationen, Typen, Konstanten und ggf. Variablen.

Besser: Zusätzlicher, erläuternder Kommentar (ein, zwei Sätze)

Noch besser: Rigorose, teilformale oder formale Spezifikation der Schnittstelle (vgl. Kap.3.2) (mit PRE, POST, Axiome, Pseudocode,...)

3.2 Schnittstelle und Verträge

3.2.1 Das Vertragsprinzip für die Definition von Schnittstellen

- Schnittstelle ist Vertrag zwischen Modul und Modulverwender
- Beschreibung mit Zusicherungen
 - o Voraussetzungen (precondition (PRE), requirements (REQUIRE))
 - o Ergebniszusicherungen (postconditions (POST))
 - o Invarianten (invariants)
 - o Verpflichtungen (obligations)
- Vertragserfüllung bedeutet:
 - o Verwender muss: Voraussetzungen erfüllen und Verpflichtungen einhalten
 - o Modul muss: Zusicherungen erfüllen, Invarianten garantieren --> unter Annahme der Vertragstreue des Modulverwenders
- Eigenschaften einer vertragsorientierten Schnittstellendefinition:
 - o Leichter lesbar als algebraische Spezifikation
 - o Präziser und eindeutiger als einfacher Kommentartext
 - o Voraussetzungen und Resultate klar formulierbar
 - o Zustandsvariablen aus dem Problembereich

3.2.2 Voraussetzungen und Ergebniszusicherungen

- Beschreiben die Operationen (Methoden) einer Schnittstelle
- Voraussetzungen:
 - o Beim Aufruf erfüllt durch Aufrufer
 - o Werden von der Implementierung nicht geprüft
- Ergebniszusicherungen:
 - o Beschreiben Effekte der Operation
 - o Durch jede Implementierung der Schnittstelle erfüllt.
 - o Voraussetzungen müssen aber erfüllt sein.
- Dargestellt durch: Prädikate, Fallunterscheidungen, natürliche Sprache, wenige Zustandsvariablen.

3.2.3 Invarianten

- Eigenschaften der Schnittstelle, die unter allen Operationen invariant sind.
- Spezifizieren Zusammenhänge zwischen Operationen
- Entlasten die Ergebniszusicherungen der Operationen
- Bsp: EinfachesKonto --> saldo immer gleich einbezahlte Beträge minus abgehobene Beträge

3.2.4 Verpflichtungen

- Pflichten, die der Aufrufer mit dem Aufruf einer Operation übernimmt
- Bsp: In einem Sperrprotokoll übernimmt jeder, der eine Sperre setzt, die Verpflichtung diese Sperre auch wieder freizugeben.

3.2.5 Voraussetzen oder Prüfen (mit Fehlermeldung)?

- Entwurfsentscheidung!
- Voraussetzen immer dann, wenn die Erfüllung dem Aufrufer zugemutet werden kann.

- Muss mit Falscheingaben gerechnet werden, dann Prüfen! (Benutzereingaben)
- Nur Prüfen, wenn sinnvolle Behandlung der Fehler möglich ist.

Voraussetzen und dennoch Prüfen: Defensives Programmieren

- o Problem: unnötige Fehlerquellen, behindern Verständlichkeit des Codes, Redundanz
- o Als abschaltbare Option verwendbar

Prüfen der Ergebnisse anstelle der Voraussetzungen

- o Fehlerbedingungen werden erzeugt bei falschen oder unzulässigen Ergebnissen
- o Aufrufer interpretiert diese und handelt danach
- o Nachteil: Umständlich, erschwert Lesbarkeit des Codes des Aufrufers
- o Vorteil: Aufrufer kennt Kontext besser: bessere Fehlermeldung möglich
- o Mit Ausnahmebehandlung lösen!

3.2.6 Ausnahmebehandlung

Ausnahmen prüfen und behandeln anstelle von Voraussetzungen

- Prüfen der Ergebnisse anstelle der Voraussetzungen
 - o Ausnahmen werden erzeugt bei falschen oder unzulässigen Ergebnissen
 - o Aufrufer behandelt Ausnahmen oder reicht sie an seinen eigenen Aufrufer weiter
 - o Nachteil: Nicht in allen Programmiersprachen verfügbar
 - o Vorteil: Code für Normal- und Ausnahmesituationen sauber trennbar, keine Zustandsvariablen zur Weitergabe von Prüfergebnissen erforderlich.
 - o Bsp: in Java mit Schlüsselworten „throws“ und „exception“

3.2.7 Dynamische Prüfung von Zusicherungen

- Geeignet formulierte Zusicherungen in Programmen sind maschinell prüfbar
- Mächtiges Mittel zur dynamischen Prüfung von Programmen
- In manchen Programmiersprachen (z.B. Eiffel) direkt programmierbar
- Sonst mit Hilfskonstrukten zu programmieren, in Java mit Ausnahmen

3.3 Zusammenarbeit

Die Art der Zusammenarbeit definiert wesentlich den Entwurfsstil.

Die Zusammenarbeit muss dokumentiert werden.

Formen der Zusammenarbeit:

3.3.1 Leistungserbringung

Hauptmotiv: Delegieren von Aufgaben (mehrere Leistungserbringer, eine Steuerkomponente)

Ausführung der Aufgabe:

- Systemzustand soll unverändert bleiben:
 - o Methode oder Funktionsprozedur f mit Parameterübergabe und Rückgabewert (Daten und Objekte)
 - o Nebenwirkungsfrei (gesamter (in)direkt beeinflussbarer Systemzustand von f soll invariant gelassen werden)
- Systemzustand kann (oder soll) verändert werden:
 - o Methode oder Prozedur p mit Parameterübergabe (Daten, Objekte und Operationen)
 - o Direkt veränderbar sind: Ausgabeparameter, Zustand des Moduls, der p enthält bzw. auf das p angewendet wird.
 - o Indirekt veränderbar sind: Alle Zustände von Elementen, die von Operationen veränderbar sind, an die p (direkt oder transitiv) Arbeit delegiert.

- Beliebige Nebenwirkungen möglich (alle Veränderungen in der Schnittstelle definieren)

3.3.2 Informationsaustausch

Information: Daten, Operationen oder Objekte

Hauptmotive:

a) Wertschöpfungskette oder Fließbandarbeit (Bringprinzip)

Weitergabe durch:

- **Eingabeparameter:**

- Wertparameterübergabe nebenwirkungsfrei, schwache Kopplung
- Operationen- und Objektübergabe mächtiger und flexibler, aber Nebenwirkungen, stärkere Kopplung

- gemeinsam genutzte **(teil)globale Variablen:** Nebenwirkungen, Synchronisationsprobleme, starke Kopplung

- A verändert direkt Variablen im von ihm referenzierten Objekt B: Kopplung sehr stark, vermeiden.

- A bringt die Info B durch einen **Vermittlungsdienst:** ermöglicht geographische Verteilung, völlige Entkopplung, Übertragbarkeit von Funktionen und Objekten kann stark eingeschränkt sein.

b) Kette von Händler (Holprinzip)

Holen durch:

- **Ausgabeparameter:**

- Wertparameterausgabe als Referenzparameter bei richtiger Verwendung nebenwirkungsfrei, sonst massive Nebenwirkungen möglich.
- Operationen- und Objektübergabe mächtiger und flexibler, aber Nebenwirkungen, stärkere Kopplung

- Gemeinsam genutzte **(teil)globale Variablen:** fast immer Nebenwirkungen und Synchronisationsprobleme, starke Kopplung.

- B liest direkt Attributwerte im von ihm **referenzierten** Objekt A: verwenden, wenn gelesene Attribute ausserhalb von A schreibgeschützt sind, und die Attributstruktur eine geringe Änderungswahrscheinlichkeit hat, sehr starke Kopplung

- B fordert über **Vermittlungsdienst** die Info an: ermöglicht geographische Verteilung, völlige Entkopplung, Übertragbarkeit von Funktionen und Objekten kann stark eingeschränkt sein.

c) Lieferverträge (Abonnementsprinzip)

Benachrichtigung durch:

- B abonniert durch Übergabe einer **Benachrichtigungsoperation** an A. A benachrichtigt und gibt Information mit oder B holt sie bei Benachrichtigung: Dient zur Trennung eng kooperierender Aufgaben in separate Module (Entkopplung)
- A und B setzen und fragen **(teil)globale Variablen** ab: aufwendig und fehlerträchtig, darum vermeiden.
- **Vermittlungsdienst:** ermöglicht geographische Verteilung, völlige Entkopplung, Übertragbarkeit von Funktionen und Objekten kann stark eingeschränkt sein.

3.3.3 Informationsteilhabe

Hauptmotiv: Komponenten sind gleichberechtigte Teilhaber an einer Menge von Information

a) gemeinsam genutzter Speicherbereich

- lesen und schreiben im gemeinsamen Speicherbereich: explizite Synchronisation erforderlich

- nutzen gemeinsame Informationen durch Aufruf von Operationen bzw. Anwendung von Methoden auf Objekte im gemeinsamen Speicherbereich: Synchronisation erfolgt direkt durch Operationen oder Methoden.

b) gemeinsames Informationsdepot (Repository)

- Vermittlungsdienst verwaltet Informationsdepot, Zugriff über Vermittlungsdienst: **Datenbanksystem** ist meist der Vermittlungsdienst.
- Arbeiten mit persistenten gemeinsamen Objekten.

3.3.4 Zusammenarbeit und Entwurstil

Beispiele von Zusammenarbeit in den verschiedenen Entwurfs- bzw. Architekturstilen:

- Funktionsorientierter Stil:
 - o Leistungserbringung mit statisch gebundenen Funktionen und Prozeduren
 - o Azyklische Aufrufhierarchie,
 - o Übergabe von Daten als Parameter
 - o Sekundär: Informationsteilhabe mit direktem Lesen/Schreiben gemeinsamer Speicherbereiche
- Datenorientierter Stil:
 - o Leistungserbringung mit statisch gebundenen Funktionen und Prozeduren
 - o Azyklische Aufrufhierarchie
 - o Übergabe von Daten als Parameter
 - o Informationsteilhabe über gemeinsame Speicher (gekapselt in ADT) oder über gemeinsames Informationsdepot
- Objektorientierter Stil:
 - o Alle Zusammenarbeitsformen möglich
 - o Leistungserbringung mit statisch oder dynamisch gebundenen Methoden
 - o Übergabe von Daten und Objekten
 - o Meistens nicht azyklisch
 - o Informationsaustausch in allen Formen; häufig nach Abonnementsprinzip
- Prozessorientierter Stil:
 - o Informationsaustausch nach Bring- oder Holprinzip mit Vermittler
 - o Informationsteilhabe mit gemeinsamen Speicherbereichen

3.3.5 Dokumentation der Zusammenarbeit im Entwurf

Durch Zusammenarbeit werden aus Modulen bzw. Komponenten Systeme

Darum ↗ Zentrale Entwurfsaufgabe

Der Dokumentationsbedarf hängt vom Grad der Unabhängigkeit der Komponenten ab

- Gemeinsam erstellte und vertriebene Komponenten
 - o Verzahnter Entwurf von Komponenten und Zusammenarbeit
 - o Bekannter Verwendungskontext jeder Komponente
 - o Entwerfende stimmen Schnittstellen und Zusammenarbeitsbedürfnisse ausfeinander ab.

↗ Dokumentation durch Nennung der verwendeten Komponenten in der Schnittstelle jeder Komponente und Übersicht mit einem Zusammenarbeitsdiagramm genügt.
- Separat erstellte und vertriebene Komponenten
 - o Verwendungskontext ist einzelner Komponente nicht klar
 - o Zusammenarbeit mit Dritten und Leistungsangebot der Komponente muss ausführlich und präzise dokumentiert sein, durch Leistungsschnittstelle und Bedarfsschnittstelle:
 - Welche Voraussetzungen darf die benötigte externe Operation höchstens machen

- Welche Ergebnisszusicherungen muss die benötigte Operation mindestens machen
- o Bei wechselseitigem Aufruf von Methoden muss noch Zusammenarbeitsprotokoll spezifiziert werden.

3.4 Spezialisierung von Schnittstellen; Schnittstellenvererbung

Generalisierungshierarchie wie bei Klassen

- **Steinbruch:** Teile einer bestehenden Schnittstelle werden übernommen, beliebig ergänzt und abgeändert.
- **Spezialisierung:** Sei S' eine Subschnittstelle von S . Die von S' offerierten Leistungen sind ein Spezialfall der von S offerierten Leistungen.
- **Substituierbarkeit:** Sei S' eine Subschnittstelle von S . Jede korrekte Implementierung von S' ist gleichzeitig auch eine korrekte Implementierung von S . Dementsprechend ist jede Implementierung von S durch eine beliebige (korrekte) Implementierung von S' ersetzbar.

Bedingungen für Subschnittstelle sind: keine stärkeren Voraussetzungen und keine schwächere oder irgendwie eingeschränkte Ergebnisse liefern als Schnittstelle selbst.

Andere Möglichkeit der Schnittstellenhierarchie ist die Benutzungshierarchie: Schnittstellen (und deren Implementierungen) benutzen andere Schnittstellen

Siehe auch Übungen.

C.Stettler <cs@rogatec.ch>

Grundlagen

Motivation

- zunehmende Abhängigkeit von Software
- Erwartung, das Software korrekt („richtig“) funktioniert
- Qualitätsmanagement nötig für Entwicklung guter Software

„richtig“ heisst

- gestellte Anforderungen definiert, an Problemstellung angepasst → Qualität definiert
- Software erfüllt alle Anforderungen → Qualität erfüllt

Software-Qualitätsmanagement

- allgemeingültige Prinzipien, Begriffe, Grundlagen, Normen
- spezifische Verfahren für Software-Qualitätsmanagement
- schwierig, weil Software immateriell und ganzer Aufwand in Entwicklung

Begriffe

- Software: Programme, Verfahren, verbundene Dokumentation und Daten für Betrieb eines Rechnersystems
- Qualität: Grad, in dem ein Satz von inhärenten Merkmalen gestellte Anforderungen erfüllt
- Anforderungen: festgelegte, vorausgesetzte oder verpflichtende Erwartung / Erfordernis
- inhärentes Merkmal: kennzeichnende Eigenschaft einer Einheit, welche diese aus sich selbst hat und die ihr nicht explizit zugeordnet ist
- Qualitätsmanagement: aufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität (Festlegung Qualitätspolitik, Qualitätsziele, Qualitätsplanung, Qualitätslenkung, Qualitätssicherung, Qualitätsverbesserung)

Qualität

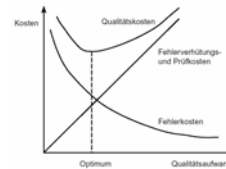
- Zielerfüllung: Ziele explizit festgelegt oder implizit durch gemeinsame Vorstellungen gegeben
- mehr als Zweckeignung / Kundenzufriedenheit
- kein absolutes Mass für die Güte einer Einheit → relativ
- muss definiert und geschaffen werden, entsteht nicht selbst
- bezug auf Produkte und zugehörige Prozesse / Projekte

Einsatz von Qualitätsmanagement

- Handwerk / historisch:
 - Qualitätsbewusstsein
 - tradierte Standards
 - direkte Rückkopplung
 - kein explizites Qualitätsmanagement
- Massenproduktion:
 - statistische Qualitätserforderungen → Anforderungen an Produktionsprozesse → Messen / Auswertungen → Rückkopplung auf Produktionsprozess → Produktqualität
- Software:
 - Individualprüfung mit schwieriger Messung / Prüfung
 - schwierige Rückkopplung auf Prozesse (bisher nur über Nebenprobleme)
 - nur Entwicklung, keine Produktion
 - keine tradierten Standards / anerkannte Qualitätsbegriffe

Qualitätskosten

- Qualität ist wirtschaftlich
- proportionale Fehlerverhütungskosten / Prüfkosten
- sinkende Fehlerkosten
- Qualitätskosten als U-Kurve → Minimum finden



Grundsätze Qualitätsmanagement

- Produktverantwortung und Qualitätsverantwortung untrennbar
- jede Person in seinem Arbeitsbereich persönlich für Qualität seiner Arbeit verantwortlich
- Ermittlung von Qualität an Spezialisten delegierbar
- Qualität muss erzeugt werden, ist nicht erprüfbar
- Beteiligte müssen über Qualität ihrer Arbeit informiert sein → Feedback
- schlechter Entwicklungsprozess fördert Entstehung schlechter Produkte

Qualitätsnormen

- ISO: Qualitätsbegriff orientiert an Erfüllung von Anforderungen, prozessorientiertes Qualitätsmanagement
- ISO 9000: Qualitätsmanagementsysteme – Grundlagen, Begriffe, Terminologie
- ISO 9001: Qualitätsmanagementsysteme – Anforderungen (Mindeststandards)
- ISO 9004: Qualitätsmanagementsysteme – Leitfaden zur Leistungsverbesserung (Verbesserung qualitätsrelevante Prozesse)

Elemente des Qualitätsmanagements

Qualitätspolitik

- übergeordnete Absichten/Ausrichtungen einer Organisation zur Qualität
- Festlegung Stellenwert von Qualität in Handeln und Auswirkungen auf Organisation, Kunden und Partner
- Festlegung der Unternehmensziele bez. Qualität
- Selbstverpflichtung zur Zielerreichung und Einhaltung der Arbeitsweisen / Verfahren
- schriftlich festgelegt, bekannt in ganzer Unternehmung
- von Geschäftsleitung festgelegt und aktiv durchgesetzt
- Totales Qualitätsmanagement (TQM)
 - Qualität als Unternehmensprinzip
 - Führungsmethode mit Kundenzufriedenheit als oberstes Ziel
 - Qualität im Mittelpunkt, alle Mitglieder eingebunden
 - übrige Unternehmensziele von Ziel Kundenzufriedenheit und verbundenen Qualitätsanforderungen abgeleitet

Qualitätsplanung

- Festlegung der Qualitätsziele und notwendigen Ausführungsprozesse
- Festlegung der zugehörigen Ressourcen zur Erfüllung der Qualitätsziele
- Festlegung der Qualitätsziele und quantifizierte Qualitätsanforderungen für jede Einheit → Qualität nicht absolut, Bezug auf Verwendungszweck
- kein Qualitätsmanagement ohne saubere, quantifizierte Spezifikation der Anforderungen

Qualitätslenkung

- Ausrichtung auf Erfüllung von Qualitätsanforderungen
- analytische Massnahmen (Prüfung → nachträglich)
 - Überprüfung Zwischen-/Endergebnisse, Fehler korrigieren (statisch: Review, statische Analyse, formale Programmverifikation; dynamisch: Testen)

- Überprüfung Einhaltung des Entwicklungsprozesses (Audits, Prozessbeurteilung, Prozessverbesserung)
- konstruktive Massnahmen (Lenkung → präventiv)
 - fehlervermeidende Prozesse definieren (Typprüfung, defensives Programmieren)
 - Integration Prüf-/Korrekturverfahren in Prozesse (Review von Dokumenten / Code)
 - Verwendung der Prüfergebnisse für Verbesserung/Optimierung des Prozesses
- keine Vorgehensweise für garantierte Erreichung der Qualitätsanforderungen für Software
 - konstruktive Massnahmen zum Halten des generellen Qualitätsniveau
 - analytische Massnahmen (Qualitätsprüfung) in allen Phase der Entwicklung → Mittel zur Sicherstellung der konkreten Qualitätsanforderungen an Software

Qualitätssicherung

- Ausrichtung auf Erzeugen von Vertrauen, dass Qualitätsanforderungen erfüllt werden
- regelmässige Überprüfung der Wirksamkeit des Qualitätsmanagements → Audits
- Publikation von qualitätsrelevanten Messgrössen
- Zertifizierung des Qualitätsmanagements
- Aktionsprogramme zur Verbesserung der Prozesse für Entwicklung, Pflege, Wartung

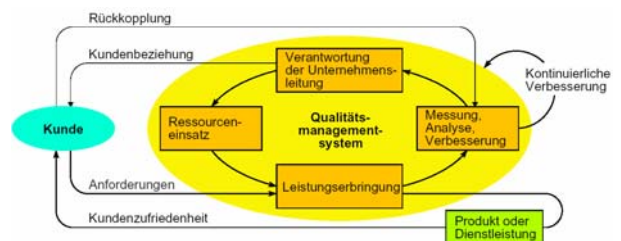
Qualitätsverbesserung

- Ausrichtung auf Erhöhung der Fähigkeit zur Erfüllung von Qualitätsanforderungen
- Behebung der gefundenen Mängel nur Symptombekämpfung, nicht immer möglich
- Modifikation in Entwicklungsprozess und Qualitätsmanagementsystem nötig
- Prozessverbesserung durch Auswertung von Fehlerursachen, Audits, Messungen

Qualitätsmanagementsystem

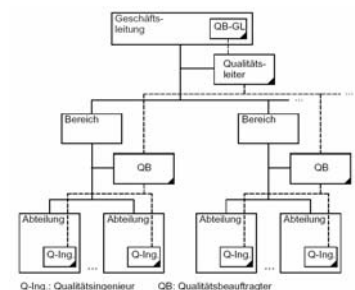
Allgemein

- Managementsystem zum Leiten / Lenken einer Organisation bezüglich Qualität
- Struktur, Verantwortlichkeiten, Mittel zur Verwirklichung des Qualitätsmanagements
- Orientierung an Kundenzufriedenheit
- prozessorientierte Organisation
- systematischer Ansatz



Aufbauorganisation

- Verankerung in Primärorganisation
 - alle Mitarbeiter in Qualitätsmanagementsystem integriert
- Sekundärorganisation aus Qualitätsfachleuten
 - Fachwissen über Qualitätsbelange/Qualitätsmassnahmen
 - Anbieten von Dienstleistungen in Qualität für Management und Entwicklung (Messung/Auswertung)
 - unabhängiger Berichtspfad bis in Geschäftsleitung
 - Verantwortlichkeit für Pflege, Weiterentwicklung, Verbesserung des Qualitätsmanagementsystems



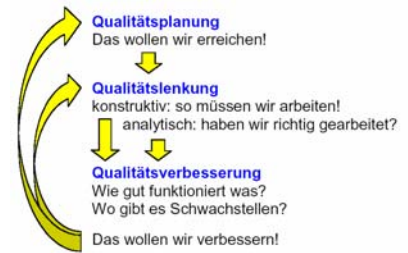
Ablauforganisation

- Festlegung der Kompetenzen, Verantwortlichkeiten, gegenseitigen Beziehungen
- nötig für alle qualitätsrelevanten Bereiche
- Beschreibung in Form von Prozessen → Prozessorganisation



Verfahren / Infrastruktur

- planen → lenken → verbessern



Dokumentation

- Dokumentation des Qualitätsmanagementsystems
- Qualitätsaufzeichnungen: Nachweise durchgeführter Qualitätsmassnahmen (Reviewberichte, Testprotokolle, Auditberichte)
- Aufbewahrung der Aufzeichnungen → Festlegen, wie lange und wo
 - Nachweis, dass Qualitätsanforderungen an Produkt erfüllt
 - Nachweis für Wirksamkeit des Qualitätsmanagements
- Identifikation und Zuordnung zum Produkt sicherstellen

Qualitätsverbesserung

- Qualitätsmanagementsystem planen
- Qualitätsmanagementsystem implementieren und betreiben
- Qualitätsmanagementsystem überprüfen, messen, auswerten
- Schwachstellen identifizieren, bewerten,
- Verbesserungspotential planen, umsetzen



Dokumentation

Aufgaben der Dokumentation

- Erfüllung von Kundenanforderungen
- Qualitätsverbesserung
- Schulung und geregelte Durchführung
- Wiederholbarkeit der Verfahren
- Nachweis durchgeführter Massnahmen
- Beurteilung der Wirksamkeit des Qualitätsmanagementsystems

Qualitätsbezogene Dokumentation

- Qualitätshandbuch: Dokumentation Qualitätsmanagementsystem
- Qualitätsmanagementplan: Dokumentation Qualitätsmanagement für spezifisches Projekt / Produkt
- Anforderungsspezifikation: Dokumentation der zu erfüllenden Anforderungen
- Verfahrens-/Arbeitsanweisungen: Beschreibung der Durchführung von Prozessen / Verfahren im Detail
- Leitfäden: Empfehlungen / Vorschläge für Vorgehensweisen
- Aufzeichnungen: Nachweis von ausgeübten Tätigkeiten / erzielten Ergebnissen

Qualitätshandbuch

- Band 1
 - Dokumentation Qualitätspolitik / Qualitätsorganisation
 - Überblick über Qualitätsmassnahmen
 - Abgabe an Kunden (auf Verlangen)
- Band 2
 - vertrauliche Version



- Dokumentation der genauen Verfahren / Massnahmen
- Dokumentation des Ablaufes der Prozesse
- Software-Entwicklungshandbuch
 - Regelung von speziellen Fragen der Software-Entwicklung

Qualitätsmanagementplan

- Inhalt
 - Zweck, Geltungsbereich
 - Prozessmodell
 - zu erstellende Dokumente
 - Werkzeuge
 - Prüfplan / Messplan (was, was, Verfahren, Aufzeichnungen)
 - Problemmeldewesen
 - Konfigurationsmanagement (Code, Dokumente)
 - Ablagesystem für Qualitätsaufzeichnungen
 - Schulung

Werkzeuge

Werkzeugbegriff

- Werkzeug: rechnergestützte Hilfsmittel für Entwicklung und Verwaltung von Software
- CASE: Computer Aided Software Engineering

Einsatz im Software-Qualitätsmanagements

- Automatisierung der Routine-Aufgaben in Prüfung und Verwaltung
- Beitrag zu Fehlervermeidung
- Unterstützung von Entwickler bei Prüfung, Fehlersuche, Fehlerkorrektur
- Senkung der Kosten der Qualitätsmassnahmen
- kein Ersatz / vollständiges Automatisieren des Qualitätsmanagements durch Werkzeuge

Werkzeuge zur Prüfung von Software

- Syntaxprüfer
 - Compiler, syntaxsensitive Editoren
- statische Prüfung
 - statische Analytoren
 - Einhaltung Codier-/Dokumentationsrichtlinien
 - Inkonsistenzen zwischen Darstellungsformen / Abstraktionsebenen
- Testunterstützung
 - Testtreiber/Teststumpf-Generatoren
 - Programmierinstrumentierer (Messung von Testüberdeckung)
 - Testauswerter (automatischer Vergleich Ergebnisse Testlauf mit Referenz)
- Debugger zur Suche unerkannter Fehler
- Programmierumgebung mit integrierten Prüfhilfsmittel

Fehlervermeidung durch Werkzeuge

- methodischen, werkzeugunterstütztes Vorgehen (z.B: Modellierung von Systemen → weniger Gefahr für grobe Fehler)
- zentrale Ablage von Entwicklungs-/Pflegetinformationen (durch Werkzeug)
 - Vermeidung von Inkonsistenzen / Lücken
 - Verringerung der Gefahr von Doppelspurigkeiten / Arbeit an gleicher Sache mit unterschiedlichen Informationen

Messwerkzeuge

- automatische Erhebung von Messgrößen in Programmen (Grösse, Anzahl Klassen, Anzahl Methoden)
- Unterstützung der Erhebung von Messgrößen in Projekten (Zeitaufschiebung, Terminverfolgung)

Werkzeuge für Konfigurationsmanagement

- Datenbanken zur Verwaltung von Software-Einheiten, Konfigurationen, Releases
- Dateivergleicher, Prüfsummenprogramme zur Identitätsbestimmung von Software-Einheiten
- Konfigurationsgeneratoren zur Erzeugung von Konfigurationen nach gegebenen Vorgaben (MAKE)
- Versionsverwalter zur effizienten Speicherung von Software-Einheiten in verschiedenen Versionen (CVS)
- Programmierumgebung zur Unterstützung getrennter Umgebungen für Entwicklung, Referent, Test, Produktion
- integrierte Konfigurationsmanagementsysteme

C.Stettler <cs@rogatec.ch>

Grundlagen

Probleme der Qualitätsplanung

- Qualität = Erreichen gesetzter Ziele
- Qualitätsplanung = Festlegen von Zielen = Spezifikation von Anforderungen
- keine Existenz von natürlichen Qualitäten

Wahl der Ziele

- bestimmen Aussehen des Produktes (bei Funktionalität und nichtfunktionalen Zielen)
- Ziele können sich konkurrenzieren
- Ziele ergeben sich nicht von selbst

Experiment von Weinberg

- identische funktionale Anforderungen, aber unterschiedliche nicht-funktionale Anforderungen (Erstellungsaufwand, Speicherbedarf, Klarheit, ...)
- Fokus auf ein Ziel erfolgt nicht automatisch zum Erreichen andere Ziele → konkurrenzierende Ziele

Qualitätsplanung / Qualitätslenkung

- Qualität ist planbar und lenkbar
- Qualitätsplanung: Festlegen von Zielen
- Qualitätslenkung: zielgerichtetes Arbeiten, Überprüfung der Zielerreichung
- Quantifizierung der Ziele und der Qualität nötig zur Messung der Zielerreichung

Qualitätsmessung

- Messen: interessierendes Merkmal quantitativ erfassen
- Merkmalswerte vergleichen, bewerten, auswerten
- Ziele der Qualitätsmessung
 - Qualität von Produkten / Prozessen lenken
 - Erfahrungen quantifizieren
 - Entscheidungsgrundlagen gewinnen
 - Prognosen stellen
- Ablauf einer Messung
 - Festlegung der interessierenden, zu messenden Merkmalen
 - Bestimmung der Merkmalseigenschaften aus Gegenstandsmenge, die von Mass berücksichtigt werden müssen
- Beispiel
 - Gegenstandsmenge: Programme
 - Merkmale: Grösse (Effizienz, Komplexität)
 - Eigenschaften von Merkmal Grösse: geordnet, additiv, vervielfachbar

Messtheorie

Modelle

- Merkmal messen: quantitatives Modell für zu messendes Merkmal bilden
- Eigenschaften des gemessenen Merkmals adäquat wiedergeben (inkl. Beziehungen, Operation)

- keine Verwendung von Beziehungen / Operationen, die in Original keine Entsprechung haben

Mass und Repräsentation

- Mass: Abbildung, die jedes Merkmal M aus Gegenstandsmenge D auf einen Messwert auf einer Skala S, so dass M und S strukturähnlich und homomorph sind
- Struktur des Originals muss sich in Modell wiederfinden (und umgekehrt)
- Strukturähnlichkeit: Beziehungen und Operationen entsprechen sich in Original und Modell
- Masse definieren oft intuitives Merkmal → keine eindeutige Lösung (z.B: Mass für Programmgröße: Codezeilen, Speicherbedarf, ...)

Skalen

- Nominalskala
 - nur Kategorisierung
 - keine Vergleichbarkeit ($<$, $>$), keine Verknüpfbarkeit
 - Relationen: $=$, \neq
 - beliebige injektive Funktion
 - Bsp: grün, gelb, blau
- Ordinalskala
 - Werte total geordnet
 - Werte vergleichbar
 - Relationen: $=$, \neq , $<$, $>$
 - Medianwert bestimmbar
 - streng monoton wachsende Funktion
 - Bsp: --, -, 0, +, ++
- Intervallskala
 - Werte total geordnet
 - Distanzen bestimmbar, sind additiv
 - willkürlichen Nullpunkt
 - Relationen: $=$, \neq , $<$, $>$, Distanz
 - Mittelwert, Standardabweichung bestimmbar
 - Lineartransformation
 - Bsp: Kalendertage
- Verhältnisskala
 - Werte total geordnet
 - Distanzen bestimmbar, sind additiv
 - absoluter Nullpunkt
 - Relationen: $=$, \neq , $<$, $>$, Distanz, Vielfaches, % (+,-)
 - Multiplikation aller Skalenwerte mit konstantem Faktor
 - Bsp: Franken
- Absolutskala
 - Skalenwerte sind absolute Größen
 - Relationen: $=$, \neq , $<$, $>$, Distanz, Vielfaches, % (+,-)
 - keine Transformation/Umrechnung in andere Skala möglich
 - Bsp: nicht-negative ganze Zahlen

Qualität von Massen

Eigenschaften von guten Massen

- Validität
 - Mass misst tatsächlich zu messendes Merkmal
- Aussagekraft
 - Messwerte sind sinnvoll interpretierbar

- Schärfe
 - verschieden wahrnehmbare Merkmale werde auf verschiedene Messwerte abgebildet
- Auswertbarkeit
 - Auswertungen sind möglich auf Messwerten (Statistik)
- Verfügbarkeit
 - Merkmal kann zum benötigten Zeitpunkt gemessen werden
 - Kosten der Messung
- Stabilität / Reproduzierbarkeit
 - Empfindlichkeit des Masses gegenüber Störungen
 - mehrfache Messungen des gleichen Merkmals liefern gleiches Resultat

Produktmasse

Allgemein

- Messen von quantitativen Software-Qualitätsmerkmalen
- Unzahl von Massen möglich

Grössenmasse

- Umfang einer Software
- Basis für weitere Masse (Fehlerrate, Modularität, ..)
- Skala: Verhältnisskala
- NCCS: Non Commented Source Statements
 - Zählung der Codezeilen (ohne Kommentare / Leerzeilen)
 - genaue Zählregeln nötig
 - unabhängig von Programmiersprache
 - leicht automatisch messbar

Komplexitätsmasse

- Komplexität einer Software
- möglicher Indikator für Fehleranfälligkeit / Pflegbarkeit
- Skala: Intervallskala
- Problem der Additivität: Kombination zweier Komponenten kann massiv komplexer sein als Summe der beiden Komplexitäten ($1 + 1 = 3$)
- Zyklomatische Komplexität
 - Messung des Flussgraphen
 - Vereinfachung: Anzahl IF, WHILE, REPEAT, .. + 1
 - Validität: Spaghetticode und gut strukturiertes Programm können gleiche Komplexität haben
 - Skala: Verhältnisskala, aber nicht additiv (sogar kontraintuitiv, Komplexität wird bei Kombination kleiner, da ein Endpunkt wegfällt)
 - gut zur Berechnung der Anzahl Programmzweige
- Software Science
 - Berechnung Komplexität anhand verschiedener Kenngrössen
 - veraltet, Nutzen nicht beweisen → nicht anwenden

Zuverlässigkeitsmasse

- Messen der Zeit (Betriebsstunden) oder Anzahl Transaktionen zwischen zwei Fehlern
- MTTF: Mean Time To Failure
 - häufigstes Zuverlässigkeitsmass
 - Testreihe mit zufälligen Testdaten durchführen
 - durch statistische Verfahren auf Verhalten des Programms schliessen
 - Konfidenzgrad bestimmt Anzahl Testfälle (oft sehr gross)

- Testverteilung muss wie in Real World Anwendung sein (Belastung / Auslastung)
- Fehlerdichte
 - Anzahl Fehler / 1000 NCSS
 - bei gleichen Entwickler-Team sind Hochrechnungen möglich

Funktionalitätsmasse

- Messen des Funktionsumfangs der Software
- Function Points
 - Zählen / Bewerten von Dateieingaben, Dateiausgaben, Anfragen, Schnittstellen, Datenbestände
 - Skala: Verhältnisskala, nicht additiv
 - Vergleich über Programmiersprachen hinweg
 - Unterschätzung bei Algorithmen-lastigen Programmen
- Erstellungsaufwand
 - nur näherungsweise Messung der Funktionalität
 - Berücksichtigung von zusätzliche Faktoren (Können der Entwickler, ...)
 - in gutem Prozess bereits als Prozessmass erhoben
 - Skala: Verhältnisskala, additiv

Prozessmasse

Allgemein

- Messen von Prozess-Qualitäten
- Bestimmung von Basisgrößen
- Ableiten von Kennzahlen
- Beurteilung / Lenkung des Software-Entwicklungsprozesses

Basisgrößen (pro Komponente)

- Aufwand total
- Aufwand für Qualitätsmassnahmen (total / nur für Fehlerbehebung)
- Durchlaufzeit (Projekt, Entwicklung Komponente, ...)
- Anzahl gefundener Fehler (vor / nach Going Live)
- Produktgrößen (NCSS, ...)

Anforderungen

- genaue Zählregeln
- sorgfältige Erfassung der Basisgrößen
- klar definierte Prozesse (Projektanfang, Projektende, ...)

abgeleitete Prozessmasse

- Entwicklungskosten
- Produktivität
- Termintreue / Kostentreue (Soll – Ist)
- Fehlerrate / Defektrate
- Fehlerkosten / Defektkosten
- Qualitätskosten

Qualitätsmodelle

Idee

- jedes Produkt / Projekt benötigt individuelle Qualitätsplanung
- viele Ziele wiederholt in ähnlicher Weise
- Verallgemeinerung in Qualitätsmodell

Qualitätsmodell von McCall

- Menge von allgemeingültigen Qualitätszielen (Faktoren)
- Satz charakteristische Merkmale pro Faktor
- messbare Kenngrößen pro Merkmal

Qualitätsfaktoren

- Korrektheit: Ausmass, in dem Software Spezifikationen erfüllt
- Zuverlässigkeit: Fähigkeit des Systems, verlangte Funktionalität unter gegebenen Rahmenbedingungen für gegebene Zeit zu erfüllen
- Effizienz: Ausmass, in dem System Leistungen mit Minimum an Ressourcenverbrauch erbringt
- Integrität: Ausmass, in dem System unberechtigte Zugriffe / Veränderungen auf Programme / Daten verhindert
- Verwendbarkeit: Einfachheit, mit der ein System bedient, Daten vorbereitet und Ergebnisse interpretiert werden können
- Wartbarkeit: Einfachheit, mit der System zur Fehlerbehebung geändert, Funktionalität erweitert, an neue Umgebung angepasst werden kann
- Flexibilität: Einfachheit, mit der System an nicht vorgesehene Umgebung / Anwendung angepasst / erweitert werden kann
- Testbarkeit: Ausmass, in dem System Erstellung von Testbedingungen, Durchführung von Test, Feststellung der Erfüllung erleichtert
- Portabilität: Einfachheit, mit der System zwischen Hard-/Softwareumgebungen transferiert werden kann
- Wiederverwendbarkeit: Ausmass, in dem Stück Software in mehreren Programmen / Softwaresystemen verwendet werden kann
- Verknüpfbarkeit: Einfachheit, mit der mehrere Systeme Informationen austauschen und benutzen können

Vorteile

- Qualitätsfaktoren sind über Merkmale / Kenngrößen nachvollziehbar / messbar definiert
- Vereinheitlichung der Vorstellungs-/Begriffswelt über Qualitäten

Nachteile

- kausale Zusammenhänge zwischen Kenngrößen, Merkmalen und Faktoren nur hypothetisch, nicht statistisch abgesichert
- keine Rücksichtnahme auf individuelle Qualitätsanforderungen von Produkten / Projekten (Bsp: keine Verfügbarkeit bei McCall, wichtig für Telefonvermittlung)

Zielorientiertes Messen

Ansätze für Messungen

- definitorischer Ansatz
 - Ausgangspunkt: irgendwelche definierte Masse
 - Hypothese: Masse messen gesuchte Größen
 - Gefahr irrelevanter, nicht validierter Messungen
- Bequemlichkeitsansatz
 - Messen, was einfach zu messen ist
 - Gefahr sinnloser Messungen

- zielorientierter Ansatz
 - Ausgangspunkt: zu erreichende quantitative Ziele
 - Suche nach Massen, die Ziele quantitativ charakterisieren

GQM: Goal-Question-Metric

- bekanntester zielorientierter Ansatz
- Dreistufiges Vorgehen
 - Goal: Festlegen eines Qualitätsziels
 - Question: Suchen von Fragen, mit denen Zielerreichung festgestellt werden kann
 - Metric: Suchen von Massen, mit denen Fragen quantitativ beantwortet werden können
- Vorteile
 - nur das messen, was zur Zielerreichung beiträgt
 - Interpretation der ausgewählten Masse ist festgelegt

C.Stettler <cs@rogatec.ch>

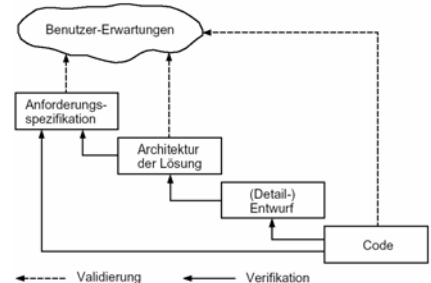
Grundlagen

Validierung

- tun wir das Richtige ?
- Prozess der Beurteilung eines System / einer Komponente während / am Ende des Entwicklungsprozesses
- Ziel: Feststellung, ob spezifizierte Anforderungen erfüllt sind
- Prüfung gegen Benutzererwartungen

Verifikation

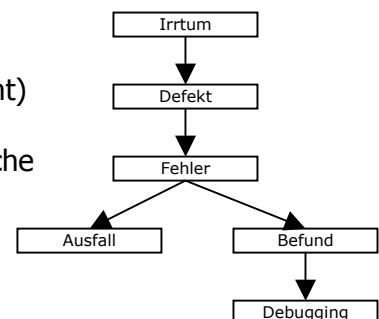
- tun wir es richtig ?
- Prozess der Beurteilung eines Systems / einer Komponente
- Ziele
 - Feststellen, ob Resultate einer Entwicklungsphase den Vorgaben der Phase entsprechen
 - Formaler Beweis der Korrektheit



Fehlermodell

Terminologie

- Fehler ist nicht gleich Fehler
- Irrtum (mistake): begangen von Person
- Defekt (defect, fault): wird als mögliche Folge von Irrtum gefunden (auch begangene Fehler / Fehlerursache genannt)
- Befund (finding): durch Inspektion gefundener Defekt
- Fehler (error): bei Ausführung von Software mit defekt, tatsächliche Ergebnisse weichen von erwarteten / richtigen Ergebnissen ab
- Ausfall (failure): mögliche Folge eines Fehlers (Systemausfall)
- bei Feststellung des Fehlers: Fehlerursache finden, Fehlerbeseitigung (Debugging)
- Umgangssprache: oft nur Fehler (error, bug)



Unterscheidung

- in Qualitätsmanagement wichtig
- Defekt hat häufig mehrere Fehler zur Folge
- Defekt führt nicht immer zu Fehler
- nicht jeder Fehler hat Defekt als Ursache
- nicht jeder Fehler hat Ausfall zur Folge

Software-Prüfung

Grundsätze

- nur gegen Vorgaben (Anforderungen / Vergleichsresultate) prüfen
- systematische Prüfung
 - Prüfstrategie festlegen

- Prüfung planen (in Projektmanagement)
- Prüfvorschriften erstellen
- Prüfung nach Vorschrift durchführen
- Prüfergebnisse dokumentieren
- Korrektur der erkannten Fehler
- Prüfverfahren müssen definiert sein, müssen reproduzierbare Ergebnisse liefern
- Prüfergebnisse müssen dokumentiert werden

Prüfverfahren

- statische Verfahren
 - Review (Inspektion, Walkthrough)
 - statische Analyse
 - Korrektheitsbeweis (formale Verifikation)
 - Messen
- dynamische Verfahren
 - Testen
 - Simulieren
 - Prototyping

Prüfstrategie

- Qualitätsmerkmale gewichten
 - für jedes Qualitätsmerkmal Risiko für Kundereklamation abschätzen
 - Priorisierung der Merkmale nach Risiko
 - je weniger Risiko, desto weniger Prüfung
- Prüfverfahren festlegen
 - was: zu prüfende Artefakte
 - wann: verbindliche Meilensteine
 - wie: Prüfverfahren
- grobe Aufwandsabschätzung
- Ergebnisse in Projektplanung übernehmen → Feedback

Kapitel 10: Testen

1. Grundlagen

- „Prozess, ein Programm mit der Absicht auszuführen, Fehler zu finden“
- durch Testen kann man Korrektheit eines Programms *nicht beweisen*
 - Bsp: Eingabe von zwei 16 Bit Zahlen: > 4'000'000'000 Testfälle ($2^{16} \cdot 2^{16} = 2^{32}$)
- Voraussetzung: Ergebnisse müssen bekannt sein
 - Test aufgrund einer Spezifikation
 - Test aufgrund vorhandener Testergebnisse (Regressionstest)
- *Fehlersymptome*, nicht *Fehlerursachen* werden gefunden
- verschiedene Testmöglichkeiten: Laufversuch, Wegwerf-Test, systematischer Test

2. Testgegenstand

- Modultest, Integrationstest, Systemtest, Abnahmetest
- Abnahme: nicht Fehler finden, sondern zeigen, dass Anforderungen erfüllt

3. Testablauf

- Planung: was – wann – wie – wie lange
 - Termine und Kosten für
 - Testvorbereitung
 - Testdurchführung
 - Testauswertung
 - Testperson bestimmen
- Vorbereitung: Auswahl der Testfälle, Erstellung der Testvorschrift
- Durchführung:
 - gemäss Testvorschrift vorgehen und Ergebnisse notieren
 - Prüfling während des Tests nicht verändern!
- Auswertung: Befundliste erstellen
- Fehlerbehebung: ist nicht Bestandteil des Tests!

4. Testfälle

- Anforderungen:
 - repräsentativ
 - fehlerintensiv
 - redundanzarm
 - ökonomisch

5. Testverfahren

5.1. funktionsorientierter Test (Black-Box-Test)

- Testfall-Auswahl aufgrund Spezifikation
- Programmstruktur kann unbekannt sein
- Wahl der Testfälle so, dass
 - Funktionsüberdeckung
 - Ausgabeüberdeckung
 - Ausnahmeüberdeckung
 - Attributüberdeckung
- Techniken der Testauswahl
 - Äquivalenzklassenbildung
 - Grenzwertüberprüfung
 - Ursache-Wirkungs-Graphen
 - Fehler raten (Error guessing)

5.2. strukturorientierter Test (White-Box-Test)

- Testfall-Auswahl aufgrund Programmstruktur
- Spezifikation muss ebenfalls bekannt sein (erwartete Resultate)
- Wahl der Testfälle so, dass
 - Anweisungsüberdeckung
 - Schwaches Kriterium
 - Fehlende Anweisungen werde nicht entdeckt
 - Zweigüberdeckung
 - Bestimmung der Zweige: Verzweigungen und Schleifen
 - Jede if-Anweisung resp. Schleife hat je zwei Zweige
 - In der Praxis angestrebt
 - Falsch formulierte Bedingungsterme werden nicht entdeckt
 - Pfadüberdeckung
 - Bestimmung der Pfade: alle Kombinationen aller Programmzweige bei maximalem Durchlauf aller Schleifen
 - Für Programme mit Verzweigungen in Schleifen nicht testbar, da die Anzahl der Pfade zu gross
 - Überdeckungsgrad

6. Testplanung und -dokumentation

- Testvorschrift
- Testprotokoll (evt. Gleich wie Testvorschrift)
- Testzusammenfassung (Nachweis über Durchführung der Tests & Gesamtergebnis)
- Aufbau einer Testvorschrift (Bsp. in den Übungen):
 1. Einleitung
 - 1.1. Zweck
 - 1.2. Testumfang
 - 1.3. referenzierte Unterlagen
 2. Testumgebung
 - 2.1. Überblick
 - 2.2. Testmittel
 - 2.3. Testdaten, Testdatenbank
 - 2.4. Personalbedarf
 3. Annahmekriterien
 4. Testfälle
- Darstellung der Testfälle:

Aufbau

Testfall Nr.	Eingabe	Erwartetes Resultat	Befund
1-1-1	Eingabewert 1	Ausgabewert 1	OK
1-1-2	Eingabewert 2	Ausgabewert 2	Nicht OK
etc.			
etc.			

Gliederung

Testsequenz 1-1 (ganze Tabelle)

Zusätzlich zu dieser Darstellung werden Zweck, Vorbereitungs- und Aufräumarbeiten dokumentiertl.

7. Testgeschirr

- stellt Daten für den Prüfling zur Verfügung
- nimmt Resultate entgegen und protokolliert sie
- berechnet / simuliert die Ergebnisse der vom Prüfling aufgerufenen Komponenten

8. Integrationstest

- Aufwärtsintegration (bottom-up)
- Abwärtsintegration (top-down)

9. Testen nicht-funktionaler Anforderungen

- Leistungsanforderungen
 - Leistungstest
 - Lasttest
 - Stresstest
 - Ressourcenverbrauch
- Testen besonderer Qualitäten
- wenig bis gar nicht testbar:
 - Zuverlässigkeit
 - Benutzbarkeit
 - Sicherheit (teilweise)

10. Testen von Benutzerschnittstellen

- Funktionalität: alle Funktionen zugänglich?
- Benutzbarkeit: Bedienbarkeit, Erlernbarkeit, Kundenbedürfnisse erfüllt?
- Dialogstruktur: Vollständigkeit, Konsistenz, Redundanz, Metapherkonformität (z.B. Navigation einer Website: Merkmal bei Menüpunkten mit Unterkapitel, dass der User weiss, dass sich darunter was verbirgt)
- Antwortzeitverhalten
- zusätzlich für Web-basierte Benutzerschnittstellen:
 - Linktest (keine 404, 403, etc)
 - Sicherheitstest
 - Zugangstest (Sichtbarkeit, Erreichbarkeit, Verfügbarkeit)
 - Kompaibilitätstest (Browser, OS, Bildschirmauflösung)

11. Kriterien für den Testabschluss

- keine Fehler mehr in den Testdatensätzen der Testvorschrift
 - übliches Kriterium bei der Abnahme
- Prüfkosten pro entdecktem Fehler > als voraus festgelegte Grenze (unwirtschaftlich)
 - Voraussetzung: Erfassung der Prüfkosten & Anz. gefundener Fehler
- während der Ausführung bei x im voraus bestimmten Menge von Testfällen keine Fehler mehr auftreten
 - z.B. bei Systemtest mit zufällig gestimmten Testdaten. Anzahl der hintereinander fehlerfrei auszuführenden Testfälle bestimmt sich aus der geforderten Zuverlässigkeit
- Fehlerdichte < als im voraus festgesetzt (Fehlerdichte z.B. 2 Fehler/1000 Codezeilen)
 - Problem: Fehler, die nie gefunden werden bzw. erst im Betrieb entdeckt

Abnahmetest	<ul style="list-style-type: none"> ▪ Test des gesamten Systems durch Kunden ▪ Test ob Anforderungen erfüllt ▪ Black-Box
Anweisungsüberdeckung	Jede Anweisung des Programms wird min. 1x durchlaufen
Äquivalenzklassenbildung	Eingabedaten werden in Äquivalenzklassen eingeteilt, aus jeder Klasse wird ein Repräsentant getestet
Attributüberdeckung	Alle geforderten Attribute testen (@Black-Box)
Ausgabeüberdeckung	Jede spez. Ausgabe min. 1x erzeugt (@Black-Box)
Ausnahmeüberdeckung	Jede spez. Ausnahme-/Fehlersituation min. 1x erzeugt (@Black-Box)
Belastungstest	Unter möglichst ungünstigen Bedingungen testen
Fehler raten (error guessing)	Intuitive Testfallauswahl, aufgrund Erfahrung, ergänzt andere Methoden
Funktionsüberdeckung	Jede spezifizierte Funktion min. 1x aktiviert (@Black-Box)
Grenzwertüberprüfung	Grenzfälle zulässiger Datenbereiche werden getestet
Integrationstest	
Lasttest	Verhalten bei (noch regulärer) Starklast
Laufversuch	<ul style="list-style-type: none"> ▪ Entwickler testet ▪ fortlaufend Fehler behoben ▪ Test endet bei vernünftigen Ergebnissen
Leistungstest	Zeiten, Mengen, Ragen, Intervalle werden getestet
Modultest	<ul style="list-style-type: none"> ▪ Test einzelner Funktionen, Methoden, Klassen durch Entwickler ▪ Zweigüberdeckung ▪ White-Box ▪ Testen während der Entwicklung
Pfadüberdeckung	Jeder Programmpfad wird min. 1x durchlaufen
Regressionstest	Es wird gegen vorhandene Testergebnisse getestet
Ressourcenverbrauch	Die in der Spezifikation definierten Maximalressourcen müssen getestet werden
Stresstest	Verhalten bei Überlast muss getestet werden (bricht das Programm ab einem bestimmten Punkt zusammen?)
Systematischer Test	<ul style="list-style-type: none"> ▪ Geplanter Test mit Testvorschrift ▪ Soll-Ist Vergleich ▪ separate Fehlersuche/-behebung ▪ Dokumentation Testfälle ▪ Test endet wenn Testziele erreicht
Systemtest	<ul style="list-style-type: none"> ▪ Test des gesamten Systems durch den Entwickler ▪ Gesamte Funktionalität min. 1x testen ▪ Funktionsorientiertes Testen (Black-Box) ▪ Systematisches Testen nach Integration, vor Abnahme

Testabschnitt	Zusammenfassung der Testfälle mit gemeinsamen Vorbereitungsaufgaben
Testgeschirr	Umgebungssoftware bei Komponenten- und Integrationstest
Testsequenz	Untergliederung der Testabschnitte zur Verbesserung der Übersicht
Überdeckungsgrad	Prozentsatz, mit dem eine angestrebte Überdeckung angestrebt wird (Verhältnis Anzahl überdeckte Elemente zu Anzahl vorhandene Elemente)
Ursache-Wirkungs-Graphen	Bestimmung von Kombinationen von Eingabedaten, die auf eine gewünschte Wirkung hinzielen
Wegwerf-Test	<ul style="list-style-type: none"> ▪ X testet ohne System ▪ fortlaufend Fehler beheben ▪ Test endet, wenn Tester sagt „ist genug“
Zweigüberdeckung	Jeder Programmzweig wird min. 1x durchlaufen

Kapitel 11: Review

1. Grundlagen

- Formell organisierte Zusammenkunft von Personen zur inhaltlichen oder formalen Prüfung einer Software-Einheit (Programm, Dokument) nach vorgegebenen Kriterien
- Technischer Review: Sachziele verfolgt
- Management-Review: Erreichung von Terminen & Kosten überprüft, weiteres Vorgehen wird entschieden
- Sonderform: Selbst-Review
- Ziel: Schwachstellen und Mängel aufzeigen, Stärken bestätigen → Qualität beurteilen
- Ergebnis: Bericht mit Befundliste
- Warum Reviews:
 - Fehler finden
 - 80-90% der Fehlerentdeckungen über gesamte Produktlebenszeit in Reviews
 - Senkung der Restfehlerraten
 - Reviewen ist billiger als testen
 - Weniger Vorbereitungs- und Durchführungsaufwand
 - Meistens Ursachen gefunden, nicht nur Symptome
 - Nicht alles testbar, aber alles reviewbar
 - Besser werden: Wissenstransfer
- Review-Material:
 - Prüfunterlagen
 - Referenzunterlagen
 - Sachlich-inhaltliche Vorgaben
 - Vorgeschriebene Standards
 - Prüflisten (Checklisten)

2. Review-Formen

2.1. Inspektion

- Prinzip: Prüfling wird von Gutachtern nach vorgegebenen Prüfkriterien Zeile für Zeile inspiziert
 - Individuelle Vorbereitung der Gutachter
 - Gutachter mit verwendeten Entwicklungsmethoden und -sprachen vertraut
 - Gegen klare Vorgabedokumente prüfen → sinnvoll
 - Wirkungsvollste Reviewtechnik
- Klassische Inspektion nach Fagan (Urform)
 - Startsituation → Vorstellung des Prüflings
 - Vorbereitung der Gutachter (keine individuellen Befundlisten), bloss lesen und verstehen des Codes
 - Sitzung: übliche Rollen (+ Vorleser; liest Code Zeile für Zeile vor)
 - Fortlaufend einhaken und Befunde bringen durch Gutachter
- Team-Inspektion (an Uni ZH gelehrt Form)
 - Vorbereitung der Gutachter inkl. Befundliste
 - Sitzung: nur zum Zusammentragen und Bewerten der Befunde
 - Doubletten und falsche Befunde eliminieren
 - Nachteil: Aufwand für die Sitzung
- N-Individuen-Inspektion (ohne Sitzung)
 - Prüfung und Befunderhebung vollständig individuell durch Gutachter
 - Review-Befund = Summe der Gutachterbefunde
 - Experimente zeigen, dass die Sitzung kaum neue Befunde erbringt, die kein Gutachter in der Vorbereitung erkannt hat

- Kritische Durchsicht/Bewertung der Individualbefunde durch die Gruppe fehlt → Anzahl der falschen Befunde steigt
- Selbstinspektion
 - Autor ist sein eigener und einziger Gutachter
 - Weniger effektiv als Inspektion durch 3. (Bsp: ich überlese eher zwei mal den gleichen Fehler als ein dritter)
 - Jederzeit möglich

2.2. Walkthrough

- Prinzip: Autor geht Prüfling mit Gutachtern durch, die Darstellung wird gemeinsam nachvollzogen
- Praktisch keine Vorbereitung → Prüfung live in der Sitzung
- Gutachter müssen Entwicklungsmethoden und -sprachen nur soweit kennen, dass sie den Ausführungen des Autors folgen können
- Weniger wirksam als Inspektion
- Autoren können Gutachter durch Redekunst blenden und in die Irre führen

3. Durchführung eines Reviews

- Ablauf
 - Planung
 - Kosten-, Termin-, Personalpläne
 - Gutachter, Räume, etc. bereitstellen
 - Bis zu 15% des Gesamtaufwandes
 - Vorbereitung
 - Material verteilen (evt. Vorbesprechung)
 - Individuelles Prüfen durch Gutachter (Prüfliste, erheben/notieren der Befunde)
 - Individuelle Aufträge für die Gutachter möglich
 - Sitzung
 - Zusammentragen und Gewichten der Befunde
 - Geführt durch Moderator
 - Schreiber führt fortlaufend die Liste der Befunde
 - Gesamtbewertung am Schluss
 - Erstellen/Unterschreiben der Review-Zusammenfassung
 - Überarbeitung und Nachkontrolle
 - Behebung der Schwachstellen durch Autor(en)
 - Nach-Review falls erforderlich, sonst Freigabe durch PL
- Rollen
 - Moderator
 - Organisiert Review und leitet Sitzung
 - Einladung, Verteilung des Materials, Erstellung und Verteilung des Review-Berichts
 - Kann auch Gutachter sein (kleine Teams)
 - Gutachter
 - Bereitet sich vor
 - Berichtet über seine Befunde
 - Bewertet die Befunde
 - Schreiber
 - Erstellt fortlaufend die Liste der Befunde (allen sichtbar)
 - Kann auch Gutachter oder Autor sein
 - Autor
 - Hört zu und erläutert falls nötig, sonst passiv
 - Kann ggf. als Schreiber eingesetzt werden
- Inspektion: 2-4 Gutachter ↔ Walkthrough: 4-6 Gutachter
- kleine Teams effizienter als grosse

- Review-Regeln
 - Material rechtzeitig verteilt, Teilnehmer rechtzeitig eingeladen
 - 3-7 Beteiligte, wovon die Gutachter gut vorbereitet sein müssen
 - Sitzungsdauer max. 2h (nicht mehr Material verteilen)
 - Probleme nur nennen, nicht lösen
 - Problemlösungen/-Ansätze bei Mittagessen („dritte Stunde“)
 - Positives und negatives nennen
 - Keine Stilfragen diskutieren
 - Produkt bewerten, nicht Produzenten
 - Review-Berichte nie zur Bewertung von MA (→ eine Hand wischt die andere Hand-Prinzip unter den MA → Reviews werden schlechter)
 - Fehler in Referenzunterlagen ebenfalls notieren (separate Liste)

4. Der Review-Bericht

- Deckblatt
- Liste der Befunde

5. Review-Verfahren

- Prüfverfahren für Inspektionen
 - Paraphrasieren/Erklären
 - Checklisten durchgehen
 - Prüfprozeduren durcharbeiten
 - Szenarien durchspielen
 - Symbolisch ausführen
 - Verschiedene Perspektiven einnehmen
- Prüfverfahren für Walkthroughs
 - Vorlesen
 - Ereignisse/Szenarien durchspielen
 - Pfade verfolgen
 - Rollenspiel
- Selbstinspektion
 - Wie bei Inspektion
- Checklisten
 - Auflistung von Prüfpunkten
 - Abgestimmt auf Prüfling
 - Allgemeine Fragen
 - Fragen zur Adäquatheit und Vollständigkeit
 - Methodengebundene Fragen
- Prüfprozeduren
 - Prüfpunkte einer Checkliste abdecken
- Szenarien: durchspielen problembezogener Benutzungsszenarien

6. Review-Aufwand

- Im wesentlichen nur Personalaufwand
 - Zeitbedarf für Organisation des Review
 - Summe der Vorbereitungszeiten aller Gutachter
 - Dauer der Sitzung x Anzahl der Teilnehmer
 - Aufwand für Nachreview bei zu vielen Befunden

Kapitel 12: Statische Analyse

Grundlagen

Definition: Statische Analyse ist die Untersuchung des statischen Aufbaus eines Prüflings auf die Erfüllung vorgegebener Kriterien.

Die Prüfung ist

- statisch → Abgrenzung vom (dynamischen) Test
- formal durchführbar → Abgrenzung vom Review

Ziele sind

- Bewertung der Qualitätsmerkmale (Pfleg- und Testbarkeit) anhand struktureller Eigenschaften (Komplexität) des Prüflings
- Erkennen von Fehlern in neuer Software
- Prüfen ob formale Vorgaben erfüllt werden

Analyse von Programmen

- Die Syntax wird durch den Compiler analysiert
- Überprüfung der Typverträglichkeit von Operationen, Zuweisungen und Operationsaufrufen
- Programm- und Datenstrukturen

Formale Eigenschaften (Richtlinien eingehalten?...)

Strukturkomplexität des Programms (Einhaltung von Strukturierungsregeln...)

Fehler in den Daten (nicht deklarierte Variablen...)

- Vorgehen: Mit Tools (z.B. Compiler)

Programm wird geparkt → Syntaxfehler erkannt → Analysen → Befunde bewertet

Analyse von Algorithmen

Laufzeit- und Speichereffizienz sowie Gültigkeitsbereich. Analyse erfolgt manuell\$

Analyse von Spezifikationen, Entwürfen und Prüfvorschriften

- Syntaxanalyse der formal beschriebenen Teile
- Teilweise Struktur- und Flussanalysen

Kapitel 13: Fehlervermeidung

Prozesse mit kontinuierlicher Prüfung

Kontinuierliche Prüfung

- deckt Fehler früh auf
- vermeidet Weiterarbeit mit falschen Vorgaben
- besteht aus Reviews aller Dokumente einschliesslich Code
- Entwicklungsprozesse zur Fehlervermeidung sind z.B.

Cleanroom-Prozess

Persönliche Software Prozess (PSP)

- Cleanroom-Prozess (IBM)

Idee:

- Inkrementelle Entwicklung
- Strenge Spezifikation
- Ziel: Zuverlässigkeitsprognose, nicht das Finden von Fehlern

Kritik an Cleanroom:

- Neue Erkenntnisse der Softwaretechnik werden ignoriert (z.B. Information Hiding)
- Alle Fortschritte der Testtechnologie werden ignoriert

- Der Persönliche Software Prozess (PSP)

Idee:

- Aufwand jeder Arbeit wird vorab geschätzt, alle tatsächlichen Aufwendungen werden gemessen
- Alle Fehler und ihre Behebung werden notiert
- Durch Analyse der Messungen Schwachstellen erkennen und eliminieren

Stärken:

- Garantiert Sammlung von Erfahrungswerten für Aufwand der Bearbeitung von Aufgaben
- Fördert Arbeitshaltung (lieber langsamer, dafür genau arbeiten)

Schwächen:

- Vernachlässigt Gruppenarbeit
- Papierkrieg
- Nur auf Fehlerminimierung optimiert

Systematisches Entwerfen und Programmieren

Systematisches Arbeiten wirkt fehlervermeidend. Für das Entwerfen und Programmieren bedeutet dies vor allem

- Saubere Abgrenzung und Kapselung von Entwurfsentscheidungen und Verantwortlichkeiten (z.B. Information Hiding) → s. Kapitel 3
- Systematisches Programmieren → s. Kapitel 2

Dokumentier- und Codier-Richtlinien

Allgemein: Richtlinien sind nur sinnvoll, wenn sie relevante Dinge regeln und ihre Einhaltung überprüft wird

Dokumentier-Richtlinien:

- erzwingen Dokumentation
- vereinheitlichen Lesbarkeit der Dokumente

Codier-Richtlinien:

- erhöhen Lesbarkeit des Codes
- verhindern Verwendung schlechter oder gefährlicher Konstrukte

14 Software-Qualitätssicherung und - Prozessverbesserung

14.1 Grundlagen

Qualitätssicherung (quality assurance) – Teil des Qualitätsmanagements, der auf das Erzeugen von Vertrauen darauf gerichtet ist, dass Qualitätsanforderungen erfüllt werden.

Mittel sind in den nachfolgenden Kapitel beschrieben.

Software-Prozessverbesserung (software process improvement) – Prozess für die Änderung von SW-Prozessen auf der Grundlage fortlaufender Prozessbeurteilungen mit dem Ziel, die Produktqualität zu sichern und zu verbessern.

14.2 Qualitätsdokumentation

- **Notwendig zur Beschreibung** der Prozesse und Verfahren
- **Vertrauen schaffen** beim Kunden ins Qualitätsmanagement durch Einblick in einen allgemeinen Teil des Qualitätshandbuches.
- Die Dokumentation ermöglicht **Rückschlüsse auf Qualität** des Entwicklungsprozesses und der entwickelten Produkte.

14.3 Audits

Existenz des Qualitätsmanagementsystems garantiert nicht dessen Wirksamkeit, darum Prüfung (Audit):

- der verlangten Qualitätsorganisation
- der Arbeit nach dokumentierten Verfahren
- der verlangten Massnahmen

Arten:

- interne Systemaudits: Unternehmensinterne, regelmässige Überprüfung des Qualitätsmanagementsystems
 - o Qualitätsleiter: erstellt Jahresplan, schult Mitarbeiter als Auditoren
 - o Linienorganisation: Nehmen Aufwand in die Budgets auf, kooperiert mit der Sekundärorganisation für Qualität bei der Durchführung
 - o Sekundärorganisation für Qualität: führt Audits gemäss Auditplan durch.
- Interne Prozess-, Projekt- oder Produktaudits:
 - o spontan angesetzt ausserhalb des Jahresplanes,
 - o Durchführung bei Anzeichen grösserer Probleme und Abweichungen.
- Lieferanten/Kundenaudits: externe Audits (mit eigenen oder fremden Auditoren)
Als Entscheidungshilfe:
 - o Erfüllen die Lieferanten die Qualitätsanforderungen?
 - o Soll dieser Kunde beliefert werden, oder sogar eine Partnerschaft eingegangen werden?
- Zertifizierungsaudits: Zertifizierung des Qualitätsmanagementsystems durch Auditoren einer für die Zertifizierung autorisierten Prüfstelle

Ablauf:

- Vorbereitung:
 - o Auditziel festlegen
 - o Auditorenteam bilden (Auditor, Co-Auditor, evtl. dritte Person als Beobachter)
 - o Aufgaben im Team verteilen
 - o Vom auditierten Bereich Unterlagen anfordern
 - o Unterlagen studieren
 - o Fragenkatalog für Audit zusammenstellen

- Auditprogramm erstellen
- Vorgespräch und Einladung (Termin, Ort und Auditprogramm mit auditiertem Bereich absprechen, Einladungen verteilen)
- Durchführung:
 - Eröffnungsgespräch
 - Durchgehen des Fragenkatalogs mit den in der Vorbereitung ausgewählten Personen des auditierten Bereichs
 - Schwachstellen/Verbesserungspotentiale, aber auch Stärken aufdecken und protokollieren.
 - Schlussgespräch
 - ✍ Befunde besprechen, ggf. Empfehlungen abgeben
 - ✍ Evtl. Verbesserungsmaßnahmen vereinbaren
 - ✍ Evtl. Nachaudit/Termine für Verbesserungsmaßnahmen vereinbaren
- Abschluss:
 - Auditbericht erstellen und verteilen

Zeitbedarf:

- Vorbereitung: 0,5 – 2 Tage (je nach Erfahrung)
- Durchführung: 0,5 Tage
- Abschluss: 0,5 Tage

14.4 Publikation von Messgrößen

Stand bzw. Fortschritt qualitätsrelevanter Größen ausweisen

- gegenüber Kunden und Lieferanten und
- zur Information, zum Ansporn der MitarbeiterInnen

Beispiele sind Produktivität, Fehler- bzw. Defektraten, Fehler- bzw. Defektkosten, Anzahl pendenter Problemmeldungen, mittlere Durchlaufzeit einer Problemmeldung, Anzahl Reklamationen pro Monat

14.5 Zertifizierung

Qualitätsmanagement einer Firma kann durch dafür **autorisierte Stelle** zertifiziert werden.

- Das Zertifikat sagt aus, dass ein Qualitätsmanagementsystem existiert, welches gewisse Mindestanforderungen erfüllt, und dass nach den Massangaben dieses Systems gearbeitet wird.
- **ISO Norm 9001** hat grosse praktische Bedeutung erlangt.
- Ein Zertifikat ist in der Regel **drei Jahre** gültig.
- Eine Zertifizierung bedeutet aber nicht automatisch, dass dieses Unternehmen Software hoher Güte herstellt, Zielerfüllung kann auch mit einem falschen Ziel erreicht werden.

Es kann auch die Qualität einzelner Produkte bestätigt werden durch Zertifizierung.

- Din 66285 beschreibt Zertifikat „Gütezeichen Software“

14.6 Software-Prozessbeurteilung

Beurteilung der Qualität des Software-Prozesses (Process Assessment)

- Schwachstellen im Prozess sollen aufgedeckt werden und gezielt verbessert werden
- Standardbeurteilungsschemata dienen zum unternehmensübergreifend vergleichbaren IST-Zustand, wie auch zu einem guten Verbesserungsprogramm.

Bekannte Prozessbeurteilungsverfahren:

- Capability Maturity Model (CMM) – siehe unten
- SPICE / ISO 15504 (Software Process Improvement and Capability dEtermination)
- Bootstrap – EU-Programm zur Prozessverbesserung
- Trillium – Erweiterung von CMM, speziell für Telekommunikationssoftware

Das Capability Maturity Model

(Anfang 90er Jahre)

- Ziele:
 - o Feststellen des Reifegrads der Software-Prozesse eines Unternehmens
 - o Handlungsrahmen bereitstellen zur stufenweisen Verbesserung des Reifegrades.
- Fünf Stufen der Prozessreife (maturity levels)
- Jede Stufe ist durch Schlüsselbereiche charakterisiert und Schlüsselpraktiken beschrieben.

Stufen:

1 Ad Hoc (Initial)

- Ad-Hoc-Prozess
- keine formelle Planung und Kontrolle
- kein oder schlechtes Konfigurationsmanagement

2 Wiederholbar (Repeatable)

- Etabliertes Projektmanagement
- Standardprojektlauf wird beherrscht, neuartige Projekte bilden grössere Risiken
- Prozess abhängig von den Personen, die ihn durchführen
- Keine Prozessverbesserungsmassnahmen sind etabliert

3 Definiert (Defined)

- Definitionsgemässer Prozessablauf
- Es existiert Gruppe, die den Prozess lenken und verbessern muss.

4 Geführt (Managed)

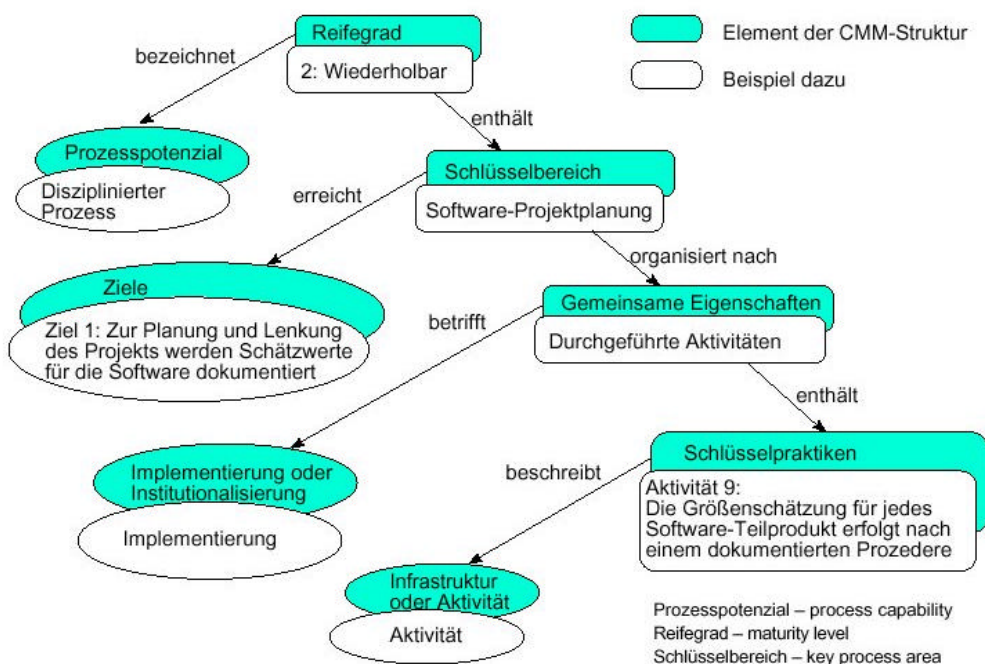
- Eine Mindestmenge an Qualitäts- und Produktivitätsmessgrößen wird erhoben und ausgewertet
- Es gibt Prozessdatenbank und Mittel zur Pflege und Auswertung dieser Daten

5 Optimierend (Optimizing)

- Etablierter Regelkreis für Messung und Verbesserung des Prozesses
- Datenerhebung und Erkennung von Schwachstellen weitgehend automatisiert
- Durchgeführte Massnahmen aus dem erhobenen Datenmaterial begründet
- Etablierte Ursachenanalyse für alle Fehler und zugehörige Fehlerpräventionsmassnahmen

Struktur des CMM

Die Struktur des CMM



Kritik am CMM:

- Zuwenig Differenziert: Mehrheit der Software-Hersteller auf Stufe 1
- Zu anspruchsvoll: kaum ein Unternehmen schafft es auf Stufe 4 oder sogar fünf
- Zu einseitig: Idealbild von mittels statistischer Analyse von Prozessdaten gelenkten SW-Prozessen
- Stufenweise Beurteilung statistisch problematisch: Wenige Schlüsselversager können zur totalen Abwertung führen.
- Stufe 1 ist keine Stufe, sondern einfach der Ausschuss

14.7 Software-Prozessverbesserung

Grundidee und zugehöriger (Meta-)prozess:



Risiken:

- Gefahr des Arbeitens an Prozessen statt am Produkt
- Gewinnen der Betroffenen ist notwendig

Beispiel anhand CMM: Verbesserungspotentiale von Stufe zu Stufe

Stufe 1 \rightarrow 2:

- Projektmanagement
- Konfigurationsmanagement
- Qualitätsmanagement

Stufe 2 \rightarrow 3:

- Ausbildung
- Qualitätsmassnahmen (Review, Test, etc.)
- Prozessdefinition und -überwachung; Etablierung einer Prozess-Gruppe

Stufe 3 \rightarrow 4:

- Messung und Analyse der Prozesses
- Quantitative Qualitätsplanung

Stufe 4 \rightarrow 5:

- Analyse und Vermeidung von Fehlern
- Verbesserungsmassnahmen Automatisieren und Institutionalisieren

Auf Stufe 5:

- Das Erreichte halten
- Auf Veränderungen des Umfeldes mit weiteren Verbesserungsmassnahmen reagieren

Beispiel von Prozessverbesserung anhand von SPICE / ISO 15504
(Internationale Norm)

- Umfasst: Prozessbeurteilung, -verbesserung, -potentialermittlung
- Verwendet 5 Prozesskategorien mit etwa 40 Prozessen
- Die Prozessreife wird mit sechs Stufen und mit neun gemeinsamen Attributen (generische Praktiken) bestimmt.

- Die Reife jedes Prozesses einer Organisation wird einzeln bestimmt und zu einem Profil verdichtet.

Folie 14-25 zeigt die neun Teile der ISO 15504