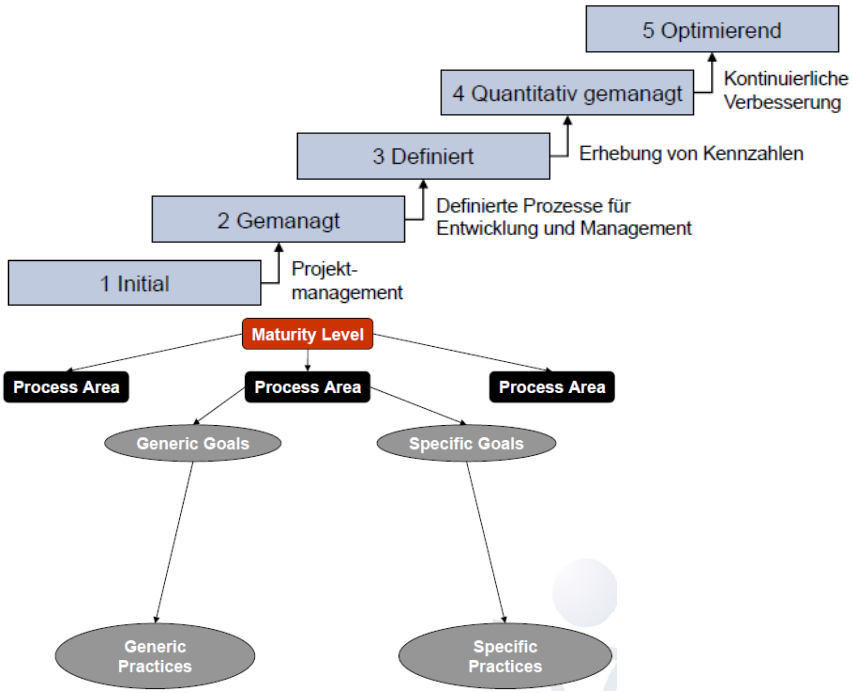
Software Quality

Grundlagen

- Qualität im klassischen Sinne: Das entwickelte Produkt sollte mit seiner Spezifikation übereinstimmen.
- Problem bei SW-Systemen
  - die Spezifikation sollte sich an den Eigenschaften des vom Kunden gewünschten Produkts orientieren
  - bestimmte Qualitätseigenschaften lassen sich nicht eindeutig festlegen
  - es ist sehr schwierig vollständige SW-Spezifikationen zu schreiben
  - trotz vollständiger Spezifikation kommt es vor, dass ein Produkt nicht den Qualitätsansprüchen der Benutzer genügt
- Qualitätsmanagementmodelle: ISO 900x, CMMI, Bootstrap, TickIT, SPICE, TQM

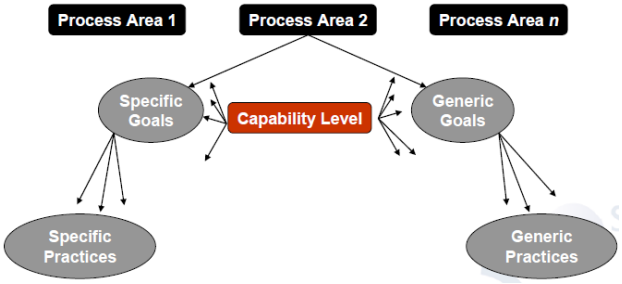
CMMI

- zwei Darstellungsformen
  - stufenförmige Darstellung (Staged Representation)
    - fünf Reifegrade



  - kontinuierliche Darstellung (Continous Representation)
    - fünf generische Ziele + Stufe 0
    - Fähigkeitsgrad *n* ist für ein Prozessgebiet erreicht, wenn das generische Ziele GG *n* erfüllt ist
    - ein Prozessgebiet hat den Fähigkeitsgrad 0, wenn kein generisches Ziel erreicht wird

Fähigkeitsgrad 0: unvollständig (*incomplete*)
Fähigkeitsgrad 1: durchgeführt (*performed*)
Fähigkeitsgrad 2: gemanagt (*managed*)
Fähigkeitsgrad 3: definiert (*defined*)
Fähigkeitsgrad 4: quantitativ gemanagt (*quantitatively managed*)
Fähigkeitsgrad 5: optimierend (*optimizing*)



- verschiedene Anwendungsgebiete
  - Software- und Systementwicklung (CMMI-SE/SW)
  - integrierte Prozess- und Produktentwicklung (CMMI-IPPD)
  - Kauf von SW (CMMI-SS, Supplier Sourcing)
- Prozessgebiete (Process Areas, PAs)
  - sind die wichtigsten Strukturelemente im CMMI
  - ein Prozessgebiet ist jeweils eine Zusammenfassung aller Anforderungen zu einem Thema
  - gemeinsame Struktur der Prozessgebiete
    - spezifische Ziele: gelten nur für das jeweilige Prozessgebiet
    - generische Ziele: beschreiben die Institutionalisierung des Prozessgebiets (regelmäßige, dauerhafte und effiziente Umsetzung spezifischer Ziele), prozessgebietübergreifend
    - jedem Ziel sind ein oder mehrere Praktiken (spezifisch oder generisch) zugeordnet
  - Kategorien von Prozessgebieten: Prozessmanagement, Projektmanagement, Ingenieurdisziplinen, Unterstützung
- Modellbestandteile
  - geforderte Modellbestandteile (generische und spezifische Ziele)
  - erwartete Modellbestandteile (Praktiken)
  - informative Modellbestandteile (Zweck eines Prozessgebiets, einführende Beschreibung, Referenzen, Namen und Zuordnungen von Zielen und Praktiken, typische Arbeitsergebnisse)
- Auswahl der Variante
  - Vorteile der stufenförmigen Darstellung
    - Pfad für Verbesserungen ist klarer vorgegeben, fest vorgegebene Reihenfolge der Prozessthemen durch die Reifegrade
    - Migration von SW-CMM ist einfacher
    - Vergleich zwischen Organisationen und Teilorganisationen ist leichter → Ziele können einfacher formuliert werden
  - Vorteile der kontinuierlichen Darstellung
    - größere Flexibilität bei der Verbesserung, Konzentration auf Themen mit dem höchsten Nutzen
    - Migration von EIA/IS 731 ist einfacher
  - weitere Möglichkeit: Kombination aus stufenförmiger und kontinuierlicher Darstellung
- Appraisal
  - Überprüfung einer Organisation im Hinblick auf die Umsetzung der Anforderungen eines Prozessmodells
  - Class A Appraisal: hoher Umfang, 1x im Jahr, hat Rating zum Ziel
  - Class B Appraisal: mittlerer Umfang, 1-2x im Jahr, Bestimmung von Stärken/Schwächen
  - Class C Appraisal: kleiner Umfang, häufig, kontinuierliche Beobachtung/Bewertung
  - Vorgehensweisen: Standard CMMI Appraisal Method for Process Improvement (SCAMPI)

Qualitätsstandards

- der Qualitätssicherungsprozess umfasst das Festlegen / Auswählen von Standards, die für den SE-Prozess bzw. das Produkt relevant sind
- Arten von Standards
  - Produktstandards
    - bezieht sich auf das Produkt
    - z.B. Standards für Dokumentation, Kommentare im Code, etc.
  - Prozessstandards
    - bezieht sich auf die Abläufe
    - z.B. Beschreibung der im Laufe des Prozesses zu erstellenden Dokumente
- Vorteile von Qualitätsstandards
  - beschreiben eine Art „Best Practice“
  - stellen die Rahmenbedingungen bereit, welche durch die Qualitätskontrolle überprüft werden können
  - gemeinsame Basis: die von einer Person ausgeführte Basis kann ohne zusätzlichen Lernaufwand von einer anderen Person weitergeführt werden
- Beispiele

Produktstandards	Prozessstandards
▪ Formular für das Entwurfs-Review ▪ Struktur des Anforderungsdokuments ▪ Format eines Prozedurkopfes ▪ Stil der Java-Programmierung ▪ Format des Projektplans ▪ Änderungsantragsformular	▪ Durchführung des Entwurfs-Review ▪ Einreichen von Dokumenten an das Konfigurationsmanagement ▪ Ablauf der Versionsfreigabe ▪ Genehmigungsprozess für den Projektplan ▪ Ablauf der Änderungsprotokolle ▪ Ablauf der Testprotokollierung

Qualitätsplanung

- sollte im frühen Stadium des SW-Prozesses beginnen
- Darlegung und Beurteilung der gewünschten Produktqualitätseigenschaften
- Auswahl von relevanten Standards
- Ergebnis ist ein Qualitätsplan
  - möglichst kurz, damit er gelesen wird
  - Festlegung der wichtigsten Qualitätsmerkmale
  - Prioritäten werden festgelegt

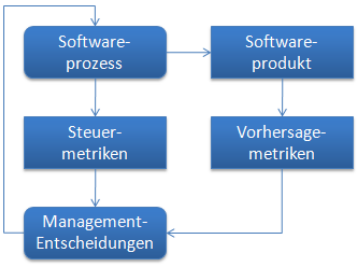
Qualitätskontrolle

- die Überwachung des SE-Ablaufs soll gewährleisten, dass die Standards und Verfahren zur Qualitätssicherung eingehalten werden
- Überprüfung der Ergebnisse des SW-Prozesses im Hinblick auf die im Qualitätskontrollprozess definierten Projektstandards
- für die Qualitätskontrolle existieren zwei (sich ergänzende) Ansätze
  - Qualitäts-Reviews
  - automatische SW-Beurteilung
- Qualitäts-Reviews
  - am weitesten verbreitete Methode zur Ermittlung der Qualität eines Prozesses / Produkts
  - manuelles Verfahren

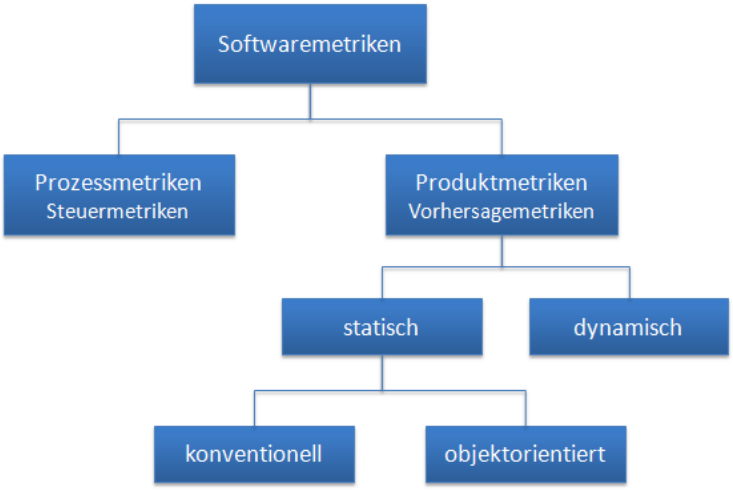
  - zur Aufdeckung möglicher Probleme überprüft ein Review-Team (teilweise oder gesamt)
    - den SW-Prozess, das System, die zugehörige Dokumentation
  - formelle Dokumentation der Review-Ergebnisse
- Review-Arten
  - Entwurfs- oder Programmüberprüfungen
    - Auffinden von Fehlern in Anforderungen / Entwurf / Code
    - Durchführung anhand einer Prüfliste
  - Fortschritts-Reviews
    - Bereitstellung von Informationen über den Gesamtfortschritt des Projekts für das Mgmt
    - Prozess- und Produkt-Review → Kosten, (Zeit-)Pläne
  - Qualitäts-Reviews
    - technische Analyse der Komponenten / Dokumentation
    - Ziel: Einhaltung der festgelegten Qualitätsstandards, Erkennung von Abweichungen zwischen Spezifikation und Entwurf / Code / Dokumentation
- Review-Team
  - die Mitglieder des Teams sollten einen effektiven Beitrag zur Durchführung des Reviews leisten können
  - der Kern des Teams besteht aus ca. 3-4 Hauptprüfern
  - ein Mitglied (leitender Entwickler) übernimmt die Verantwortung für wichtige technische Entscheidungen
  - das Team kann andere zur Teilnahme am Review einladen
- automatische SW-Beurteilung
  - ein Programm arbeitet die erstellte SW und Dokumente durch
  - Vergleich zwischen der SW / Dokumente und der für das jew. Projekt festgelegten Standards
  - quantitative Messung von SW-Merkmalen möglich → SW-Messung und -Metriken

SW-Messung/-Metriken

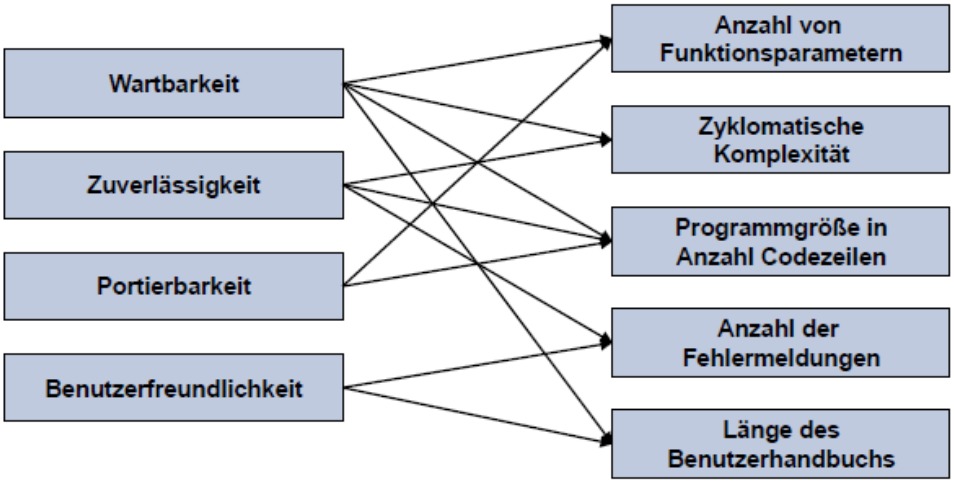
- SW-Messung
  - Ableitung von numerischen Werten für bestimmte Merkmale eines SW-Produkts / -Prozesses
  - Schlussfolgerungen über die Qualität der SW bzw. des SW-Prozesses anhand von Vergleichen zwischen den Werten untereinander oder mit organisationsweit geltenden Standards
- SW-Metrik
  - „jede Art von Messung, die sich auf ein SW-System, einen Prozess oder auf die Dokumentation bezieht“
  - „Eine SW-Qualitätsmetrik ist eine Funktion, die eine SW-Einheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der SW-Einheit.“
  - Beispiele: LOC, Fog-Index (Lesbarkeit eines Textbereichs), Anzahl gemeldeter Fehler
- Arten von SW-Metriken
  - Steuermetriken
    - beziehen sich auf die SW-Prozesse
    - Bsp: Aufwand / Zeit für die Beseitigung festgestellter Fehler
  - Vorhersagemetriken
    - beziehen sich auf die SW-Produkte
    - Bsp: zyklomatische Komplexität eines Moduls oder die durchschn. Länge von Bezeichnern



- Klassifikation von SW-Metriken

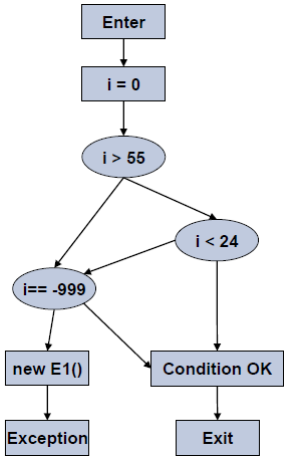


- Produktmetriken
  - befassen sich mit den Eigenschaften der SW selbst
  - dynamische Metriken
    - werden bei der Ausführung eines Programms gesammelt
    - hilfreich für die Beurteilung der Leistungsfähigkeit, Zuverlässigkeit, etc.
    - stehen in der Regel eng in Beziehung mit den Qualitätsmerkmalen
  - statische Metriken
    - werden bei Messungen der Systemdarstellung (z.B. Entwurf, Programm, Dokumentation) gesammelt
    - hilfreich für die Beurteilung der Komplexität, Verständlichkeit etc.
    - stehen in indirektem Verhältnis zu den Qualitätsmerkmalen
- Softwaremerkmale
  - oft können bestimmte Qualitätsmerkmale (z.B. Wartungsfreundlichkeit) nicht direkt gemessen werden
  - diese (externen) Merkmale hängen von verschiedenen Faktoren ab, es existieren keine direkten Metriken
  - interne SW-Merkmale können direkt gemessen werden (z.B. LOC)
  - um externe SW-Merkmale zu messen, müssen sie zu internen Merkmalen in Beziehung gesetzt werden können



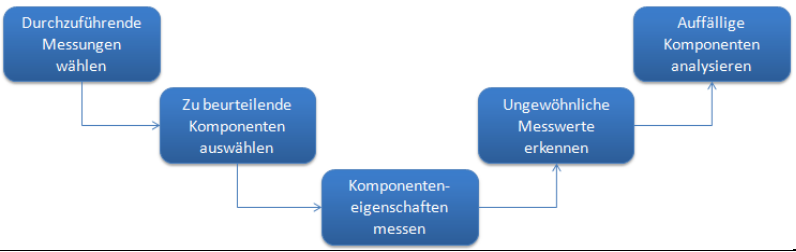
- Cyclomatic Complexity
  - Basis der Berechnung: Kontrollflussgraph eines Programms
  - Vorteil: einfache Berechnung
  - Nachteile: Komplexität der Module wird nicht berücksichtigt, hoher Wert impliziert nicht unbedingt Unübersichtlichkeit (z.B. switch-case-Anweisung)
  - Formel: V(G) = e – n + 2p (oder Anzahl der Entscheidungen + 1)
    - e: #Kanten
    - n: #Knoten
    - p: #verbundener Komponenten/Kontrollflussgraphen (ein Graph pro Funktion/Prozedur)

```
class Test {
  void test() throws E1 {
    int i = 0;
    if(i > 55 && i < 24 || i == -999)
      System.out.println("Condition is OK");
    else
      throw new E1();
  }
}
```



V(G) = 10-9+2

- Function Point Analyse (FPA)
  - FPA ist ein Maß für die Anzahl/den Umfang fachlicher Funktionen, die von einem SW-System unterstützt werden
  - unabhängig von der Art der Anwendung
  - die Anzahl der verwalteten Datenbestände, Erfassung-/Abfrage- und Auswertungsfunktionen bestimmen den Function-Point-Wert
  - fachliche Anforderungen (Spezifikation) müssen bekannt sein!
  - FPA-Phasen:
    - Phase 1: Zählen der Daten und Funktionen → unjustierter FP-Wert
    - Phase 2: Bewertung bestimmter Systemeigenschaften (nicht-funktionale Anforderungen) + unjustierten FP-Wert → justierter FP-Wert
    - Phase 3: Aufwandsschätzung mit justiertem FP-Wert, Bewertung anhand von unternehmenseigenen Referenzdaten (Erfahrungen aus Projekten, etc.)
  - FPA-Einsatzgebiete: Benchmarks, SLAs, SW-Beschaffung, Aufwandsschätzung
  - FPA-Nutzen:
    - Verbesserung der Aufwandsschätzung (Planungsqualität)
    - Preisvorteile bei der Beschaffung
    - besser Qualität bei SLAs
    - Wirtschaftlichkeit der Anwendungsentwicklung wird verständlicher
  - Ablauf einer Produktmessung



- Schwierigkeiten
  - Analyse
    - Schwierigkeit, die wirkliche Bedeutung der gesammelten Daten zu verstehen
    - häufig falsche Interpretation der Daten
  - Anwendung
    - viele Firmen haben die SE-Prozesse immer noch zu schlecht organisiert, um von Metriken Gebrauch zu machen

Verifikation und Validierung

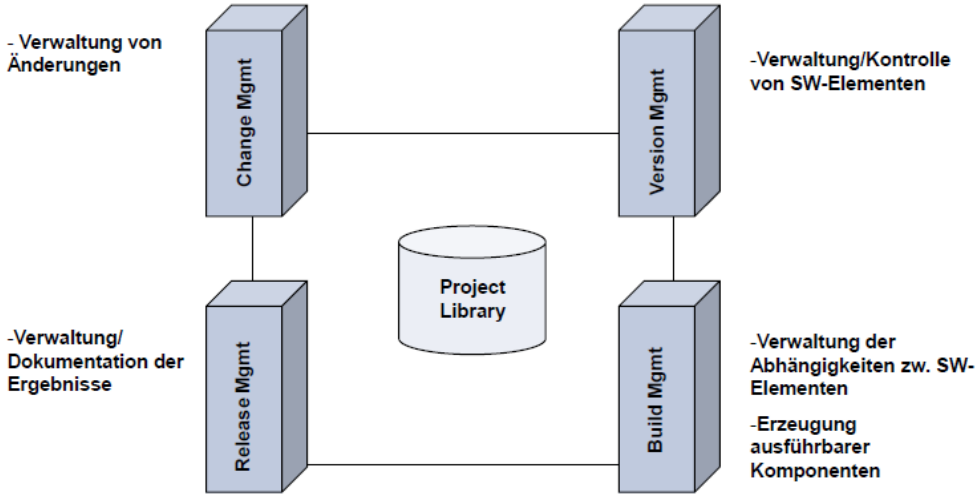
- Prozess der Überprüfung und Analyse zur Sicherstellung, dass eine SW mit ihrer Spezifikation übereinstimmt
- erstrecken sich über den gesamten Lebenszyklus (von Anforderungs-Reviews bis Produkttests)
- Unterschied nach Boehm
  - Validierung: Erstellen wir das richtige Produkt?
  - Verifikation: Erstellen wir das Produkt richtig?
- zwei Verfahren zur Prüfung und Analyse
  - SW-Inspektionen
    - Erscheinungsformen des Systems (Diagramme, Code, etc.)
    - statisches Verfahren, da eine Ausführung des Systems nicht erforderlich ist
  - Testen der SW
    - Untersuchung der (mit Hilfe von Testdaten) erzeugten Ausgaben und des Betriebsverhaltens
    - dynamisches Verfahren, da die Ausführung des Systems erforderlich ist

Configuration Management

Motivation

- SW-Systeme werden ständig weiterentwickelt
- es existieren im Laufe der Zeit viele verschiedene Versionen eines Systems (Windows/Linux, Demo/Vollversion)
- gleichzeitige (Weiter-)Entwicklung verschiedener Versionen eines Systems
  - für unterschiedliche Kunden / HW / OS
- „Daily Builds“
  - regelmäßige (tägliche) Erstellung des Gesamtsystems
    - Abliefern (Einchecken) der geänderten Komponenten durch die Entwickler
    - Erstellung einer neuen Systemversion
    - Durchführung von Systemtests mit Dokumentation und Rückmeldung an die Entwickler
  - Vorteile
    - größere Chance, früh mögliche Probleme zu entdecken, die beim Zusammenspiel der Komponenten auftreten
    - durch tägliches Erstellen werden Komponenten häufiger getestet und damit qualitativ besser
    - psychologischer Vorteil, da die Entwickler nicht Schuld am Scheitern der Systemerstellung sein wollen
- Bedarf nach Konfigurationsmanagement
  - Definition

    - KM ist „die Entwicklung und Anwendung von Standards und Verfahren zur Verwaltung eines sich weiterentwickelnden Systemprodukts“
  - KM ist wichtiger Bestandteil des SW-Engineerings
  - KM legt fest, wie Systemänderungen festgehalten und bearbeitet werden
  - Werkzeuge (CASE-Tools) unterstützen das KM
    - Speichern der unterschiedlichen Komponentenversionen
    - Erstellen der Gesamtsysteme
    - Möglichkeit, bestimmte Versionen wiederherzustellen
    - Verfolgen der freigegebenen Versionen
- vier Säulen des Konfigurationsmanagements



Begriffsabgrenzung

- Konfigurationsmanagement (KM) / Software Configuration Management (SCM)
- Konfigurationskontrolle
- Änderungsmanagement / Change Management (CM)
- Konfigurationseinheit / Configuration Item (CI)
- Configuration Baseline
  - bezeichnet die Konfiguration eines Produktes (Service, System, etc.) zu einem bestimmten Zeitpunkt
  - diese Konfiguration kann bei Bedarf zu jeder Zeit wieder hergestellt werden

Konfigurationsmanagement

- Ausprägung abhängig von Projektgröße, Beteiligten, geographischen Gegebenheiten, etc.
  - kleines „3-Mann-Projekt“: KM besteht evtl. nur aus dem Einsatz eines CVS und Richtlinien zur Verwendung
  - großes (~500), internationales Projekt: KM z.B. strikte Einhaltung von KM-Standards, mehr-sprachige Dokumentationen, etc.
- Planung des Konfigurationsmanagements
  - KM-Plan beschreibt die zu verwendenden Standards und Verfahren
  - Ausgangspunkt: allgemeine, firmenweite Standards für das KM
  - Anpassung an die jeweiligen Projekte
  - wichtig: Festlegung von Zeitplänen
    - Übergabe der Dokumente, Komponenten an die QS bzw. KM
    - Benennung von Reviewern, etc.

KM-Plan

- Inhalt eines KM-Plans (SOM04)
  - Definition und Bezeichnung der zu verwaltenden Objekte
  - Festlegung der Zuständigkeiten beim KM
  - Festlegung von Regeln für Änderungskontrolle und Versionierung
  - Beschreibung der aufzubewahrenden Aufzeichnungen
  - Beschreibung der einzusetzenden Tools und deren Verwendung
  - Definition der Konfigurations-DB
- Inhalt eines KM-Plans (IEEE98)
  - Introduction
    - Purpose (für wen ist der Plan und warum)
    - Scope (allgemeine Beschreibung, Identifikation von Cis, Anforderungen, Prämissen, etc.)
    - References (zu Richtlinien, Standards, Dokumente, etc.)
  - SCM Management
    - Organisation (funktionale Rollen der organisatorischen Einheiten in der Projekt-Struktur, Abhängigkeiten der Einheiten)
    - SCM Responsibilities (Zuordnung der SCM-Activities zu den Einheiten)
    - Applicable policies, directives & procedures (externe Beschränkungen und deren Einfluss)
  - SCM Activities
    - Identifizierung aller Funktionen und Aufgaben, die für das SCM notwendig sind
    - allgemeine, technische und Management-Aktivitäten
    - Gruppierung in Configuration Identification, Configuration Control, Status Accounting, Configuration Audits/Reviews
  - SCM Schedules
    - Koordination der identifizierten SCM Activities & aller im KM-Plan definierten Ereignisse
    - Zeitpunkte der Erstellung von Configuration Baselines, Change Control-Aktivitäten und Start/Ende von Configuration Audits
    - Definition von Projekt-Milestones
  - SCM Resources
    - Identifikation aller benötigten Ressourcen (Werkzeuge, Techniken, Ausrüstung, Schulung) für jeden Typ von SCM Activities
  - SCM Plan Maintenance
    - Anpassung des KM-Plans während des Projekt-Lebenszyklus
    - Festlegung von Zuständigkeiten, Zeitintervallen, Richtlinien
    - wird vor jeder Phase gereviewt und bei Bedarf angepasst, anschließend ans Team verteilt

Konfigurationseinheiten

- Identifikation von Konfigurationseinheiten
  - bei der SE entstehen sehr viele verschiedene Dokumente
    - Ideen, Skizzen, techn. Dokumente, interne Mitteilungen, Protokolle
  - des Weiteren existieren unterschiedliche Konfigurationseinheiten
    - Spezifikationen, Projektpläne, DBs, Anwender-Doku, Entwürfe, Testdaten, Code etc.
  - es muss entschieden werden, welche Dokumente bzw. CIs für die zukünftige Wartung/Wie-terentwicklung notwendig sein könnten, also der Konfigurationssteuerung unterliegen sollen
  - die Auswahl der CIs ist ein sehr wichtiger Faktor im Entscheidungsprozess des Projektmgmts
  - die Auswahl hat Einfluss auf Entwicklung, Kosten, Zeitplan, etc.

  - zu viele CIs führen zu übermäßiger Anzahl von Dokumenten und Spezifikationen
  - eine zu kleine Auswahl von CIs verhindern eine ausreichende Sicht auf das Gesamt-Design und Entwicklungsanforderungen
- Benennung von Konfigurationseinheiten
  - der KM-Plan sollte eine Spezifikation zur eindeutigen Identifizierung jeder Einheit (und deren Version) liefern
  - Identifizierungsverfahren könnten z.B. Namenskonventionen, Versionsnummer beinhalten
  - der KM-Plan sollte Methoden zur Benennung von CIs für die unterschiedlichen Gebiete beschreiben (Speicherung, Tracking, Verteilung, etc.)
  - herstellerspezifische Identifizierungsschemata und Bezeichnungen müssen ggfs. beachtet werden (insb. bei Kooperationen unterschiedlicher Partner)
- Erfassen der Konfigurationseinheiten
  - physikalische Speicherung der Konfigurationseinheiten der Baseline in Librarys
  - für jede Library sollte Format, Ort, Anforderungen, etc. spezifiziert werden
  - Angaben zu Backups, Disaster Prevention/Recovery sollten beschrieben sein

Konfigurations-DB

- Ziel: Protokollieren aller relevanten Informationen über die Konfiguration
- Bereitstellung der Verwaltungsinformationen zur Unterstützung der Beurteilung von Systemänderungen
- DB-Schema und Verfahren zur Protokollierung/Abfrage müssen festgelegt werden
- Idealfall: Integration der Konfigurations-DB in das Versionsmanagement
- typische Anfragen
  - An welchen Kunden wurde eine bestimmte Systemversion ausgeliefert?
  - Welche HW- und welche OS-Konfiguration ist für die Ausführung einer bestimmten Systemversion erforderlich?
  - Wie viele Systemversionen wurden (wann) erstellt?
  - Welche Systemversionen werden von der Änderung einer bestimmten Komponente beeinflusst?
  - Wie viele Änderungsanträge sind für eine bestimmte Version noch unerledigt?
  - Wie viele gemeldete Fehler sind in einer bestimmten Version vorhanden?

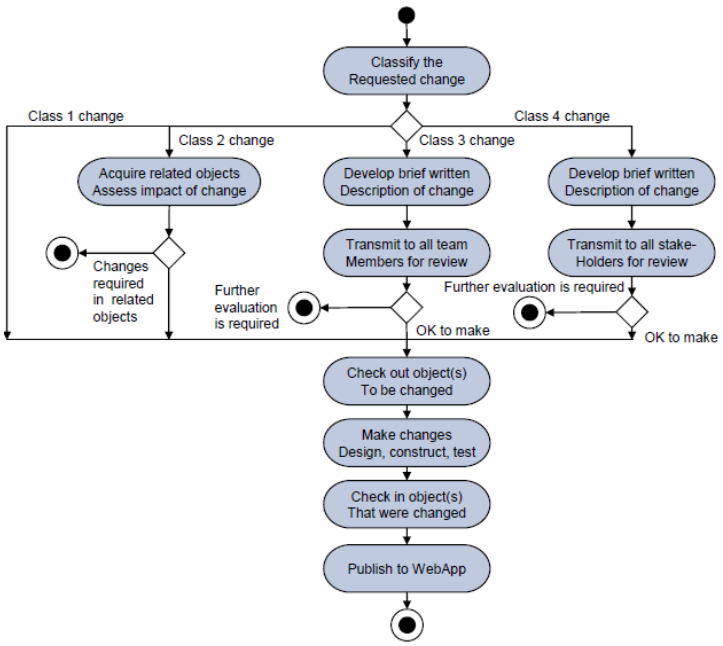
Konfigurationskontrolle

- unterstützt das Management von Änderungen im SW-Lebenszyklus
  - Prozess zur Ermittlung notwendiger Änderungen
  - Genehmigung bestimmter Änderungen
  - Unterstützung bei der Implementierung von Änderungen
  - Management von Sonderfreigaben (waivers) und Abweichungen (deviations) von den Projektanforderungen
- Aktivitäten
  - Änderungsantrag/-anfrage
  - Änderungen zustimmen oder ablehnen
  - Implementierung der Änderungen

  } Änderungen betreffen „baselined CIs“
  - Änderungen umfassen Fehlerkorrektur und Verbesserung
  - benötigte Formalitätsanforderungen abhängig von betroffener Projekt-„Baseline“ und von den Auswirkungen der Änderung auf die Konfigurationsstruktur

- Problem- und Änderungsmanagement
  - Ziel ist es, dokumentierte Problemmeldungen oder Änderungsanträge zu bewerten und zu beschließen
  - Gründe für Änderungsanforderungen: fehlende Funktionalität, technologische Probleme, Fehlverhalten des Systems, Veränderungen des eigenen Umfelds, Missverständnisse
  - Tätigkeiten
    - Erfassung, Verwaltung und Verfolgung eingehender Fehlermeldungen, Problemmeldungen und Verbesserungsvorschlägen in Form von Änderungsanforderungen
    - Bewertung und Entscheidung über die Bearbeitung von Änderungsanforderungen
    - Berücksichtigung technischer, finanzieller, organisatorischer und terminlicher Auswirkungen auf den Projektverlauf oder das System in der Nutzung

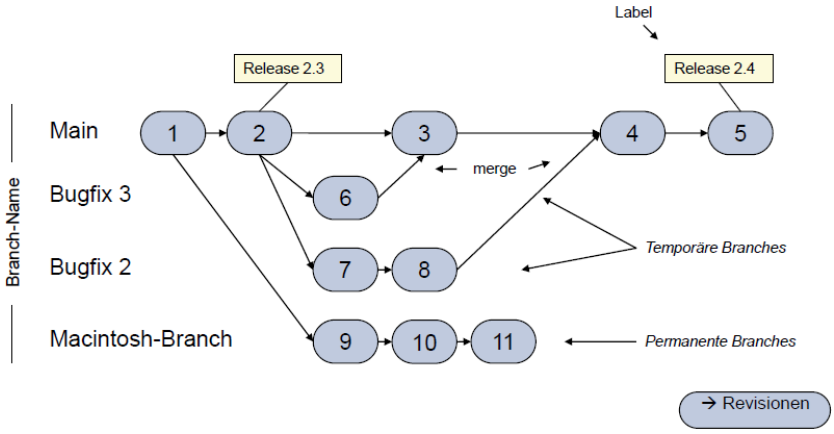
  1) Problemmeldung/Änderungsantrag erstellen
     - Informationen zum Antrag: Hinweise zur Identifikation, Beschreibung des Problems/der Änderung, Begründung des Änderungswunsches, evtl. Lösungsvorschlag
  2) Problemmeldung/Änderungsantrag bewerten
     - Problem analysieren, Lösungsvorschläge erarbeiten
  3) Änderungen beschließen
     - Vorbereitung der Entscheidung, Änderungsanträge und -bewertungen präsentieren, Entscheidung beschließen, Auswirkungen ermitteln, Entscheidung kommunizieren
  4) Änderungsstatusliste aktualisieren
     - betroffene Informationen: Änderungsstatus, Termine der Bewertung/Entscheidung/Umsetzung, Referenzen auf Änderungsbewertung/-entscheidung
- Configuration Control Board (CCB)
  - akzeptiert vorgeschlagene Änderungsanfragen oder weist diese zurück
  - Anzahl der Mitglieder abhängig von der Projektgröße
  - Befugnisse können abhängig von unterschiedlichen Kriterien (z.B. kritischer Zustand des Objekts, Art der Änderung) auf mehreren Ebenen erteilt werden
  - die Zusammenstellung der CCBs ist ebenfalls abhängig von unterschiedlichen Kriterien
- Änderungsmanagement in Web-Applikationen – Einteilung in Änderungskategorien
  - Class 1: Content- oder Funktionsänderung, die Fehler behebt oder lokale Inhalte und Funktionalität verbessert
  - Class 2: Content- oder Funktionsänderung, die Einfluss auf andere Content-Objekte und funktionale Komponenten hat
  - Class 3: Content- oder Funktionsänderung, die Einfluss auf die gesamte Web-Applikation hat
  - Class 4: bedeutende Designänderung (z.B. Schnittstellendesign, Navigationsstruktur) die verschiedene Benutzergruppen betrifft



Versionsmanagement

- Motivation
  - Möglichkeit, verschiedene Versionen und Varianten eines Dokuments zu sichern (Archivierung)
  - Möglichkeit der Wiederherstellung jeder Version eines Systems
  - Protokollierung aller Änderungen
  - gemeinsamer Zugriff mehrere Entwickler
  - gleichzeitige Entwicklung mehrerer Branches
  - Erhaltung der Konsistenz der Konfigurationsdatenbank
- Begriffe
  - System-Version
    - Ausprägung eines Systems, welche von anderen Ausprägungen abweicht (unterschiedliche Funktionen/Leistungen, Fehlerbehandlung, Auslegung für versch. HW/SW)
  - System-Release
    - Version, welche an Kunden ausgeliefert wurde
    - ein System-Release sollte neue Funktionen beinhalten, für andere HW entwickelt sein oder ähnliches
    - es existieren immer mehr System-Versionen als System-Releases
- große Anzahl von Komponenten können in verschiedenen Versionen vorliegen → Festlegung eines eindeutigen Weges, wie die Komponenten-Versionen zu bezeichnen sind
  - Versionsnummern
    - Komponente erhält eindeutige Versionsnummer
    - gebräuchlichstes Bezeichnungsschema
    - nicht zwingend linear
    - Nachteile: umfangreiches Informationsmanagement zur Verfolgung von Unterschieden zwischen den Versionen notwendig, Finden von speziellen Versionen kann schwierig sein
    - 6.6.0.13 → Baseline.Release.Patch.Build
  - attributbasierte Bezeichnung
    - jede Komponente besitzt einen Namen und eine Reihe von versionsunabhängigen Attributen (Sprache, Status, Plattform, …)
    - Kombination aus Namen und Attribute ergibt die System-Version
    - z.B. AC3D (Sprache = Java, Plattform = NT4, Datum = Jan1999)
    - Vorteile: Abfrage von Versionen über Attribute möglich, Hinzufügen einer neuen (abgeleiteten) Version ist einfacher
    - Nachteil: zusätzliches Änderungsmanagement nötig, um die Beziehung zwischen Version und Änderung herauszufinden
  - änderungsorientierte Bezeichnung
    - Benennung wie bei der attributbasierten Bezeichnung
    - zusätzliche Information über Änderungen
    - Kombination aus Namen und Änderungen ergibt die System-Version
    - wird meist für Systeme (nicht für Komponenten) verwendet
    - jede Änderung verfügt über einen (damit verbundenen) Änderungssatz
    - prinzipiell kann eine Version über einen bestimmten Satz von Änderungen erstellt werden
    - praktisch jedoch nicht möglich, beliebige Änderungssätze auf ein System anzuwenden (Konflikte, …)

- pessimistische Versionskontrolle
  - Prinzip: gleichzeitiges Bearbeiten von Dokumenten kann zu Konflikten führen und muss verhindert werden
  - der Entwickler muss zur Bearbeitung eines Dokumentes einen Lock anfordern und hat damit das „alleinige“ Recht zur Bearbeitung
  - Vorteile: keine Inkonsistenzen, kein Zusammenführen von gleichzeitig bearbeiteten Dokumenten nötig
  - Nachteil: keine gleichzeitige Bearbeitung von Dokumenten möglich
  - Beispiel: RCS (Revision Control System)
- optimistische Versionskontrolle
  - Prinzip: gleichzeitiges Bearbeiten von Dokumenten führt selten zu Konflikten und können (manuell) aufgelöst werden
  - mehrere Entwickler können gleichzeitig ein Dokument bearbeiten
  - Konflikte werden beim Einchecken festgestellt und müssen aufgelöst werden
  - Vorteil: gleichzeitiges Bearbeitung von Dokumenten möglich, Zeitersparnis
  - Nachteil: auftretende Konflikte müssen aufgelöst werden
  - Beispiel: CVS (Concurrent Version Control System)
- Zweige (Branches)



- CheckIn-/CheckOut-Verfahren
  - Dokumente/SW-Komponenten werden in einem Repository gespeichert
  - CheckOut
    - Kopie des gewählten Elements wird abgerufen und kann geändert werden
    - weitere CheckOuts können unterbunden werden (bzw. erzeugen einen neuen Zeig)
  - CheckIn
    - die Kopie wird als neue Version zurück in das Repository gespeichert
    - die alte Version bleibt erhalten
    - Delta-Methode zur Speicherung

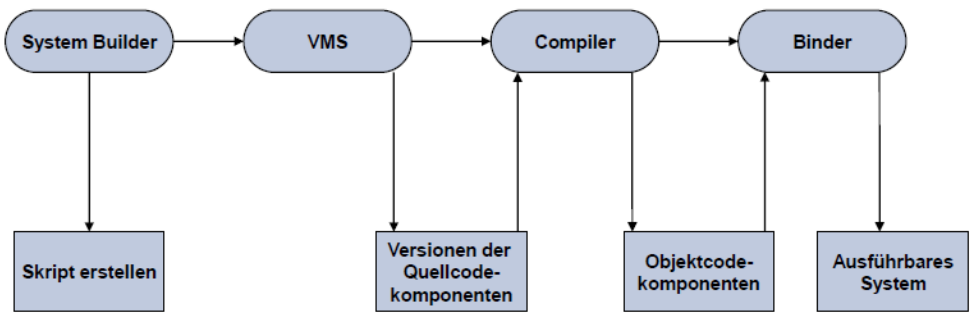
Release-Management

- Release = System-Version, die an den Kunden ausgeliefert wird
- Releases können u.a. folgende Elemente enthalten
  - Konfigurationsdateien, Datendateien, Installationsprogramm, Dokumentation (elektronisch/gedruckt), Verpackung/Werbung
- Release-Manager
  - entscheidet, wann ein Release freigegeben wird
  - verwaltet den Erstellungsprozess der Releases und Vertriebsdatenträger

  - dokumentiert das Release
- Problem der Release-Abhängigkeit
  - es kann nicht davon ausgegangen werden, dass alle Kunden immer alle Releases installieren
  - Bsp: Release2 wird nicht von allen installiert, Release3 braucht aber Datenfiles von Release2
- Einflussfaktoren auf die Strategie für die Freigabe von System-Releases
  - technische Qualität des Systems, Konkurrenz, Marketinganforderungen, Änderungsvorschläge von Kunden, Lehmans fünftes Gesetz (durchschnittliche Erhöhung der Funktionalität in einem neuen Release in etwa konstant)
- Release-Entscheidungen
  - Vorbereitungs-/Verteilungsprozess ist sehr aufwendig (insb. bei großen Stückzahlen)
  - für nicht kundenspezifische SW gilt:
    - zu häufige Releases → Kunden installieren möglicherweise nicht alle neuen Releases
    - zu seltene Releases → evtl. geht Marktanteil verloren, weil sich die Kunden für Alternativen entscheiden
- Release-Erstellung
  - Dateien/Dokumente, welche alle Komponenten enthalten werden erstellt und bilden das System-Release
  - Zusammenstellung und Bezeichnung von Programm-Code und Daten-Dateien
  - Bereitstellung sonstiger Informationen (Konfigurationsbeschreibungen, Anweisungen für Benutzer, Installationsanweisungen, Skripte, …)
  - Vertrieb (CD, Download, etc.)
- Release-Dokumentation
  - Dokumentation jedes Releases (evtl. zur Nachproduzierung), besonders wichtig bei kundenspezifischer SW
  - Inhalt der Dokumentation eines Releases
    - alle jeweiligen Versionen des Quellcodekomponenten
    - Wurden die richtigen Versionen der Komponenten aufgenommen?
    - alle Daten- und Konfigurationsdateien
    - Versionen des verwendeten OS, Compilers, Frameworks, …
  - evtl. Speicherung spezieller Dateien im Versionsmanagementsystem

System Building

- Kompilieren und Binden der SW-Komponenten zu einer ausführbaren Anwendung
- sog. Build Instructions stellen sicher, dass die Reihenfolge der Build-Prozesse eingehalten werden
- wichtige Fragestellungen
  - Wurden alle Komponenten aufgenommen?
  - Wurden die richtigen Versionen der Komponenten aufgenommen?
  - Wurden alle benötigten Daten-Dateien hinzugefügt?
  - Ist die richtige Version des Compilers und anderer Werkzeuge verfügbar?



Build Management

- Was gehört zu einem SW-Produkt? Quellcode, verwendete Komponenten, Dokumentation/Hilfe, Ressourcen (Bilder, Videos, Sounds), Sprachdateien – alles liegt in verschiedenen Versionen vor
- Charakteristika des Erstellungsprozesses
  - viele unterschiedliche Aufgaben: Abhängigkeiten prüfen/verwalten, Kompilieren, Testen, …
  - häufig wiederkehrende Aufgaben: Nightly Builds, Testläufe (z.B. Regressionstests)
  - verteiltes Arbeiten in Teams: individuelle Entwicklungsumgebungen, unterschiedliche Sprach- und Designeinstellungen, andere Treiber, …
- Lösungsansatz
  - durch den Einsatz formaler Methoden die Komplexität reduzieren (vgl. auch Konfigurations-management)
  - immer wiederkehrende Aufgaben durch Automatisierung effizienter gestalten
- Definition Build Management: Build Management umfasst alle Tätigkeiten zur Automatisierung des Erstellungsprozesses von SW-Produkten. Im Kern steht die automatisierte Abwicklung einer Vielzahl regelmäßig wiederkehrender Aufgaben zur Produktion eines Endergebnisses, das sogenannte Build. Ein Build ist dabei charakterisiert über eine eindeutige (fortlaufende) Build-Nummer und di e vollständige Reproduzierbarkeit.
- Werkzeuge
  - unix make: effizient und robust, aber umständliche Syntax und hoher Einarbeitungsaufwand
  - Apache Ant: verbreitet im Java-Bereich, Toolunterstützung, lesbare XML-Dateien, aber Automatisierung bloß i.e.S.
  - Maven: verbreitet im Java-Bereich, Framework zum Management von SW-Produkten (Kompilieren, Testen, Dokumentationserzeugung, Berichtsgenerierung, Distribution), ganzheitlicher Ansatz mit hohem Maß an Wiederverwendung
  - MS Build: verbreitet im .NET-Bereich, Integration in Entwicklungsumgebung, vereinfachter reproduzierbarer Erstellungsprozess, aber Automatisierung bloß i.e.S.

Software Configuration Status Accounting

- unterstützt das effektive Management der SW-Konfiguration
  - Software Configuration Status Information
    - Sammlung relevanter Informationen, die den Status der aktuellen Konfiguration betreffen
  - Software Configuration Status Reporting
    - Reports können von verschiedenen Organisationseinheiten verwendet werden (z.B. Entwicklung, Qualitätssicherung, Projektmanagement, Wartung)
    - ermöglicht Ad-hoc-Abfragen sowie vorgefertigte Reports
- Informationen des Status Accounting dienen als Basis für das Measurement (z.B. Change Requests pro SCI, durchschnittlich benötigte Zeit zur Implementierung einer Change Request)

Configuration Audit

- ergänzen formale technische Reviews
- Fragestellungen bei Configuration Audits zur Überprüfung von Änderung
  - Wurden die Änderungen übernommen? Gab es weitere Modifikationen?
  - Wurden technische Reviews durchgeführt?
  - Wurde der SW-Prozess eingehalten und wurden Standards richtig angewendet?

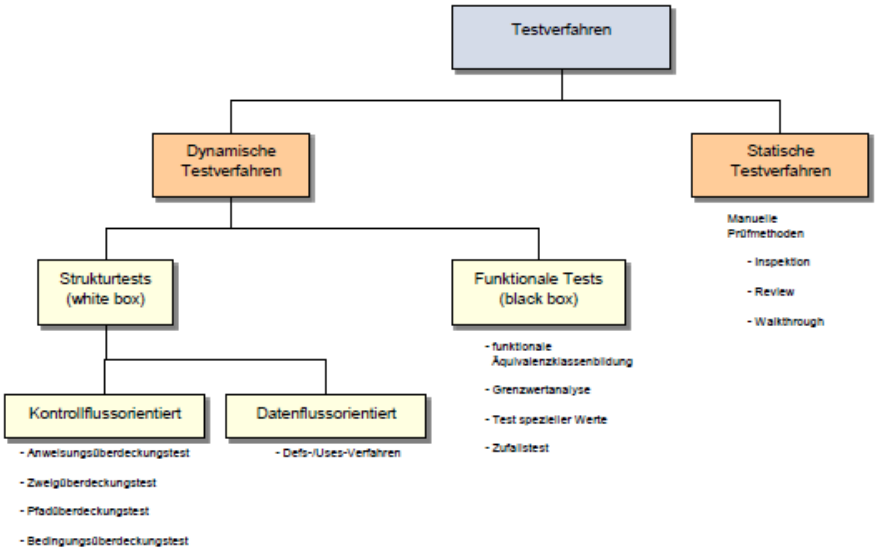
  - Wurden die Änderungen am SCI vermerkt? Wurden das Änderungsdatum und der Autor der Änderung festgelegt?
  - Wurden die SCM-Verfahren eingehalten?
  - Wurden all von der Änderung betroffen SCIs richtig aktualisiert?
- Arten von Configuration Audits
  - Software Functional Configuration Audit (FCA)
    - stellt die Konsistenz des SCI mit der zugrunde liegenden Spezifikation sicher
    - als Input dienen Ergebnisse der SW-Verifikation und -validierung
  - Software Physical Configuration Audit (PCA)
    - stellt die Konsistenz des Designs und der Referenzdokumentation mit der neu erstellten SW sicher
  - In-process Audits of a Software Baseline
    - Untersuchung spezifischer Elemente der Konfiguration während des Entwicklungsprozesses

Software Testing

Klassifikation analytischer Qualitätssicherungsverfahren

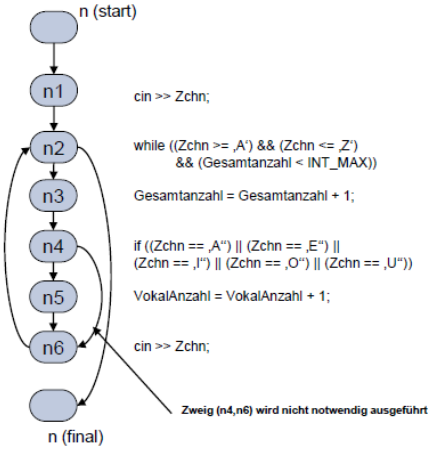
- testende Verfahren (→ Fokus der Vorlesung)
  - Ziel: Fehler erkennen
- verifizierende Verfahren
  - Ziel: Beweisen der Korrektheit einer Systemkomponente
  - „Zeigen, was ein Programm macht (automatisiert)“
- analysierende Verfahren
  - Ziel: Vermessung/Darstellung von bestimmten Eigenschaften einer Systemkomponente

Testverfahren

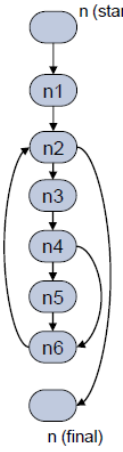


- statische Verfahren
  - beziehen sich ausschließlich auf den Quellcode der Anwendung
  - keine Ausführung des Programms
  - Durchführung in der Regel von einem Team durch manuelle Prüfmethoden
    - Inspektion, Walkthroughs, Reviews
  - untersuchte Bereiche

  - Berechnungen, Variablenverwendung, Schnittstellen
  - Schleifenterminierung, Steuerfluss, E/A-Fehler
- dynamische Verfahren
  - untersuchen den Ablauf der Anwendung
  - Betrachtung der Auswirkungen von festgelegten Eingaben
  - abhängig vom Bekanntheitsgrad des Quellcodes erfolgt eine Unterscheidung in Struktur- und funktionale Tests
- Strukturtests (Whitebox-Tests)
  - der Quellcode (Programmlogik) ist dem Tester bekannt
  - Untersuchung der internen Struktur des Programms
  - Arten von Whitebox-Tests (Verfahren zur Ermittlung der Testfälle)
    - Anweisungsüberdeckung: jede Anweisung (Auswahl/Falluntersch.) mind. 1x ausführen
    - Zweigüberdeckung: jeder Programmzweig (Schleife/Auswahl) wird mind. 1x durchlaufen



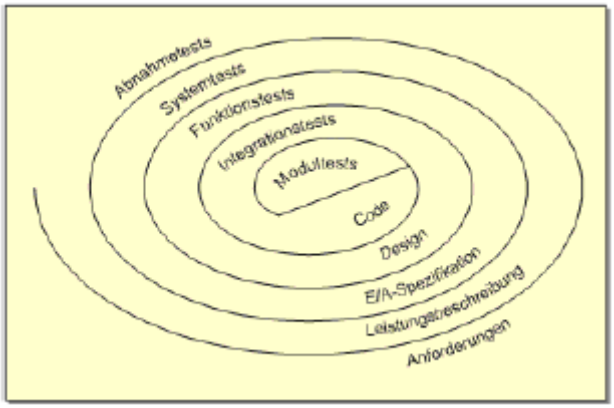
Anweisungsüberdeckung (alle Knoten)



Zweigüberdeckung (alle Kanten)

    - Pfadüberdeckung: jeder mögliche/denkbare (Pfad) durch das Programm wird mind. 1x durchlaufen
    - Bedingungsüberdeckung: jeder Bedingungsfall wird mind. 1x getestet
- funktionale Tests (Blackbox-Tests)
  - der Quellcode (interne Struktur) ist dem Tester nicht bekannt bzw. spielt keine Rolle
  - Entdeckung von Umständen, bei denen sich das Programm nicht der Spezifikation entsprechend verhält
  - Arten von Blackbox-Tests
    - funktionale Äquivalenzklassenbildung:
      - Voraussetzung: alle Werte einer Äquivalenzklasse erzeugen dasselbe Ergebnis
      - ermöglicht eine Verringerung der Testfallanzahl
      - Auswahl von Testfällen: an der Grenze von Klassen und in der Mitte
    - Grenzwertanalyse
      - Voraussetzung: Ordnung der Werte einer Äquivalenzklasse möglich
      - Grenzwerte (z.B. Maximum/Minimum, etc.=) dieser Ordnung werden getestet
    - Test spezieller Werte/Zufallswerte
      - es wird mit bekannten/kritischen Werten (z.B. extrem große/kleine Werte, willkürliche Eingaben) oder Zufallswerten getestet

testende Verfahren

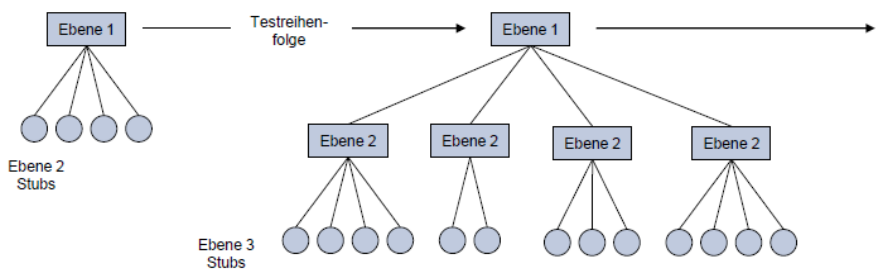


Modultests

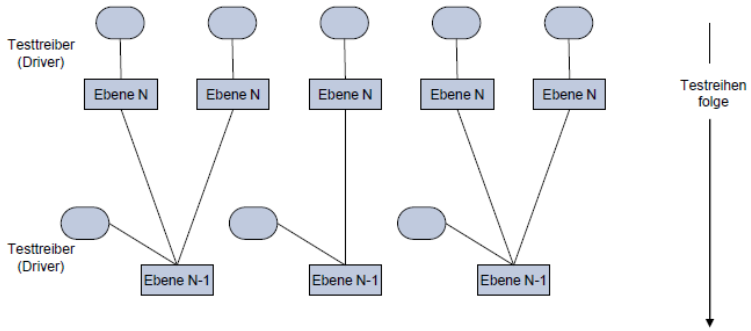
- beziehen sich auf kleinere Einheiten, wie z.B. Module, Komponenten, Unterroutinen
- in der Regel handelt es sich um Whitebox-Tests
- parallele Durchführung unterschiedlicher Module möglich
- Überprüfung von Schnittstellen, lok. Datenstrukturen, Randbedingungen, Pfade
- evtl. Ergänzung durch Blackbox-Tests

Integrationstests

- Untersuchung der Zusammenarbeit einzelner Einheiten
- Testen auf Fehler, die sich aus dem Zusammenwirken der Komponenten ergeben
- Verwendung von Driver und Stubs
  - Driver (Treiber) stellen die Umgebung für den Test zur Verfügung
  - Stubs simulieren als Stellvertreter die Funktionalität von noch nicht fertigen Einheiten
- Ansätze
  - Big-Bang-Ansatz
    - nicht inkrementelles Verfahren
    - Kombination aller SW-Einheiten zum Gesamtprogramm in einem Schritt
    - Testen des gesamten Programms → viele Fehler werden entdeckt
    - Isolation eines Fehlers ist sehr schwierig
    - empfiehlt sich nur für kleine (gut strukturierte) Programme
    - bei großen Programmen entsteht häufig Chaos
  - Top-down-Ansatz
    - inkrementelles Verfahren
    - zunächst werden die Systemkomponenten getestet, die in der Baumhierarchie bzw. Schichtenstruktur am weitesten oben stehen
      - bei einer objektorientierten Schichtenarchitektur beginnt die Integration bei der Benutzeroberfläche (bzw. Komponenten, die direkt mit der GUI kommunizieren)
      - anschließend werden die untergeordneten Einheiten in die Struktur aufgenommen



- Bottom-up-Ansatz
  - Integrationstests beginnen auf unterster Ebene bei den so genannten Terminalmodulen (Module, die keine anderen Module aufrufen)
  - es werden keine Stubs benötigt, da jedes übergeordnete Modul erst getestet wird, wenn das darunter liegende Modul bereits getestet wurde
  - Möglichkeit der Clusterbildung



- qualitativer Vergleich der Integrationstests

	Bottom-up	Top-down	Big Bang	Sandwich
Integration	früh	früh	-	früh
Fertigstellung der Anwendung	spät	früh	spät	früh
benötigt Treiber	ja	nein	nein	ja
benötigt Stubs	nein	ja	nein	ja
Parallelität	mittel	gering	hoch	mittel
Teste best. Pfade	leicht	schwer	leicht	mittel
Planung/Kontrolle der Reihenfolge	leicht	schwer	leicht	schwer

Regressionstests

- Aktivität, die sicherstellt, das Änderungen am System (z.B. Hinzufügen von Komponenten, Fehlerbehebung) kein unbeabsichtigtes Verhalten oder zusätzliche Fehler erzeugen
- Korrektheit des Tests wird nicht anhand der Spezifikation entschieden, sondern anhand der Ausgaben der aktuellen Version und der des Vorgängers
- Möglichkeiten der Durchführung von Regressionstests
  - manuell
  - erneutes Ausführen einer Teilmenge bereits durchgeführter Tests
  - automatisierte Capture-/Playback-Tools
- Regressionstestmenge (Teilmenge der bereits durchgeführten Tests) enthält drei Klassen von Testfällen
  - repräsentative Tests, die alle Funktionen abarbeiten
  - zusätzliche Tests, die Funktionen fokussieren, welche von Änderungen betroffen sind
  - Tests, die SW-Komponenten fokussieren, welche geändert wurden
- mit steigender Anzahl integrierter SW-Komponenten starker Anstieg der Regressionstests

Funktionstests

- Ziel: Aufdeckung von Unstimmigkeiten zwischen dem Programm & seiner externen Spezifikation
  - die Spezifikation enthält eine exakte Beschreibung des Programmverhaltens aus Sicht der Umgebung
- insbesondere Einsatz von Blackbox-Tests (Äquivalenzklassenbildung, Grenzwertanalyse)

Systemtests

- haben den Zweck, das System bzw. das Programm mit den ursprünglichen Leistungsbeschreibungen zu vergleichen
- getestete Eigenschaften (Auswahl)
  - Vollständigkeit, Zuverlässigkeit, Wartbarkeit, Kompatibilität
  - Volume
    - Grenzen der Belastbarkeit eines Systems werden getestet
    - Code wir durch große Mengen von Daten belastet
    - im Normalfall SW so ausgelegt, dass die Grenzen in der Praxis nicht erreicht oder überschritten werden
    - Problem: implizite Annahme der Entwickler, die von den Bedingungen in der Praxis abweichen
    - häufige Fehlerfälle: Pufferüberläufe, Programmabstürze, Endlosschleifen
  - Last
    - Belastung des Systems innerhalb eines kurzen Zeitraums (im Gegensatz zu Volume Test)
    - Belastung des Systems durch parallele Aktivitäten (Multiuser)
    - 2 Arten von Lasttests
      - Performancemessungen: mehrfache Wiederholung ausgewählter Testfälle unter Last um die Performanz einzelner Funktionen zu überprüfen
      - Lasttests im engeren Sinne: Testen von Prozessketten, um die Performanz mehrerer Funktionen im Zusammenspiel zu überprüfen
    - Ausprägungen von Lasttests
      - Stresstest: Lasttests im oberen Bereich, Austesten der System-Performanzgrenzen
      - Dauerlasttest: Lasttest über einen längeren Zeitraum
      - Fail-Over-Test: Überprüfung des Systemverhaltens unter Last bei Ausfall von Systemkomponenten, dient zur Überprüfung von Notfallszenarien (Einsatz von Zusatzressourcen)
  - Usability
    - wie angenehm, effektiv und funktional gestaltet sich die Interaktion eines menschlichen Benutzers mit einem technischen System
    - Konzept Usability
      - Fokus der Betrachtung liegt auf dem Anwender
      - Menschen setzen SW und Systeme ein, um ihre Produktivität zu steigern
      - es liegt am Anwender zu beurteilen, ob er leicht mit dem Produkt umgehen kann
    - Vergleich der tatsächlich eingesetzten mit den vorhandenen Funktion des Produkts
    - Resultate im Bereich Usability Tests häufig nicht eindeutig (da subjektive Eindrücke)
    - Forderungen, die die SW erfüllen muss
      - Einhaltung der Standards und Richtlinien, intuitives Verständnis der geforderten Aktionen durch den Benutzer, Konsistenz über das gesamte Programm, Flexibilität, Komfortabilität, Richtigkeit, Nützlichkeit
  - Recovery
    - Erzwingen von SW-Ausfällen über unterschiedliche Wege
    - verifiziert, dass die Wiederherstellung/Wiederaufnahme ordnungsgemäß durchgeführt wird
    - automatische Wiederherstellung → Evaluierung der Fehlerfreiheit bei
      - Reinitialisierung, Checkpoint-Mechanismen, Data Recovery, Neustart

        - manuell (menschlicher Eingriff) → Evaluierung der Mean-Time-To-Repair (MTTR)
            - MTTR muss innerhalb der akzeptablen Limits liegen
    - Sicherheit
        - verifiziert, dass die implementierten Sicherheitsmechanismen das System vor missbräuchlichen Angriffen schützt
        - der Tester spielt die Rolle des Eindringlings
        - verwendet verschiedene Angriffstechniken
            - DoS, Erzeugen von Fehlerfällen um einen Angriff zu ermöglichen, Durchsuchen ungesicherter Daten, Passwort-Sniffing, etc.

Abnahmetests

- beziehen sich auf den Bereich der Anforderungen
- sollen zeigen, dass sich das Programm so verhält, wie es in den Anforderungen spezifiziert wurde
- Abnahmetests werden für gewöhnlich vom Endbenutzer durchgeführt

objektorientiertes Testen

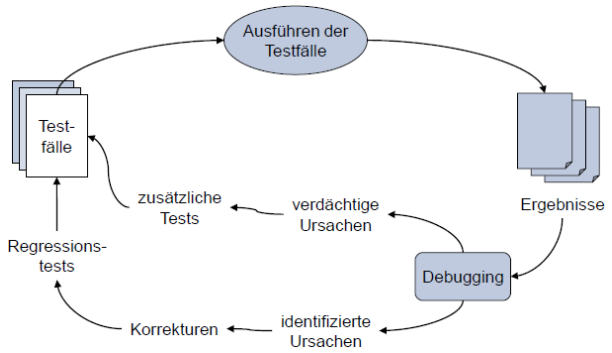
- Unterschiede zwischen objektorientierten Systemen und nach funktionalen Modellen entwickelte Systeme
    - Objektklassen sind als Einzelkomponenten oft größer als einzelne Funktionen
    - zu Subsystemen integrierte Objekte sind meist nur lose verbunden, und es gibt keine offensichtliche Systemhierarchie
    - bei Wiederverwendung von Komponenten steht den Testern zur Analyse möglicherweise kein Quellcode zur Verfügung

    → Entwicklung alternativer Ansätze
- Testebene objektorientierter Systeme
    - Testen der einzelnen, zu Objekten gehörenden, Funktionen
        - bei Funktionen und Prozeduren können Black-Box- und White-Box-Ansätze angewendet werden
    - Testen einzelner Objektklassen
        - Prinzip der Black-Box-Tests, aber Erweiterung von Äquivalenzklassen, um verwandte Operationsfelder zu behandeln
    - Testen von Objekt-Clustern
        - Black-Box und White-Box nicht geeignet, stattdessen szenariobasierte Tests
    - Testen des objektorientierten Systems
        - Verifikation und Validierung anhand der Systemspezifikation wie bei konventionellen Systemen
- Testen von Objektklassen
    - vollständige Testabdeckung
        - isoliertes Testen aller zum Objekt definierten Operationen
        - Setzen und Abfragen aller Attribute des Objekts
        - Ausführung des Objekts in allen möglichen Zuständen (Simulation von Ereignissen)
    - Einsatz von Vererbung erschwert den Entwurf für Objektklassen
    - untergeordnete Klassen sollten mit allen geerbten Operationen getestet werden
    - Konzept der Äquivalenzklassen bei Objektklassen
        - Beispiele: Tests, die dieselben Objektattribute verwenden, könnten der selben Äquivalenzklasse angehören

- Objektintegration
  - Cluster Testing: Gruppen von Klassen werden getestet, die zusammen arbeiten und eine Reihe von Diensten zur Verfügung stellen
  - weder Top-down- noch Bottom-up-Integration angemessen, da keine klare Hierarchie
  - mögliche Ansätze von Integrationstests
    - Anwendungsfall oder szenariobasierte Tests
    - Thread-Tests
    - Test der Wechselwirkungen zwischen Objekten

Debugging

- Aktion, die zur Behebung durch das Testen aufgedeckter Fehler durchgeführt wird
- Debugging kann als ein Prozess gesehen werden, bei dem die Symptome eines Problems/Fehlers einer Ursache zugeordnet wird
- die Beziehung zwischen der externen Erscheinungsform eines Fehlers und der internen Ursache ist häufig schwer ersichtlich
- Debugging-Prozess



- Warum ist Debugging kompliziert? Charakteristiken von Bugs:
  - Symptom und Ursache sind geographisch verteilt
  - das Symptom verschwindet (temporär) nachdem ein anderer Fehler korrigiert wurde
  - das Symptom wird nicht durch wirkliche Fehler ausgelöst (z.B. Rundungsfehler)
  - das Symptom ist das Ergebnis von Koordinierungsproblemen, nicht von Prozessproblemen
  - Schwierigkeiten bei der Reproduktion von Eingabebedingungen
  - Probleme treten nur sporadisch auf (häufig bei Embedded Systems)
  - Symptome werden durch Aufgaben verursacht, die über mehrere Prozessoren verteilt sind
- Debugging-Strategien
  - Brute Force
    - am häufigsten verwendet und sehr ineffizient, um die Ursache des Fehlers einzugrenzen
    - durch Tracing, Memory Dumbs und Ausgabeanweisungen werden viele Informationen erzeugt, aus denen möglicherweise die Ursache eines Fehlers ersichtlich wird
  - Backtracking
    - Zurückverfolgung des erkannten Symptoms über den Quellcode zu seiner Ursache
    - Problem: bei steigender Anzahl LOC sind die potenziellen Rückwärtspfade nicht mehr zu beherrschen
  - Cause Elimination
    - Daten, die in Verbindung mit dem Auftreten des Fehlers stehen, werden organisiert
    - es werden Hypothesen aufgestellt
    - die relevanten Daten werden benutzt, um die Hypothese zu beweisen / zu widerlegen
    - Alternative: Erfassen möglicher Ursachen und Eliminierung durch Tests

Rollen und Aktivitäten beim Testen

- Entwickler → Modultests
- Test-Team → Integratonstests, Systemtests
- Benutzer → Akzeptanztests, Abnahmetests

Testdriven Development (TDD)

- traditionelle SE: erst Anforderungen umsetzen, dann testen
- TDD: erst testen, dann die Anforderungen umsetzen
- TDD führt zu einer benutzerorientierten Programmierung
  - es wird erst darüber nachgedacht, wie eine bestimmte Funktionalität verwendet wird, bevor diese implementiert wird
- Mantra: „Rot, Grün, Refactor“
  - Rot: Schreibe einen Test, der fehlschlägt
  - Grün: Sorge dafür, dass der Test läuft (egal wie, aber schnell)
  - Refactor: Code verbesser/aufräumen (Wiederholungen entfernen, etc.)

NUnit

- kostenloses Test-Framework zur Durchführung von Unit-Tests
- unterstützt alle .NET-Sprachen, benutzerfreundliche Oberfläche, unkomplizierte Handhabung

Framework for Integrated Test (FIT)

- Testen nicht zum Selbstzweck → hoher Aufwand, Nutzen?
- Testen als Entwicklungsmodell
  - vgl. TDD, analog: Modelldriven, Architecturedriven
- Testen zur Qualitätssicherung
  - SW-Tests als Instrumente des Qualitätsmanagements
  - auch: vertragliche Aspekte im Zusammenhang mit Tests, z.B. Version x.y auslieferbar wenn 0 Kategorie 1 Tests „rot“, max. 2 Kategorie 2 Tests „rot“ und max. 10 Kategorie 3 Tests „rot“
- FIT ist ein Framework zur Automatisierung von Akzeptanztests
- Vorteile
  - einfach, vom Nutzer/Kunden zu erstellen
  - keine Offenlegung des Codes notwendig (Blackbox-Test)
  - hoher Automatisierungsgrad → Wirtschaftlichkeit (und impliziert dadurch eine höhere Qualität)
  - einsetzbar für Qualitätssicherung und TDD
  - generisch (auch „plattformunabhängig“)
- Nachteile
  - Framework bedarf Aufwand zur Anpassung
  - nicht für alle Arten von Tests geeignet
- Architektur