Software Enginnering für betriebliche Anwendungen

Summary

Kompiliert am: February 2, 2020

Konstantin Kuchenmeister

Name:	Konstantin Kuchenmeister
Matrikelnummer:	03717439
Universität:	Technische Universität München
Studiengang:	Wirtschaftsinformatik
Abschlussjahr:	2021

Contents

1 IT-Unterstützung betrieblicher Anwendungen

1.1 Klassifikation betrieblicher Anwendungen

Definition:
Eine betriebliche Anwendung ist die Gesamtheit aller Programme und die dazugehörigen Daten für ein konkretes betriebliches Anwendungsgebiet.

Betriebliche Anwendungen dienen...:

- Als Unterstützer und "Automatisierer" betrieblicher Prozesse
- Zur Umsetzung neuer Produkte und Ideen

Klassifikation nach Verwendungszweck

- **Administrations- und Dispositionssysteme:** Finanzbuchhaltung, Lohnabrechnung, Verwaltung von Beständen, Konten, Verträge
- **Führungsinformationssysteme:** Verwenden interne und externe Daten und stellen diese in flexibler Form bereit.
- **Querschnittsysteme:** Unabhängig von Einordnung in Unternehmenshierarchie → Nutzung über Schnittstellen mit anderen Administrations- und Dispositionsystemen.

1.1.1 Enterprise Resource Planning - ERP

- Integriertes Gesamtsystem, welches alle wesentlichen Funktionen der Administration, Disposition und Führung unterstützt.
- Bezeichnet damit also die Softwarelösung zur Ressourcenplanung (Wird in der Regel durch Standardsoftware realisiert.).
- **Funktionen:** Beschaffung, Produktion, Vertrieb, Personalwesen, Controlling.

1.2 Standard- und Individualsoftware:

Betriebliche **Individualsoftware**:

- ist für ein Unternehmen speziell entwickelt, und auf Anforderungen zugeschnitten
- wird individuell gepflegt und an Veränderungen angepasst
- ist das Ergebnis eines Projektes für einen bekannten Auftraggeber

Betriebliche **Standardsoftware**:

- ist für einen Markt entwickelt
- ist in mehreren Unternehmen einsetzbar
- wird individuell gepflegt und an Veränderungen angepasst
- wird von einem Softwarehaus entwickelt

1.2.1 Adaptionstechniken betrieblicher Standardsoftware:

Konfiguration
Bezeichnet Funktionalitäten und Techniken, die bei der Einführung nötig sind und es ermöglichen vordefinierte Einstellungen vorzunehmen die zu einer individuellen Variation der Standardsoftware führen.

Erweiterung
Bezeichnet Funktionalitäten und Techniken die für den produktiven Einsatz optional sind und es ermöglichen nicht vorhergesehene Anforderungen abzubilden die vom Hersteller implementiert wurden, um das Leistungsspektrum zu erweitern.

Kopplung
Bezeichnet Funktionalitäten und Techniken um externe Systeme anderer Hersteller anzubinden, Systeme des gleichen Typs anzubinden, die in Form von Schnittstellen vordefiniert sind.

1.2.2 Herausforderungen

Mandantenfähigkeit eines betrieblichen Informationssystemen bedeutet, dass mehrere Firmen in einem System abgebildet werden können, dass zwischen Mandanten -abhängigen und -unabhängigen Daten unterschieden wird, dass bestimmte Einstellungen nur für unterschiedliche Mandanten vorgenommen werden können.
Multilingualität eines betrieblichen Informationssystem ermöglicht es, Texte und ggf. Grafiken in verschiedenen Sprachen im System zu hinterlegen und anzuzeigen
Lokalisation: beinhaltet neben der Multilingualität lokalisierung von metrischen Daten, Zeitzonen, Währungen, Kalender...

1.3 Charakteristika betrieblicher Anwendungen

Anforderungen und Stakeholder
Anforderungen werden zu Beginn des Projekts sehr umfangreich durch Lastenheft und Pflichtenheft definiert. Die Anforderungen ändern sich im Laufe der Zeit und dienen als Basis bei der Projektplanung.

Persistenz
Dadurch das Daten für Unternehmen einen sehr hohen Wert besitzen ist Datenkonsistenz(Daten dürfen nicht verloren gehen)eine typische Anforderung.

Verteilung
Viele Benutzer greifen auf zentrale Daten zu → Client-Server Architektur sehr verbreitet, Kommunikation muss stets über Netzwerk erfolgen, Parallele z Zugriffe, Authentifizierung und Autorisierung nötig.

Integration
Anwendungslogik und Datenhaltung meist getrennt → Verschiedene Anwendungen arbeiten auf den gleichen Datenbanken, Anwendungen kommunizieren miteinander. (Anwendungslandschaften mit > 5000 Anwendungen möglich)

Herausforderungen: Integration verschiedener Programmiersprachen, Vermeidung von zu enger Kopplung.

Skalierbarkeit
Betriebliche Anwendungen können weltweit von Mitarbeitern benutzt werden(enorm hohe Anzahl von Anwendern), unterschiedliche Auslastung der Hardware.

Herausforderungen: Zeitversetzte Ausführung ressourcenintensiver Operationen, stetig stagnierende Anzahl von Nutzern, einzelner Server kann die Last nicht bearbeiten.

2 Requirements Engineering

Softwarelebenszyklus:
Requirements Engineering → System- und Softwareentwurf → Implementierung und Unit-Test → Integration und Systemtest → Betrieb und Wartung

Funktionale Requirements: beschreiben die Interaktionen zwischen dem System und seiner Umgebung unabhängig von deren Realisierung, lässt sich als Aktivität formulieren.

Nichtfunktionale Requirements: beschreiben Eigenschaften des Systems, welche nicht Interaktionen darstellen.

Constraints(Beschränkungen): bestimmen den Lösungsraum für die Realisierung e.g. Programmiersprache.

Lastenheft
Fachliches Ergebnisdokument der Anforderungsermittlungsphase. Vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrags.

→ Formulierung der Anforderungen sollte so allgemein wie möglich und einschränkend wie nötig sein.
Aufbau:

1. Zielbestimmung:
2. Produkt-Einsatz: Wofür? Für wen? Betriebsbedingungen?
3. Produkt-Umgebung: Software, Hardware, Orgware
4. Produkt-Funktionen: definiert Funktionalität aus Benutzersicht /F10/...
5. Produkt-Daten: Beschreibung langfristig zu speichernder Daten aus Benutzersicht /D10/
6. Produkt-Leistungen: Zeit, Umfang, Benutzerzahl, Genauigkeit,...
7. Qualitätsbestimmungen: Tabellarische Form
8. Glossar

Herausforderungen

- Verschiedene Interessensgruppen können widersprüchliche Anforderungen erheben
- Die Leute, die für das System zahlen, sind selten diejenigen, die es auch benutzen.
- Die Organisation und die technische Umgebung kann sich nach der Systemeinführung verändern.

Pflichtenheft
Vom Auftragnehmer erarbeitete Realisierungsvorhaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenhefts.

Das Was(Lastenheft) wird detailliert um das Wie ergänzt(Definiert den Zweck des Systems).
Dokumentierung in informeller, natürlicher Sprache.

Aufbau:

1. Zielbestimmung: Grundfunktion des Systems in natürlicher, informeller Sprache in einem Satz.
   (a) Muss-Kriterien: Was muss das Produkt leisten?(=Funktionale Anforderungen)
   (b) Wunsch-Kriterien: Was soll das Produkt evtl. noch leisten?
   (c) Abgrenzungskriterien: Was soll das Produkt nicht leisten?
2. Produkt-Einsatz:
   (a) Anwendungsbereiche: An welchen physischen Orten kann das System angewandt werden?
   (b) Zielgruppe
   (c) Betriebsbedingungen
3. Produkt-Umgebung: Nutzer des Systems in einem Satz beschreiben
   - Software: Client und Server Spezifikationen darstellen
   - Hardware: Hardware mit genauen Betriebssystemen darstellen
   - Orgware: Infrastruktur
4. Produkt-Funktionen: definiert Funktionalität aus Benutzersicht /F10/...
5. Produkt-Daten: Beschreibung langfristig zu speichernder Daten aus Benutzersicht /LDx0/

6. Produkt-Leistungen: Pflichten die das Programm zu erfüllen hat(=Nichtfunktionale Anforderungen) /LLx0/
7. Benutzeroberfläche: Mock-Ups, Screenshot
8. Qualitätsbestimmungen: Ergebnisinterpretation, Messung der Lösungsgüte
9. Testfälle: Mögliche Testszenarien darstellen /Tx0/
10. Entwicklungsumgebung: Code-Richtlinien, Entwicklungswerkzeuge(=Constraints)
11. Ergänzungen
12. Glossar

3 Konzeptionelle vs. Implementierungsnahe Modellierung mit UML

Konzeptionell vs. Implementierungsnah		
	konzeptionell	implementierungsnah
Sichtbarkeiten	Nein	Ja
Attribute mit Datentypen	Ja	Ja
Methoden	Nein	Ja
Vererbung	Sparsam	Ja
Abstrakte Klassen	Nein	Ja
Assoziationsklassen	Ja	Nein

1. **Klassen:**
   - Substantiv im Singular (kein Homonym oder Akronym)
   - so konkret wie möglich

   (a) Identifiziere/Unterstreiche relevante Substantive.
   (b) Identifiziere Synonyme, Homonyme(Ein Name der zwei verschiedene Konzepte benennt), Akronyme und kategorisiere Substantive
   (c) Streiche Substantive, die keine eigenständigen Klassen bezeichnen.
   (d)

2. **Attribute:**
   - Substantiv im Singular oder Plural (kein Homonym)
   - so konkret wie möglich

   (a) Identifiziere Attribute (meist Adjektive) → Selektion nach fachlicher Notwendigkeit

(b) Identifiziere Typ der Attribute

3. **Assoziationen:**

   (a) Identifiziere Verben und Substantive, die Aktionen oder Prozesse darstellen

   (b) Identifiziere für jede Assoziation die beteiligten Klassen

   (c) Bestimme Multiplizitäten.

4 Aufwandsschätzung

Ziel: Erwartete Aufwände und Kosten für ein Softwareprojekt vor der Durchführung bzw. zu einem möglichst frühen Zeitpunkt vorhersagen.
→ Aufwandsschätzungen erfolgen im Laufe eines Projekt mehrmals mit unterschiedlichem Detaillierungsgrad.

Einflussfaktoren der Aufwandsschätzung

1. Quantität: Größe des Programmtextes, Funktions- und Datenumfang, Komplexität (leicht, mittel,schwer)
2. Qualität: Je höher die Qualitätsanforderungen, desto höher der Aufwand (Setzt sich aus vielen Qualitätsmerkmalen zusammen, zudenen sich Kennzahlen zuordnen lassen.)
3. Produktivität: Lernfähigkeit und Motivation von Mitarbeitern von Mitarbeitern, viele weitere Faktoren.
4. Entwicklungsdauer: Weniger Entwicklungsdauer → Mehr Projektarbeiter → Höherer Kommunikationsaufwand.

4.1 Methoden zur Aufwandsschätzung

4.1.1 Schätzmethoden

Top-Down-Strategie: Schätzung des gesamten Projektaufwands mit Hilfe von mathematischen Algorithmen auf Basis der funktionalen Anforderungen.

Bottom-Up-Strategie: Aufwaende jedes Aufwandspostens werden getrennt ermittelt und zum Gesamtprojektaufwand summiert.

Vergleichsmethode allgemein:
Auf Basis der Aufwandsanalyse bereits durchgefuehrter aehnlicher Entwicklungen im eigenen Unternehmen bzw. der gleichen Branche wird auf zu erwartenden Aufwand des zu schaetzenden Softwareprodukts geschlossen.

Algorithmische Methode allgemein:
Erwarteter Entwicklungsaufwand fuer Softwareprodukt wird mit algorithmischen Methoden (einer geschlossenen Formel) berechnet.
Herleitung dieser Formel erfolgt auf Grundlage mathematischer Modelle bzw. des IST-Aufwands bereits abgeschlossener Projekte.

Kennzahlenmethode allgemein:
Durch Hochrechnung des Aufwands fuer einzelne ermittelte Leistungseinheiten oder Projektphasen wird auf den Gesamtaufwand des Softwareprodukts geschlossen.

1. 1 Vergleichsmethoden Analogiemethode
Vergleich der zu schätzenden Entwicklung mit bereits abgeschlossenen, ähnlichen Produktentwicklungen anhand von Einflussfaktoren. (grob)

Mögliche Kriterien: Ähnliches Anwendungsgebiet, ähnlicher Produktumfang, ähnlicher Komplexitätsgrad, Programmiersprache

Nachteile: fehlende allgemeine Vorgehensweise und Nachvollziehbarkeit

1.2 Vergleichsmethoden Relationenmethode
Vergleich der zu schätzenden Entwicklung mit bereits abgeschlossenen, ähnlichen Produktentwicklungen anhand von Einflussfaktoren die in Punkten ausgedrückt sind. (grob)
Aufsummierung der Punkte wird mit denen des anderen Projekts verglichen.

2.1 Algorithmische Methoden Gewichtungsmethode

Einflussfaktoren für das Projekt werden festgelegt, diese werden subjektiv oder objektiv gewichtet und anschließend dann nach vorgegebener mathematischer Formel verknüpft und ergeben dann den Gesamtaufwand. (genauer)

2.2 Algorithmische Methoden Stichprobenmethode
Es werden beispielhaft einige Funktionalitäten des Softwareprojekts implementiert und deren Aufwand festgehalten.
Stichprobenaufwand wird dann mit Anzahl ähnlicher Produktanforderungen multipliziert und addiert. (genauer)

3.1 Kennzahlenmethode Multiplikatormethode
Statt Entwicklungsaufwand werden Kosten für das Softwareprodukt berechnet, welche in verschiedene Arten eingeteilt werden. (genauer)
Pro Produktanforderung wird für jede Kostenart eine Kennzahl fixiert. Kosten = Kennzahl je Kostenart und Leistungseinheit * Anzahl der einheitlichen Leistungseinheiten.

3.2 Kennzahlenmethode Prozentsatzmethode
Nach Schätzung einer Phase durch eine andere Schätzmethode wird der Aufwand für die übrigen Phasen berechnet. Es wird anfangs erst festgelegt wieviel Prozent diese Phase ausmachen wird. (grob)

4.1.2 Konkrete Verfahren

Funktion Point Methode

Durchführung:

1. Kategorisierung jeder Anforderung:
   (a) Eingabendaten
   (b) Abfrage
   (c) Ausgabedaten (Abfrage und Zusatzoperation)
   (d) Datenbestände (intern)
   (e) Referenzdaten (extern) (=Produktdaten des Lasten/Pflichtenhefts)
2. Klassifizierung der Produktanforderungen(einfach, mittel, komplex)
3. Eintrag in Berechnungsformular (Kategorie, Anzahl, Klassifizierung, Gewichtung, Zeilensumme)
4. Bewertung der Einflussfaktoren
5. Berechnung der bewerteten Function Points: Summe der Kategorien*(Summe der 7 Einflussfaktoren/100 + 0.7)
6. Ablesen des Aufwands in Personenmonaten (FP-PM Kurve)
7. Aktualisierung der empirischen Daten

Vorraussetzungen:
Bewertung kann erst durchgeführt werden, wenn die Projektanforderungen bekannt sind, das gesamte Projekt sollte im Blickfeld sein, Sicht des Auftraggebers.

Vorteile:

- Anpassbarkeit an neue Anwendungsbereiche und Techniken
- Anpassbarkeit an unternehmensspezifische Verhältnisse
- Verfeinerung der Schätzung entsprechend dem Entwicklungsfortschritt: (1. Lastenheft, 2. Pflichtenhefts, 3. Nach Erstellung des formalen Modells)
- Gute Schätzgenauigkeit

Nachteile:
Es kann nur der Gesamtaufwand geschätzt werden → Umrechnung auf einzelne Phasen muss mit der Prozentsatzmethode erfolgen, zu stark funktionsbezogen, Qualitätsanforderungen werden nicht berücksichtigt.

5 Konfigurationsmanagement

Konfigurationsmanagement ist ein systematischer, werkzeuggestützter Ansatz zur Kontrolle der Evolution von Software in Entwicklung.

5.1 Version Management

Verwaltet alle Ergebnisse/Zwischenprodukte, die im Projektverlauf entstehen.

Manuell erzeugte Elemente: Programmquellen, Dokumente wie Testfallbeschreibungen, Projektplanung, Spezifikationen
→ Alle manuell erstellten Elemente versionieren.

Generierte Elemente: sind theoretisch aus manuell erstellten Elementen ableitbar, ausführbare Programme, Zwischenprodukte(generierte Dokumentationen)

Version Management Versionierung von Elementen

- Überarbeitungszustand eines Elements zu einem bestimmten Zeitpunkt kann identifiziert und rekonstruiert werden. (Ältere Elemente heißen Revision)
- Eine neue Version wird angelegt, wenn der Überarbeitungszustand festgehalten werden soll

- Eine einmal erstellte Version sollte nicht mehr verändert werden.
- Jede Version hat eine eigene Namen (z.B. fortlaufende Nummer)
- Ersteller und zugehörige Änderungsanforderung werden auch gespeichert

Es werden entweder nur die Änderungen (schwaches Datenwachstum) oder die komplette neue Dateiversion gespeichert.

Subversion
Subversion protokolliert Veränderungen an einzelnen Dateien (Deltas) zwischen Projektversionen.

1. Check-Out: Entwickler holen sich Elemente von anderen Entwicklern in Arbeitsbereich.
2. Beteiligte Personen bearbeiten in ihrem Workspace
3. Update: Personen passen ihren Arbeitsbereich von Zeit zu Zeit an den Stand im öffentlichen Repository an.
4. Check-In: Entwickler fügen neue Elemente zur Veröffentlichung zu Repository
5. Commit: Personen spielen aktuellen Stand ihres lokalen Arbeitsbereichs in das Repository zurück.

Verteiltes Versionsmanagement Git
Eine Version ist ein vollständiger Snapshot aller Dateien. Von allen Dateien werden Hash-Werte berechnet und Versionen verwalten Listen von Hash-Werten. Dadurch muss jede version einer Datei nur einmal gespeichert werden.

- **git add:** Fügt die modifizierten Dateien der Staging-Area hinzu (Datei wird in den nächsten commit aufgenommen)
- **git commit -m:** Bestätigt Dateiänderungen und fügt Dateien in die Historie hinzu.
- **git push:** Committete Änderungen werden in das Remote Repository übertragen.
- **git fetch:** Änderungen am Remote-Repository werden in (Commit-Area) übertragen.
- **git pull:** Änderungen am Remote-Repository werden abgerufen, heruntergeladen und im lokalen Repository zusammengeführt.
- **git checkout:** Wechselte oder erstellt eine neue Branch.

5.2 Build Management

Das Erzeugen ablauffähiger Software aus Programmquellen erfordert eine Reihe von Arbeitsschritten (Kompilieren, Packen, ...) Dieser Prozess wird als Produktionskette bezeichnet. (make, ant Java Version)

Typischer Aufbau eines ANT-Deployment-Skriptes

1. Kompilieren
2. Dokumentation erstellen
3. Tests Ausführen
4. Projekt in .jar Datei packen
5. .jar Datei auf Zielserver deployen

Beispielscode:

```
<project name="project-name" default="main">

<property name="folder.dir" value="folder"/>
<property name="src.dir" value="src"/>
<property name="classes.dir" value="bin"/>
<property name="jar.dir" value="${classes.dir}/jar"/>
<property name="doc.dir" value="doc/"/>
<property name="lib.dir" location="lib"/>
<property name="build.dir" value="build"/>
<property name="jar.dir" value="${build.dir}/jar"/>

<property name="wildfly.dir"
    location="../../../konstantinkuchenmeister/Downloads/wildfly-18.0.1.Final"
    />

<target name="compile">
<mkdir dir="${classes.dir}"/>
<javac srcdir="${srcdir}" destdir="${classes.dir}"
    includeatruntime="false"/>

</target>

<target name="javadoc" depends="compile">
<mkdir dir="${doc.dir}"/>
<javadoc destdir="${doc.dir}">
<fileset dir="./src"/>
</javadoc>
</target>

<target name="jar" depends="javadoc">
```

```
<mkdir dir="${jar.dir}"/>
<jar destfile="${jar.dir}/${ant.project.name}.jar"
    basedir="${classes.dir}">
<manifest>
<attribute name="Main-Class" value="${main-class}"/>
</manifest>
</jar>

</target>

<target name="deploy-server" depends="jar">
<copydir src="${lib.dir}"
    dest="${wildfly.dir}/standalone/deployments"></copydir>
</target>




</project>
```

Apache Maven vs. Apache Ant:

- Maven basiert auf der Annahme, dass Software immer dieselben Lebenszyklen durchläuft: validate, compile, test, package, verify, install, deploy
- Die Konfigurationsdatei von Maven ist deutlich strikteren Regeln unterworfen als die von ANT, was mehr einer Programmiersprache entspricht.
- Maven ist erweiterbar durch Plugins und das Project-Object Model ist die zentrale Konfigurationsdatei eines Projekts (pom.xml)

5.3 Release Management

Jedes Softwarepaket, das an den Auftraggeber ausgeliefert wird, heißt Release. Beinhaltet Stückliste mit allen ausgelieferten Elementen inklusive Versionsnummern und Anleitung zur Inbetriebnahme. (Stücklist+Anleitung = Konfiguration)

5.4 Change Management

Das Change-Management befasst sich mit der geordneten Abwicklung aller Änderungsanforderungen an der produktiven Anwendung.

Beschreibung einer Änderungsanforderung: Eindeutige Bezeichnung, Name des Meldenden, Prioritäten aus der Sicht des Meldenden, Informationen zur Systemumgebung.

5.5 Integration von Software- und Datenevolution:

Probleme:

- Betriebliche Systeme werden häufig geändert
- Die Systeme sind verteilt und besitzen Abhängigkeiten untereinander
- Die Änderungen müssen mit bestehenden Produktivdaten zurechtkommen

Konsequenz:

- Änderungen müssen ausreichend getestet sein
- Migrationsstrategien für bestehende Daten und Konfigurationen müssen entwickelt werden
- Abhängigkeiten von Systemen untereinander müssen dokumentiert sein.

6 Technische Grundlagen betrieblicher IS

Schichtenarchitektur:
Schichten unterteilen die Komponenten einer Software verringern dadurch die Komplexität. (high cohesion, low coupling) Eine Schicht darf auf alle unter ihr liegenden Schichten zugreifen.

- Strikt: Eine Schicht darf nur auf die direkt unter ihr liegende Schicht zugreifen → Einfachere Wartung
- Offen: Eine Schicht darf auf alle unter ihr liegenden Schichten zugreifen. (Bessere Performance).

6.1 Verteilte Softwaresysteme

Der Client fragt einen Dienst beim Server ab und erhält eine Antwort.
Ein Server bedient mehrere Clients wobei ein anderer Client selbst ein Server sein kann.

Web Server
Stellt seinen Clients statische oder dynamisch generierte Inhalte über HTTP/HTTPS bereit(HTML, CSS, Dateien, Bilder).

Kann mit Nginx, Apache Tomcat oder Jetty implementiert werden.

Anwendungsserver
Unterscheiden sich nach Typ: Java EE, .NET, SAP Web Application Server.

Aufgaben: Nachrichtenversand, Authentifizierung, Asynchrone Kommunikation, Kapselung von Datenbanken, Transaktionshandling.

Datenbankserver
Softwareseite:
Beinhaltet eine Datenbanksoftware: Relational, XML, NoSQL
Bietet Tools zum Administrieren.

Hardwareseite: Laufen auf eigener Maschine, nehmen die Rolle des Servers im Client-Server Modell ein.

JEE Architekturüberblick

- **Client Machine:** Java EE Application(s) - Client Tier
- **Java EE Server:**
  - Web Tier: JSP Pages
  - Business Tier: Enterprise Beans
- Database Server: Database - EIS Tier

6.2 Bibliotheken und Frameworks

Eine **Bibliothek** ist eine wiederverwendbare Softwarekomponente welche aus einer Vielzahl von Klassen bestehen kann. Die Funktionen können über die API genutzt werden.
Die Bibliotheken liegen in Form von .jar Dateien vor.

Ein **Framework** ist ein halbfertiges Softwaresystem, welches aus einer Vielzahl von aufeinander abgestimmten Softwarekomponenten besteht, aus mit relative geringem Aufwand ein angepasstes Softwaresystem erstellt werden kann.
→ bietet eine Basisarchitektur
→ eine hohen Grad an Reuse und eine gegebene Reihenfolge von Funktionen die der Benutzer erweitern kann.

Beispiel: JUnit

Unterschied Inversion of Control
"Don't call us, we'll call you(Hollywood Prinzip).

- Der Programmierer registriert seinen Code beim Framework, welches den registrierten Code des Programmierers aufruft, und die Lebenszyklen kontrolliert.

Vor-und Nachteile von Frameworks:
Vorteile:

1. Wiederverwendung von Design und Implementierung möglich.

2. Schnellere Entwicklung möglich
3. Weniger Fehler durch erprobte Mechanismen
4. Standardisierung

Nachteile:

- Hoher Einarbeitungsaufwand
- Programmiersprache und Umgebung streng vorgegeben
- Nur für einen bestimmten Problembereich anwendbar
- Hoher Entwicklungsaufwand zur Framework-Entwicklung
- Frameworks lassen sich schlecht kombinieren.

6.3 Aspektorientierung und Java-Annotationen

Aspektorientierte Programmierung ist ein Programmierparadigma um generische Funktionalität über mehrere Klassen hinweg zu verwenden.

- Interceptors: Unterbrechen den Programmablauf. Können before, after oder around Methoden zulassen.
- Joinpoints: Mögliche Stellen, an denen man Interceptors ausführen kann, z.B Methodenausführung, Objekterstellung oder Exception
- Pointcut: Ein Pointcut kann mehrere Joinpoints beinhalten, z.B. Ausführung jeder Methode der Klasse Kunde, deren Name mit "set" anfängt.
- Advice: Ein Advice beinhaltet den auszuführenden Code.
- Aspect: Ein Aspect fasst Pointcuts und Advices zusammen.

Reflexion:
Ein Vorgang, bei dem ein Programm auf Informationen zugreift, die nicht zu den Daten des Programms, sondern zur Struktur des Programms selbst gehören. Diese Reflexionen können jedoch nicht modifiziert werden.

6.4 Serialisierbarkeit

Durch Serialisation wird ein der Zustand eines Objekts persistent gemacht. Der Zustand wird also in ein Byte-Stream verwandelt. Funktioniert analog auch in von Stream zu Objekt. Dazu werden Atrributwerte, Typen, Objekttypen und alle referenzierten Objekte benötigt.
→ Das Interface Serializable muss implementiert werden.

7 Verteilung

Ein Verteiltes System ist ein System mit mehreren Prozessräumen in welchen wiederum mehrere Prozesse ablaufen können. Die einzelnen Subsysteme kooperieren dabei koordiniert um eine gemeinsame Aufgabe zu lösen.

→ Client-Server Modell **Vorteile:**

- Performancegewinn durch Parallelisierung
- Erhöhung der Rechenleistung
- Ausfallsicherheit durch Redundanz
- Integration existierender Funktionalität.

Nachteile:

- Erhöhung der Komplexität des Systems
- Entstehung von Sicherheitslücken
- Aufwändigeres Testen

Charakteristika:

- **Skalierbarkeit:** Erweiterung des Systems durch neue Bestandteile oder Komponenten, Erhöhung der Komplexität.
- **Fehlertoleranz:** Funktionalität trotz begrenzter Zahl defekter Subsyteme.
- **Transparenz:** Verteilte Struktur wird vor dem dem Benutzer verborgen und Erscheint als Einheit. (Heute nicht mehr das Ziel)

7.1 Kommunikationsformen

1. Zugriff auf einen gemeinsamen Speicher
2. Auslösen von Ereignissen in einem entfernten System
3. Austausch einer Sequenz von Nachrichten in einem Nachrichtenstrom
4. Durchführen eines entfernten Methodenaufrufs
5. Nutzung einer entfernten Komponente

Initiation über push (Senderprozess initiiert die Kommunikation mit dem Empfängerprozess) und pull (Empfängerprozess fragt nach vorliegenden Aufrufen)

Kopplung der Abläufe asynchron(Sender und Empfängerprozess haben gekoppelten Ablauf) oder asynchron (Senderprozess wartet nicht auf Empfängerprozess).

Kommunikation über gemeinsamen Speicher

Senderprozess schreibt einen Wert in gemeinsamen Speicher, Empfängerprozess prüft regelmäßig nach Änderungen im gemeinsamen Speicher (pull).
→Blackboard Architektur: gemeinsamer Speicher entspricht "schwarzem Brett"

Ereignisbasierte Kommunikation
Client kann Ereignistypen (autonome, asynchrone Begebenheit) angeben, über die er benachrichtigt werden möchte und registriert diese beim Server.(push)

Kommunikation mittels Nachrichtenstrom

Senderprozess erzeugt eine Nachricht, kodiert diese zur Versendung an den Empfänger und adressiert den Empfänger der Nachricht. Empfängerprozess dekodiert eingehende Nachrichteninformationen und erhält eine Kopie der ursprünglichen Nachricht.

Kommunikation über Methodenaufruf
Der Client ruft eine entfernte Methode wie eine lokale Methode auf.

- **Marshalling:** Umwandlung von strukturierten Argumenten bzw. Ergebnisse in lineare Nachrichten.
- **Unmarshalling:** Umwandlung von linearen Nachrichten in strukturierte Arugumente bzw. Ergebnisse.
- **Proxy** und **Skeleton** werden von Middleware bereitgestellt und codieren die Nachrichten.

Client und Server können dabei unabhängig voneinander abstürzen und es können Nachrichten verloren gehen.

Der Client sendet eine Anfrage und wartet bis die Antwort eintrifft, oder eine bestimmte Zeitspanne abläuft:

- **Maybe:** Der Client wiederholt bei Zeitablauf seine Anfrage nicht.
- **At least once:** Der Client wiederholt bei Zeitablauf seine Anfrage
- **At most once:** Der Client wiederholt seine Anfrage bei Zeitablauf und markiert sie als Wiederholung.

Kommunikation über entfernten Methodenaufruf
Implementierung mit Java RMI
Entfernter Aufruf und Objektübergabe über ein Stub (Schnittstelle entspricht der des entfernten Objekts)
↓ Client und Server benutzen Java

Kommunikation mit entfernten Komponenten
Eine Komponente

- hat eine definierte Schnittstelle
- hat einen persistenten Zustand
- kann nebenläufig aufgerufen werden
- hat einen eigenen Adressbaum
- kann dynamisch angelegt werden
- ist selbstbeschreibend.

Der Java EE Container

- ist die Laufzeitumgebung für Java EE Anwendungen/Komponenten
- stellt den Funktionsumfang der Java EE API zur Verfügung

Kommunikation entfernt Enterprise Java Beans (EJB)

EJBs kapseln Applikaitonslogik in einer verteilten Anwendung
Der Container bildet damit die Schnittstelle (Interface).
Der Client greift nie direkt auf die EJBs zu.
Das Interface kann Local oder Remote sein, die EJB Klasse implementiert dann das Interface
Der Zugriff auf das Remote Interface erfolgt über lookup(JNDI Namen)

Session Beans: (synchron, asynchron)

Namenskonventionen:
Remote Interface: $< bean - name >$
Local Interface: $< bean - name > Local$
Bean-Klasse: $< bean - name > Bean$

- **Stateful Session Beans:** sind mit einem bestimmten Client assoziiert, interner Zustand bleibt während der kompletten Client-Session erhalten, müssen Activation und Passivation behandeln
- **Stateless Session Beans:** kann auch mehrere verschiedene Clients bedienen, Zustand bleibt nur für einen Methodenaufruf erhalten, alle Instanzen einer Stateless Session Bean sind äquivalent, keine Activation oder Passivation
- **Singleton Session Beans:** Es existiert nur eine Instanz einer Singleton Session Bean pro Anwendung, kann auch mehrere Clients bedienen, behalten Zustand über mehrere Aufrufe aber nicht über Crashes hinaus.
- **Message Driven Beans:** (asynchron) reagieren auch Nachrichten, die in eine Warteschlange eingehen und haben keinen internen Zustand sonder verarbeiten externe Nachrichten.

Remote Interface Eigenschaften

- kann von überall aufgerufen werden.
- Aufruf erfolgt mittels Verteilungstechnologie
- Methodenparameter müssen java.io.Serializable implementieren
- Methodenparameter werden als Kopien übergeben (call-by-value)

Local Interface Eigenschaften

- Local Interface kann nur von Komponenten in derselben Deployment-Einheit benutzt werden (EJBs muessen co-located sein)
- Aufruf erfolgt mittels direktem Java Methodenaufruf
- Methodenparameter koennen direkt uebergeben werden (call-by-reference)

EJB-Klasse Implementierung

- (legt über @RemoteBinding fest, welche EJB Interfaces zugeordnet sind)
- legt den JNDI Namen fest
- muss einen StandardKonstruktor haben

7.2 Namenskonventionen

7.2.1 JNDI

JNDI ist eine einheitliche Schnittstelle zu Namensdiensten im Java Umfeld und assoziiert benutzerfreundliche Namen und Objekte.

- **Binding:** Ist die Assoziation eines Objekts mit einem Namen
- **Context:** Repräsentiert eine Menge von Name-Objekt Bindungen.
  Stellt eine Operation für die Auflösung des Namens zur Verfügung
  Kann Operationen zum Erzeugen und Aufheben von Bindungen, sowie eine Liste aller Bindungen zur Verfügung stellen.
- **Naming System:** Repräsentiert eine Menge verbundener Context Objekte
- **Naming Service:** Stellt Operationen auf einem Naming System zur Verfügung.

Verbindungsaufbau durch JNDI

1. Server erzeugt Interface Implementierung
2. Server registriert II beim Naming Service
3. Client fragt via JNDI beim Naming Service nach und erhält eine Referenz
4. Im Hintergrund wird der Methodenaufruf über das Netzwerk an ein serverseitiges Stellvertreterobjekt übertragen
5. Der Stellvertreter delegiert den Aufruf an eine vom Container bereitgestellte Instanz, die daraufhin eine Rückmeldung generiert.

8 Persistenz

Persistente Daten: Daten werden über die Lebensdauer der Sitzung eines Benutzers oder ein der Betriebssystemprozesse, die die Daten verarbeiten, hinaus gespeichert.

Verschiedene persistente Datenspeicher

- **Dateisystem:** Mittels einer I/O-API (BufferedReader, FileWriter) können Daten unter verschiedenen Betriebssystemen in Dateien geschrieben werden. (überall verfügbar, aber fehlende Erweiterbarkeit und problematische Updates)
- **XML-Dateien:** Durch XML können Daten plattformunabhängig, erweiterbar und menschenlesbar gespeichert werden. Nachteile: Kompliziert im Detail, beschränkt auf hierarchische Schachtelung
- **Relationale Datenbank:** (sollte bekannt aus Grundlagen: Datenbanken sein.)

Contentmanagementsystem: Ein CMS ist ein System, das es mehreren Benutzern/Programmen ermöglicht Informationen gemeinsam zu bearbeiten und zu speichern.

NoSQL Datenbanken
Steht für "Not only SQL" und sollte anstatt von relationalen Datenbanken eingesetzt werden wenn es um big data geht. Außerdem können sie besser mit vielen Schreib- und Leseanfragen umgehen, sind besser hinsichtlich redundanter, verteilter Datenhaltung, bieten dafür aber nur schwache Garantien hinsichtlich Konsistenz.

Eigenschaften/Definition:

- Das zugehörige Datenmodell ist nicht relational
- Das System ist von Anfang an eine verteilte und horizontale Skalierbarkeit ausgerichtet
- Das NoSQL-System ist Open-Source
- Das System ist schemafrei
- Das System bietet eine einfache API

8.1 Zugriff auf persistente Datenspeicher

In Java kann durch JDBC mit einer Datenbank kommuniziert werden.
Vorteile: günstig um: stored Procedures aufgerufen werden sollen, spezielle Abfragen ausgeführt werden müssen, auf proprietäre Datenbank-Funktionalität zugegriffen werden soll.

Nachteile: fehleranfällig während der Entwicklung, erfordert die Festlegung auf eine bestimmte Persistenzstrategie, Aufrufe dürfen nicht direkt in Geschäftslogik-Quellcode integriert werden.

Objektrelationales Mapping
OR-Mapping versucht den Zustand von Java-Objekten auf Daten in einer relationalen Datenbank abzubilden, um transparent persistente Datenhaltung bereitzustellen.

Single Table Strategy
Bei der Single Table Strategy werden alle Klassen der Subklassenhierarchie auf eine Tabelle und alle Attribute auf Spalten.

→ Einfache PK-Behandlung, suche nach Instanzen der Klassen ohne JOIN möglich, allerdings erhält man viele NULL-Werte.

In Java:

- @Entity über die Klasse
- @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
- @DiscriminatorColumn(name="Diskriminator-Spaltenname")
- @DiscriminatorValue("Diskriminator-Spalteninhalt")

Table per Class Strategy
Bei der Table per Class Strategy werden alle Klassen jeweils auf eine Tabelle und die Attribute auf Spalten der jeweiligen Relation abgebildet.

→ Such nach Instanzen ohne JOIN möglich, wenige NULL-Werte, allerdings komplexe PK-Behandlung

- @Entity über die Klasse
- @Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)

Joined Table Strategy
Bei der Joined Table Strategy werden alle Klassen jeweils auf eine Tabelle und nur die Attribute der Klasse (!Ohne Vererbung) auf Spalten abgebildet.

→ Keine NULL-Werte, einfache PK-Behandlung, allerdings suche nach Instanzen mit JOINS

- @Entity über die Klasse
- @Inheritance(strategy=InheritanceType.JOINED)

8.2 Persistent Entities

Persisten Entities bieten Objektorientierten Zugriff auf die persistenten Informationen in der Datenbank.
Jede Instanz einer Entity besitzt dabei einen eindeutigen Primary Key(analog zum Datenbankeintrag). Entities leben solange die zugehörige Datenbank existiert oder sie explizit gelöscht werden.

Spezifikationen:

- Eine Entity gehorcht der JavaBean Spezifikation (Alle Attribute haben getter und setter, sind also private, die Klasse hat einen Standardkonstruktor und muss java.io.Serializable implementieren)
- Eine Entity wird mit @Entity annotiert, der Primärschlüssel mit @Id und @GeneratedValue

Objektrelationale Abbildung bei Assoziationen

- **One-To-One:** @OneToOne("x") über den Getter der Attribute
- **One-To-Many:** @ManyToOne über 1, @OneToMany(mappedBy="Attributname des anderen Attributs") über *
- **ManyToMany:** @ManyToMany über die getter

Optional:

- Ladezeit: fetch=FetchType.EAGER/LAZY beeinflusst die Ladezeit des referenzierten Objekts.
- cascade=CascadeType.REMOVE (Löschweitergabe)

Entity Manager
Der javax.persistence.EntityManager bietet Methoden zum

- void persist(Object) speichert das Objekt in der Datenbank
- void remove(Object) löscht das Objekt aus der Datenbank
- Object find(Class, primaryKey) findet das Objekt mit dem zugehörigen PrimaryKey in der Datenbank
- void merge(Object) verändert das Objekt in der Datenbank.

Anfragen mit dem EntityManager
Anfragen mit dem EntityManager werden mit JPQL oder der Criteria API durchgeführt.

Sowohl bei der Criteria API als auch bei JPQL ist keine Typsicherheit garantiert. Zusätzlich kann der Compiler bei JPQL weder die Richtigkeit der Schlüsselwörter(select, where...) noch die korrekte Schreibweise von Namen überprüfen.

JPQL in Java:

```
public class QueryManagerBean implements QueryManager{

   @PersistenceContext(unitName="sebaEM")
   protected EntityManager em;

   public List<Cat> findCatsByLives(int livecount) {
      Query q = em.createQuery("select c from Cat c where c.lives = ?1");
      q.setParameter(1, livecount);
      return (List<Cat>) q.getResultList();
   }
}
```

8.3 Criteria API

Die Schritte der Criteria API:

```
public class QueryManagerBean implements QueryManager{

   @PersistenceContext(unitName = "sebaEM")
   private EntityManager em;
   public List<Customer> getPrivatKundenCA(){
      //Immer gleich
      CriteriaBuilder cb = em.getCriteriaBuilder();
      CriteriaQuery<Customer> cqry = cb.createQuery(Customer.class);

      //SELECT
      Root<Customer> root = cqry.from(Customer.class);
      cqry.select(root);

      //WHERE Klausel (mit Predicates)
      Predicate pGtAge = cb.gt(root.get("age"),10); //gt = greater than
      cqry.where(pGtAge);

      //Immer gleich
      Query qry = em.createQuery(cqry);
      List<Customer> results = qry.getResultList();

      }
}
```

1. CriteriaBuilder Objekt mithilfe von EntityManager em erstellen
2. CriteriaQuery Objekt mithilfe mithilfe Criterbuilder erstellen
3. SELECT-Klausel: Root Objekt mithilfe von CriteriaQuery Objekt erstellen.
4. WHERE-Klasel mithilfe von Predicate Objekt erstellen.
   - cb.gt = greater than
   - cb.equals = gleicher Wert
   - cb.and = WHERE a AND b
5. Query Objekt erstellen und Ergebnisliste ausgeben.
6. Typsicherheit auf Attributen allerdings nur mit externen MetaModell gewährleistet!

9 Betrieb und Wartung

Ziele: Problemsituationen durch geplantes Vorgehen vermeiden, optimale Unterstützung der Wertschöpfung durch zuverlässigen Betrieb möglich machen.

9.1 Einordnung in den Softwarelebenszyklus

Betrieb und Wartung startet nach Einführung der Software und macht oft mehr als 2/3 der Kosten des Softwareprojekts aus, denn Nutzen entsteht nur während "Betriebs- und Wartungsphase.
→ Wartungsaufwände meistens mehr als das Doppelte der Entwicklungskosten und verdrängen somit das Budget für Neuentwicklungen.

Nach Abschaffung liefert das System keinen Nutzen mehr allerdings können immernoch Kosten anfallen.

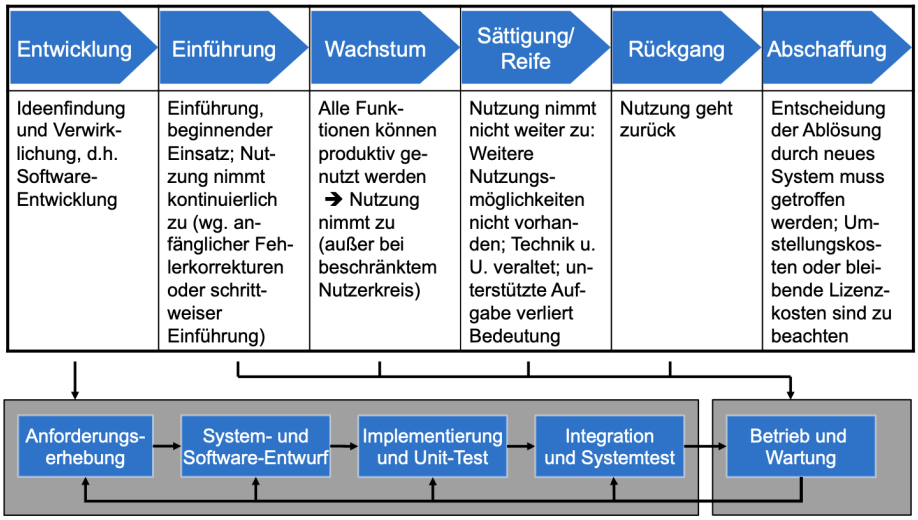

Figure 1: Softwarelebenszyklus.

9.2 Ziele, Herausforderungen, Aktivitäten

Kategorien der Wartung

- Reaktiv:
  - Verbesserung: Adaptive Wartung: Anpassung der Anwendung an eine neue Umgebung (z.B. OS)
  - Korrektur: Korrektive Wartung: Korrekturen zunächst unbekannter Fehler, die beim Einsatz der Anwendung auftreten

- Proaktiv:
  - Verbesserung: Perfektionierende Wartung: Verbesserung der Leistung im Sinne einer Performanceverbesserung, aber auch Restrukturierung der Anwendung mit dem Ziel den zukünftigen Wartungsaufwand zu minimieren.
  - Korrektur: Präventive Wartung: Behebung latenter Fehler, bevor diese effektiv aufgetreten sind.

Adaptive und perfektionierende Wartung
Da adaptive und perfektionierende Wartungen meist nicht kurzfristig sind oder zeitkritisch werden diese nach folgendem formalisiertem Verfahren durchgeführt:

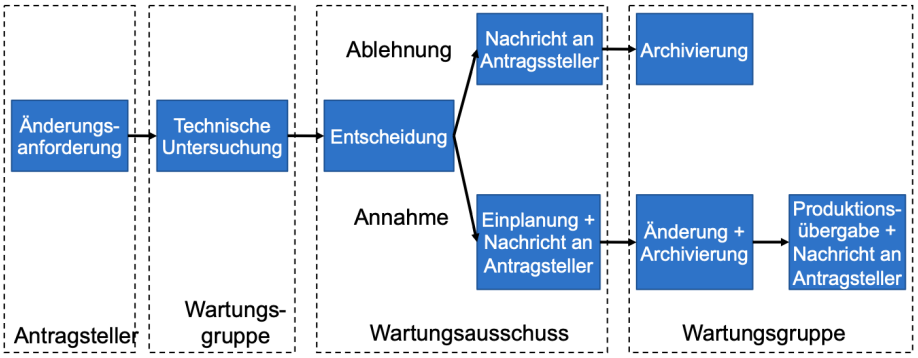

Figure 2: Adaptive und präventive Wartung.

Durchführung von Wartungsaktivitäten

Kombinierter Ansatz: Dieselben Mitarbeiter übernehmen Wartung und Entwicklung der Software.

Vorteile

- Programmierer die System entwickelt haben, kennen es gut und können Wartung besser und sorgfältiger durchführen.
- Nur wenige Programmierer sind bereit ausschließlich in der Wartung zu arbeiten.

Nachteile

- Perfektionismus könnte zur Gefahr werden.
- Wenn mehrere Projekte gleichzeitig laufen könnte Wartung zu kurz kommen.

Separater Ansatz: Personelle Trennung von Wartungs- und Entwicklungsarbeiten

Vorteile

- Entwickler werden nicht von Wartungsarbeiten gestört und können sich voll auf Entwicklung fokussieren.
- Dokumentation wird besser ausgeführt
- Dringliche Wartungsarbeiten werden schneller erledigt

Nachteile

- Hohe Abhängigkeit des Wartungsteams von Entwicklern bei schlechter Dokumentation.
- Lernaufwand für das Verständnis der Software ist Voraussetzung für Wartung.

9.3 Grundablauf des Wartungsprozesses

1. **Wartungsfall erfassen:** Wartungsfall erfassen mit Releasenummer, Umstand, Ansprechpartner, Reproduktionsanweisung.
2. **Erste Klassifzierung:** Sicherheitsrelevanz? Nachbesserungen?
3. **Analyse: Ursache finden**
4. **Vorläufige Klassifizierung**
5. **Analyse: Lösungsansätze finden:** Änderungen im Code? Änderungen in der Architekture?
6. **Auswirkungen auf Lieferung:** Klärung der Auswirkung auf die Lieferung
7. **Auswirkungen im Code:** Ist sichergestellt das nichts beschädigt wurde?
8. **Entscheidung Change Control Board:** In welchem Release wird die Veränderung eingebracht, von welchem Budget.
9. Implementieren, Release erstellen, Testen
10. **Analyse: Auswirkungen**
11. **Liefern:** Wenn die Änderung ausgeliefert wurde ist der Wartungsfall vollständig abgeschlossen.

Probleme bei Analyse und Umsetzung von Wartungsaufgaben

- Unverständliche Bezeichner, Unverständliche Kommentare
- Ungenutzter Code, Code-Cloning, "Programmiertricks", Sprach-Mix
- Unbekannte Abhängigkeiten

9.4 IT Infrastructure Library (ITIL)

ITIL ist eine Sammlung vordefinierter Prozesse, Funktionen und Rollen die die mit 37 Kernprozessen die Komponenten und Abläufe des Lebenszyklus von IT-Services.
→De-facto Standard für Service Management weltweit.

Bestandteile:

- ITIL-Core (Kernpublikationen): Bilden einen Satz von 5 Büchern:
  - Service Strategy
  - Service Design
  - Service Transition
  - Service Operation
  - Continual Service
- ITIL Complementary Guide (Ergänzungen)
- ITIL Web Support Services

Ablauf des Wartungsprozesses nach ITIL: Unterschiede sind bold.

1. **Wartungsfall erfassen:** Wartungsfall erfassen mit Releasenummer, Umstand, Ansprechpartner, Reproduktionsanweisung.
2. **Erste Klassifzierung: nach Auswirkung, Anzahl Betroffene, Abteilung, Ort, Dringlichkeit**
3. **Analyse: Ursache finden**
4. **Vorläufige Klassifizierung ITIL unterscheidet zwischen Incidents und Service Requests und definiert unterschiedliche Prozesse fuer beide Arten, Verantwortlichkeit bei Incident Manager**
5. **Analyse: Lösungsansätze finden:** Änderungen im Code? Änderungen in der Architekture?**Verantwortlichkeit bei Supporteinheit**
6. **Auswirkungen auf Lieferung:** Klärung der Auswirkung auf die Lieferung
7. **Auswirkungen im Code:** Ist sichergestellt das nichts beschädigt wurde?
8. **Entscheidung Change Control Board:** In welchem Release wird die Veränderung eingebracht, von welchem Budget. **Rolle Incident Manager**
9. Implementieren, Release erstellen, Testen **Rolle Supporteinheit**
10. **Analyse: AuswirkungenRolle 1st Level Support**
11. **Liefern:** Wenn die Änderung ausgeliefert wurde ist der Wartungsfall vollständig abgeschlossen.

9.4.1 Service Level Agreements

SLAs sind verbindliche Vereinbarungen, um Leistungen (=Services) zwischen dem Kunden und dem Servicebereich festzuschreiben. Diese ermöglichen eine sinnvolle Beurteilung der erbrachten Servicequalität und werden oft nur verwendet wenn externe Dienstleister beauftragt werden.

→ Legen Leistungsniveau fest, allerdings ist Übererfüllung des SLAs kostspielig und schadet dem Unternehmen.

Inhalt