Manufacturing Planning System HAW

Praktikum 2

Hashemilalam, Röhricht, Sahoo & Spijker

Gliederung

- Finale Architektur
  - Kontextsicht
  - Bausteinsicht
  - Laufzeitsicht
- Risikomanagement
- Implementation
  - Auftragserstellung
  - Fertigung
  - Test
- Offene Fragen & Annahmen

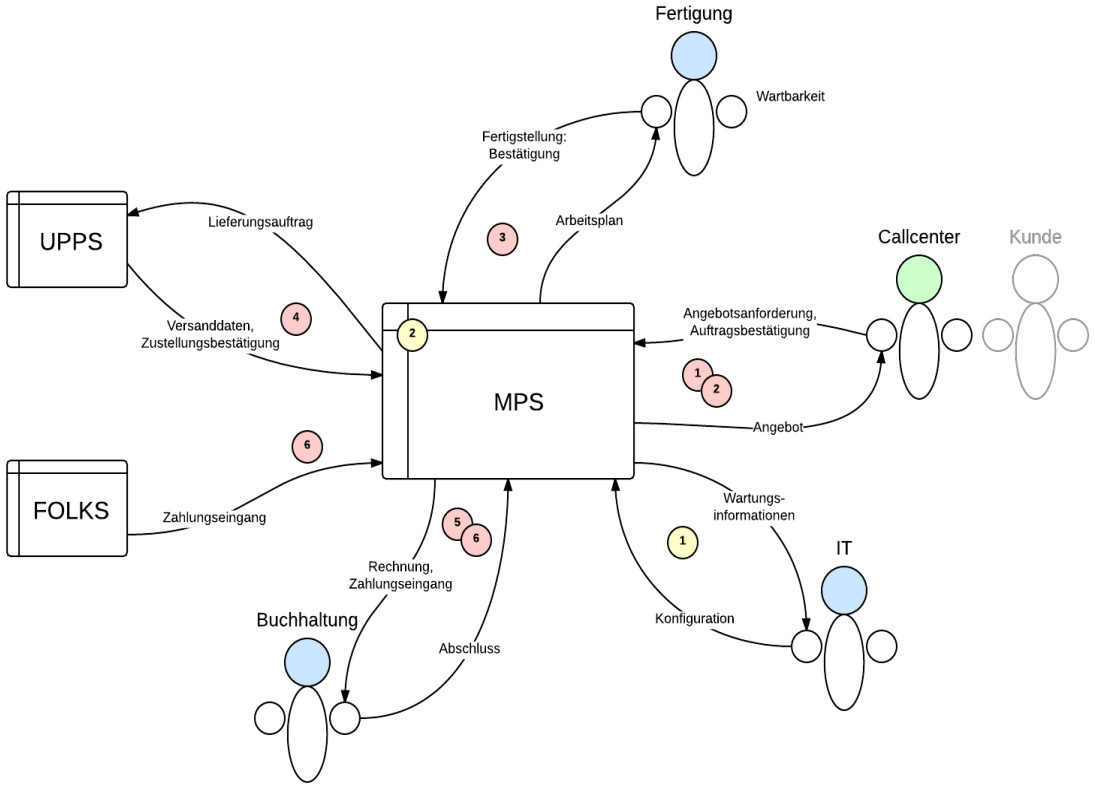
Finale Architektur

Funktionale Anforderungen

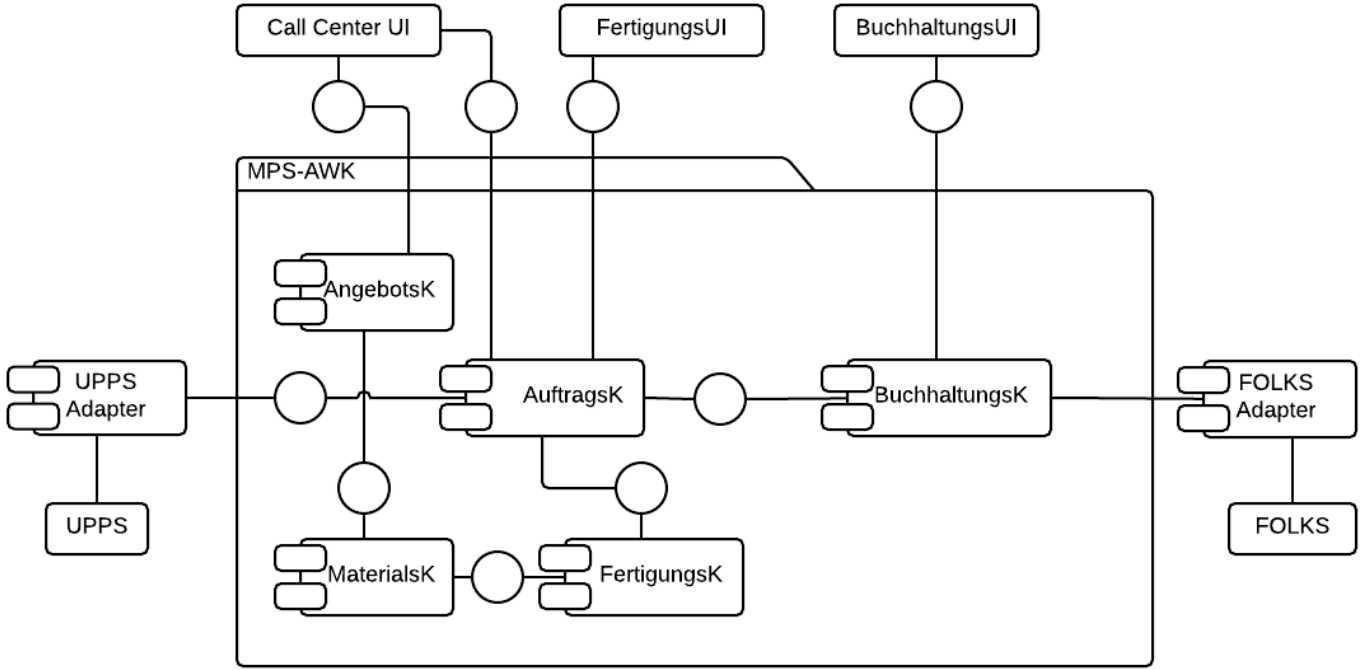
1. Verwaltung Angebote und Aufträge
2. Auftragserstellung
3. Fertigungsplanung
4. Transport und Lieferungsaufträge
5. Rechnungserstellung
6. Zahlungsverbuchung

Nichtfunktionale Anforderungen

1. Wartbarkeit
2. Robustheit



Finale Architektur



Finale Architektur

CCUI => Angebot
AngebotsNr berechneAngebot(KundenNr k, BauteilNr b, Integer number)

CCUI => Auftrag
AuftragsNr erstelleAuftrag(AngebotsNr n)

Auftrag => Fertigung
FertigungsauftragsNr fertigeAn(AuftragsNr n)

Auftrag => Material
Fertigungsplan zeigePlan(BauteilNr n)

Fertigung => FertigungsUI
void fertigeAn(FertigungsplanNr fpNr, FertigungsauftragsNr faNr)

Finale Architektur

FertigungsUI => Fertigung
void stelleFertig(FertigungsauftragsNr n)

Fertigung => Material
Integer zeigeInventar(BauteilNr n)
Boolean istKomplexesBauteil(BauteilNr n)
void erzeuge(BauteilNr n, Integer number)
void verbrauche(BauteilNr n, Integer number)

Fertigung => Auftrag
void stelleFertig(FertigungsauftragsNr n)

Auftrag => UPPS
SendungsNr liefere(Adresse sende, Adresse liefer)

Finale Architektur

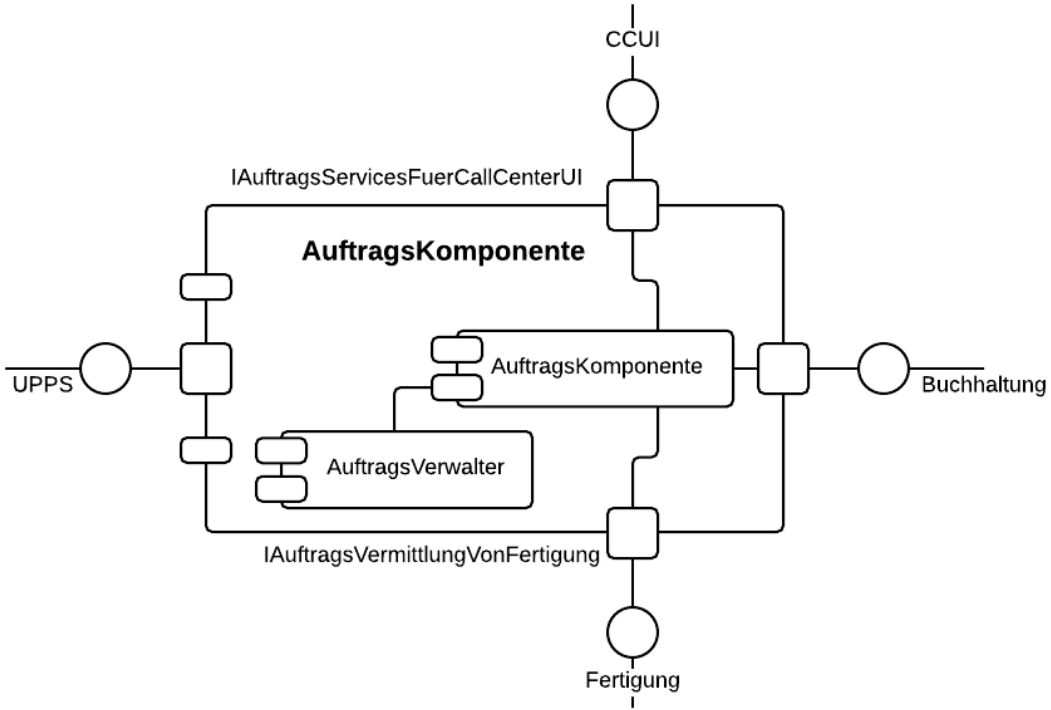
Auftrag => Buchhaltung
RechnungsNr erstelleRechnung(KundenNr k, AuftragsNr n)

Buchhaltung => BuchhaltungsUI
void sendeRechnung(RechnungsNr r)
void zahlungEingegangen(AuftragsNr n, Integer cent)

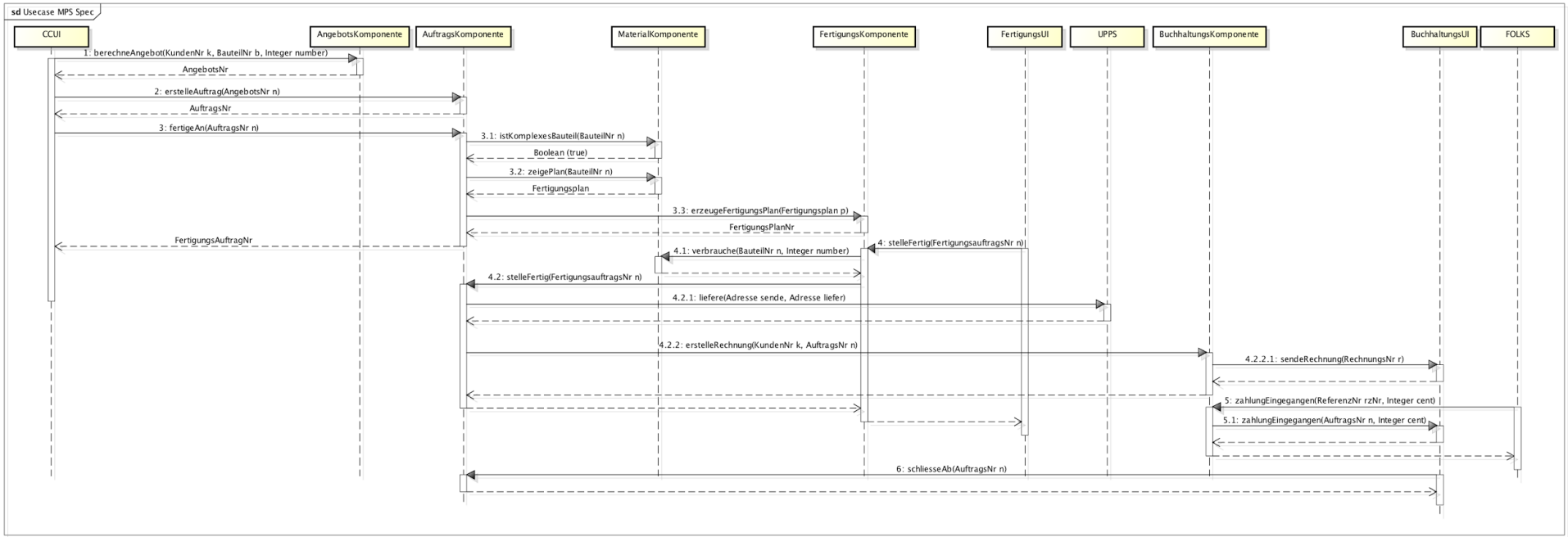
FOLKS => Buchhaltung
void zahlungEingegangen(ReferenzNr rzNr, Integer cent)

Buchhaltung => Auftrag
void schliesseAb(AuftragsNr n)

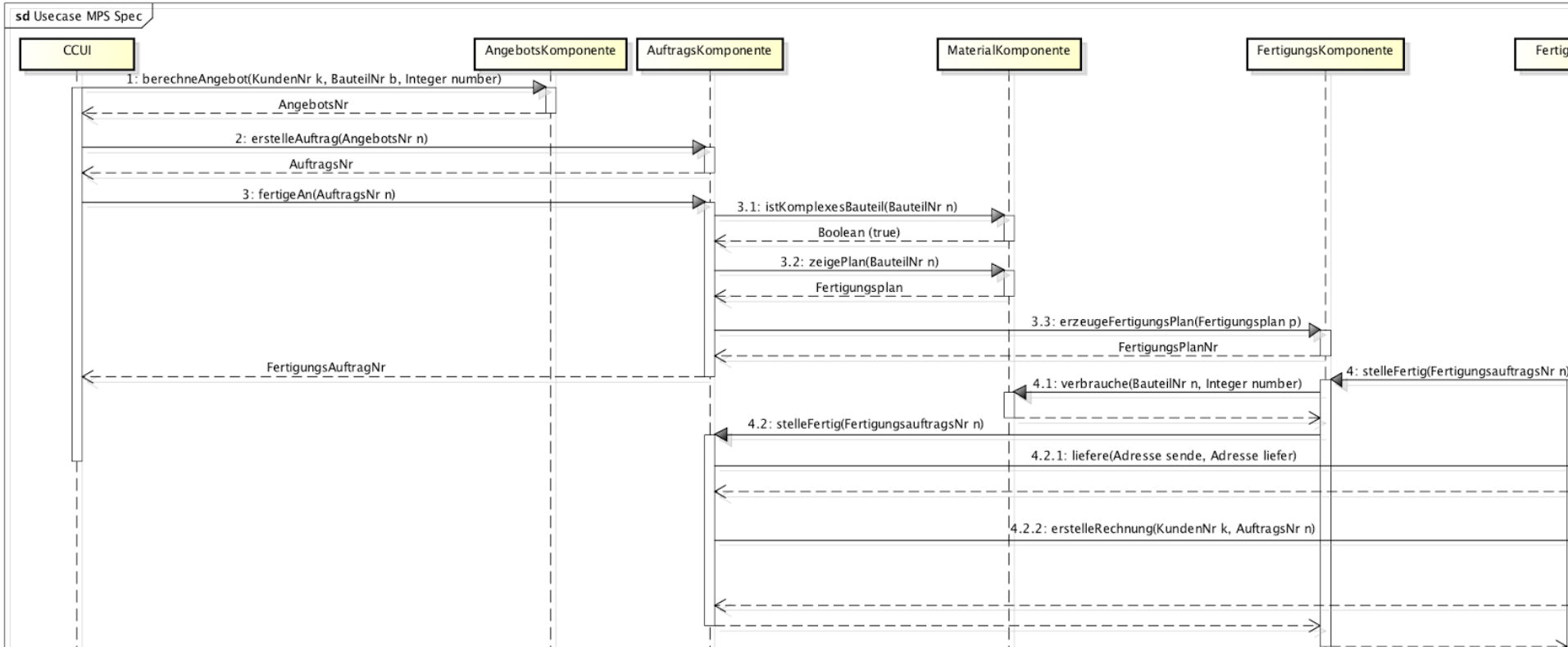
Finale Architektur



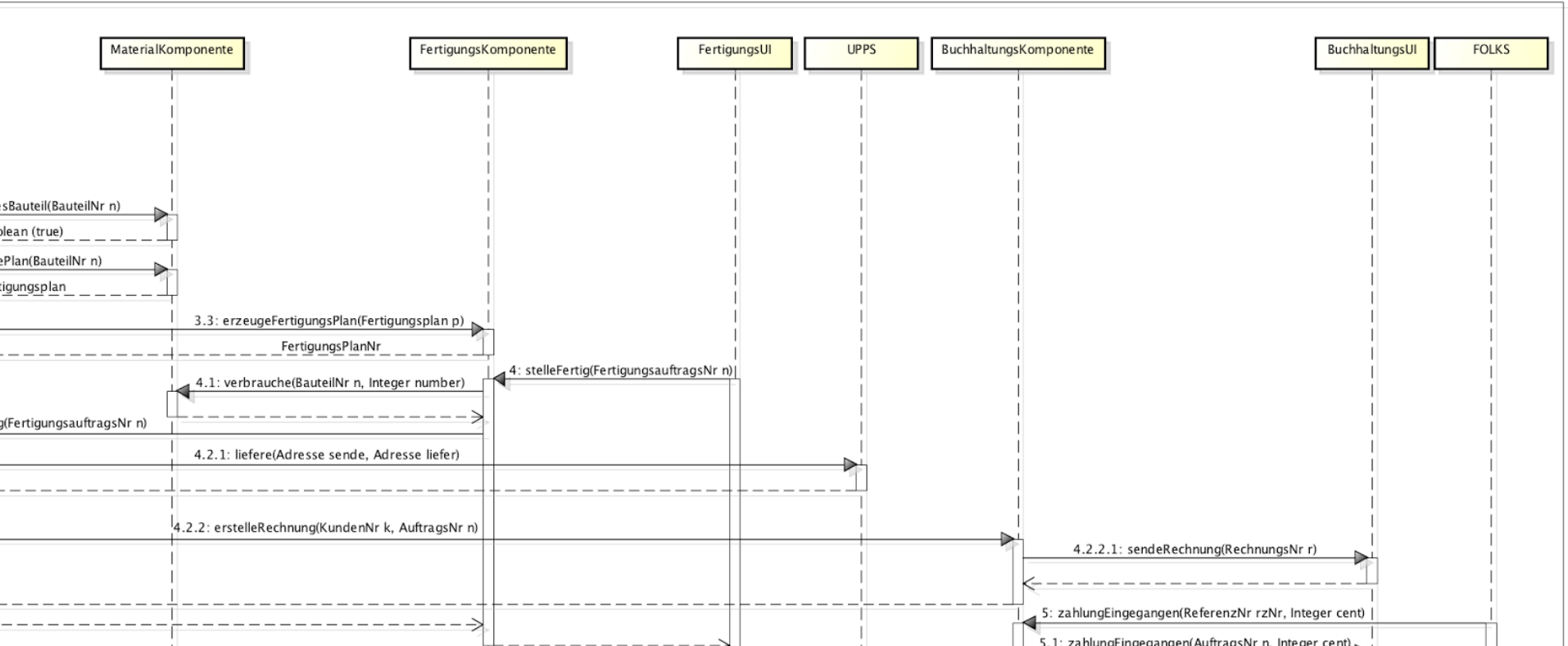
Finale Architektur



Finale Architektur



Finale Architektur



Risikomanagement

- personelle Ressourcen
- Zeitplanung
- Kosten und Leistung
- Anforderungs Spezifikation
- externe Faktoren
- Anwendungs bezogene Risiken
- technische Risiken
- geschäftliche Risiken
- Projekt Umfeld

Risikomanagement

personelle Ressourcen

- Stehen die besten und genügend Mitarbeiter zur Verfügung?
- Passen Personen in Schlüsselpositionen zusammen?
- Entsprechen sich Kompetenz und Aufgabe der Mitarbeiter?
- Stehen Mitarbeiter für die gesamte Projektdauer zur Verfügung?

- 1
- 1
- 1
- 1

Risikomanagement

Zeitplanung

- Das Projektteam ist bereits eingespielt
- Die nötige Ausstattung (Hard- und Software) ist vorhanden
- Ein genauer Zeitplan (Meilenstein etc.) ist festgelegt
- Die Anforderungen sind klar spezifiziert
- Änderungen der Anforderungen sind eher nicht zu erwarten

- 1
- 1
- 3
- 5
- 4

Risikomanagement

Kosten und Leistung

- Die Anforderungsspezifikation ist abgegrenzt und nicht zu komplex
- Die Hardware beschränkt die Funktionalität der Software nicht
- Es handelt sich um eine eigene Anwendung, ohne in andere Systeme eingebunden zu sein
- Die eingesetzten Techniken wurden bereits öfter verwendet

- 3
- 1
- 2
- 1

Risikomanagement

Anforderungs Spezifikation

- Wurde mit dem Auftraggeber bereits zusammengearbeitet?
- Hat der Auftraggeber eine genaue Vorstellung der Anforderungen und wurden diese schriftlich festgehalten?
- Versteht der Auftraggeber die technische Komplexität des Projekts?
- Kennt der Auftraggeber den Software-Engineering Prozess?
- Ist die geforderte Funktionalität der Software klar und verstanden?

- 1
- 3
- 3
- 2
- 3

Risikomanagement

externe Faktoren

- Müssen keine vom Auftraggeber gestellten Komponenten eingebunden werden?
- Entsprechen alle eingesetzten Werkzeuge der notwendigen Leistungsfähigkeit?
- Wurden extern bezogene Komponenten ausführlich getestet?
- Sind die Zusagen der externen Stellen verlässlich?
- Richten sich die externen Stellen genau nach den Vorgaben?

- 3
- 1
- 3
- 2
- 3

Risikomanagement

Anwendungs bezogene Risiken

- Sind Mitarbeiter vorhanden, die Erfahrung beim lösen von Performance-Problemen haben?
- Sind Messverfahren der Performance während der Implementierung vorgesehen?
- Beinhaltet die Anforderungsspezifikation Performance-Vorgaben?
- Gibt es Korrekturpläne für den Fall von Performance-Engpässen?
- Wurde nach möglichen Performance-Engpässen gesucht?

- 1
- 3
- 4
- 2
- 4

Risikomanagement

technische & geschäftliche Risiken

- Ist ein Software-Projektmanagement und Prozessmanagement Tool verfügbar?
- Sind passende Werkzeuge für Systemanalyse und Design verfügbar?
- Sind die Mitarbeiter für die eingesetzten Werkzeuge geschult?
- Sind Experten vor Ort, um Fragen zu den Tools zu beantworten?
- Die Vertragsgestaltung berücksichtigt ein Änderungsmanagement
- Der Umfang des Projektes ist nicht wesentlich größer als bei vergangenen Projekten
- Die Verträge sind rechtssicher
- Die Finanzierung notwendiger Investitionen ist sichergestellt

- 1
- 1
- 1
- 1
- 2
- 1
- 2

Risikomanagement

Projekt Umfeld

- Relevante Gesetze, Vorschriften und Normen sind beachtet
- Eine Änderung relevanter gesetzlicher Vorschriften o. ä. ist nicht zu erwarten
- Es gibt keine "wichtigeren" Projekte die zeitgleich bearbeitet werden müssen
- Notwendige Veränderungsprozesse im Unternehmen sind erkannt und werden von der Mehrheit der Stakeholder unterstützt

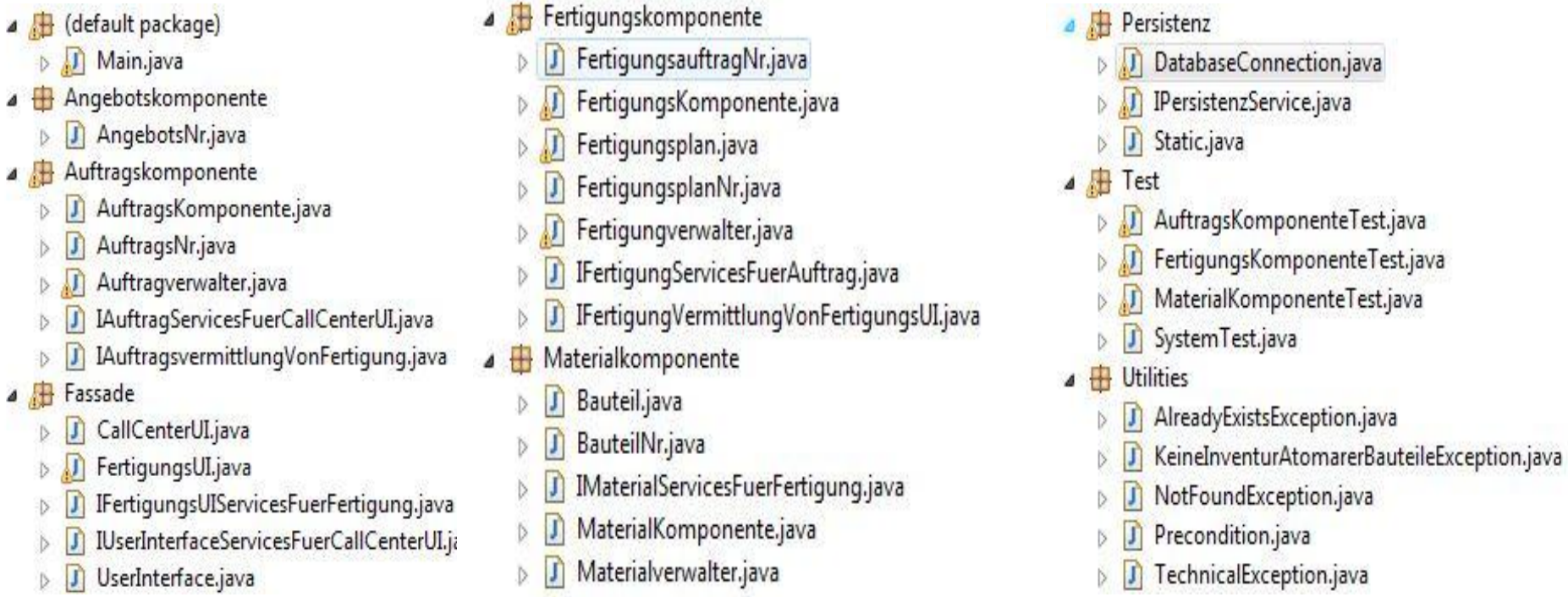
- 2
- 1
- 3
- 3

Implementation

Entscheidungen

- Das MPS wird in Java implementiert.
- Als Datenbank wird eine (quasi-)SQL Datenbank verwendet.
- Testing mit JUnit. Komponententests und Systemtest
- Je Komponente
  - Komponente.java implementiert Services sowie Anwendungsfälle
  - Verwalter.java für die Anbindung die Persistenz
  - Typ.java für die fachlichenDatentypen
- Interfaces verwenden fachliche Datentyp-nummern, niedrige Kopplung

Implementation



Implementation

Auftragserstellung

```java
public class AngebotsNr {
    Integer n;
    private AngebotsNr(Integer nr) {
        n = nr;
    }
    public static AngebotsNr angebotsNr(Integer nr) {
        return new AngebotsNr(nr);
    }
    public int getNR() {
        return n.intValue();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        AngebotsNr that = (AngebotsNr) o;

        if (n != null ? !n.equals(that.n) : that.n != null) return false;

        return true;
    }

    @Override
    public int hashCode() {
        return n != null ? n.hashCode() : 0;
    }
}
```

```java
public class AuftragsNr {
    Integer n;
    private AuftragsNr(Integer nr) {
        n = nr;
    }
    public static AuftragsNr auftragsNr(Integer nr) {
        return new AuftragsNr(nr);
    }
    public int getNR() {
        return n.intValue();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        AuftragsNr that = (AuftragsNr) o;

        if (n != null ? !n.equals(that.n) : that.n != null) return false;

        return true;
    }

    @Override
    public String toString() {
        return n.toString();
    }

    @Override
    public int hashCode() {
        return n != null ? n.hashCode() : 0;
    }
}
```

Implementation

Auftragserstellung

```
public class Auftragverwalter {

    private IPersistenzService pServ;

    public Auftragverwalter(IPersistenzService persServ) {
        this.pServ = persServ;
    }

    public AuftragsNr erzeugeAuftragAusAngebot(AngebotsNr n) throws NotFoundException, TechnicalException {
        String rep = pServ.read(ANGEBOT, n.getNR());
        pServ.delete(ANGEBOT, n.getNR());
        int afNr = pServ.create(AUFTRAG, rep);
        return AuftragsNr.auftragsNr(afNr);
    }

    public BauteilNr getAuftragBauteilNr(AuftragsNr n) throws NotFoundException, TechnicalException {
        String s = pServ.read(AUFTRAG, n.getNR());
        return bauteilNr(parseInt(s.split(",")[1]));
    }

    public Integer getAuftragBauteileAnzahl(AuftragsNr n) throws NotFoundException, TechnicalException {
        String s = pServ.read(AUFTRAG, n.getNR());
        return parseInt(s.split(",")[2]);
    }
}
```

Implementation

Auftragserstellung

```
public class AuftragsKomponente implements IAuftragServicesFuerCallCenterUI, IAuftragsvermittlungVonFertigung{

    private Auftragverwalter afVerw = null;
    private IFertigungServicesFuerAuftrag fServ = null;
    public List<FertigungsauftragNr> fertiggestellteFertigungsauftraege = new ArrayList<>();
    public AuftragsKomponente(IPersistenzService pServ, IFertigungServicesFuerAuftrag fServ) {
        afVerw = new Auftragverwalter(pServ);
        this.fServ = fServ;
    }
    // AnwendungsfÃ¤lle
    @Override
    public AuftragsNr erstelleAuftrag(AngebotsNr nr) throws NotFoundException, TechnicalException {
        return afVerw.erzeugeAuftragAusAngebot(nr);
    }

    public FertigungsauftragNr fertigeAn(AuftragsNr anr) throws TechnicalException, KeineInventurAtomarerBauteileException, NotFoundException {
        return fServ.fertigeAn(anr);
    }

    @Override
    public void stelleFertig(FertigungsauftragNr fnr) {
        System.out.println("Fertigungsauftrag " + fnr + " fertiggestellt.");
        fertiggestellteFertigungsauftraege.add(fnr);
    }

    @Override
    public BauteilNr getAuftragBauteilNr(AuftragsNr n) throws NotFoundException, TechnicalException {
        return afVerw.getAuftragBauteilNr(n);
    }
    @Override
    public Integer getAuftragBauteileAnzahl(AuftragsNr n) throws NotFoundException, TechnicalException {
        return afVerw.getAuftragBauteileAnzahl(n);
    }
}
```

```java
// AnwendungsfÃ¤lle
@Override
public FertigungsauftragNr fertigeAn(AuftragsNr n) throws NotFoundException, TechnicalException, KeineInventurAtomarerBauteileException {
    // FertugungsauftragNr erzeugen sowie anzahl und beuteilnr holen
    Integer anzahl = afVerm.getAuftragBauteileAnzahl(n);
    BauteilNr btn = afVerm.getAuftragBauteilNr(n);

    // atomare bauteile mÃ¼ssen nicht gebaut werden.
    if (!mServ.istKomplexesBauteil(btn)) {
        // atomare Bauteile sind nach Annahme immer verfÃ¼gbar. Es muss nichts gemacht werden.
        FertigungsauftragNr leererPlanAuftrag = fVerw.erzeugeLeerenplanAuftrag();
        afVerm.stelleFertig(leererPlanAuftrag);
        return leererPlanAuftrag;
    }


    // falls ich genÃ¼gend auf Lager habe, brauche ich nichts zu machen.
    int vorhanden = mServ.zeigeInventar(btn);
    if (vorhanden >= anzahl) {
        FertigungsauftragNr leererPlanAuftrag = fVerw.erzeugeLeerenplanAuftrag();
        afVerm.stelleFertig(leererPlanAuftrag);
        return leererPlanAuftrag;
    }

    // Sonst erzeuge ich `zuErzeugen`-viele Bauteile und verbrauche dessen Bestandteile.
    int zuErzeugen = Integer.valueOf(anzahl - vorhanden);
    Map<BauteilNr, Integer> plan = new HashMap();
    plan.put(btn, zuErzeugen);

    List<BauteilNr> verbrauch = mServ.bestandTeilevon(btn);
    fuegealsVerbrauchHinzu(verbrauch, plan);

    Fertigungsplan p = Fertigungsplan.fertigungsPlan(plan);
    FertigungsplanNr fpNr = fVerw.erzeugeFertigungsplan(p);
    FertigungsauftragNr fnr = fVerw.erzeugeFertigungsauftragAusAuftrag(n, fpNr);
    // Fertingungsplan an die Abteilung verschicken.
    fuiServ.fertigeAn(fpNr, fnr);
    return fnr;
}
```

Implementation

Fertigung

```
public class Fertigungverwalter {
    private IPersistenzService pServ;

    public Fertigungverwalter(IPersistenzService persServ) {
        this.pServ = persServ;
    }


    public FertigungsplanNr erzeugeFertigungsplan(Fertigungsplan p) throws TechnicalException {
        return fertigungsplanNr(pServ.create(FERTIGUNGSPLAN, p.toStringRep()));
    }

    public FertigungsauftragNr erzeugeFertigungsauftragAusAuftrag(AuftragsNr n, FertigungsplanNr fpNr) throws TechnicalException {
            return fertigungsauftragNr(pServ.create(FERTIGUNGSAUFTRAG, fpNr.toString()));
    }

    public FertigungsauftragNr erzeugeLeerenplanAuftrag() throws TechnicalException {
        String fpStr = fertigungsPlan(new HashMap<BauteilNr, Integer>()).toStringRep();
        FertigungsplanNr fpNr = fertigungsplanNr(pServ.create(FERTIGUNGSPLAN, fpStr));
        return fertigungsauftragNr(pServ.create(FERTIGUNGSAUFTRAG, fpNr.toString()));
    }

    public Fertigungsplan holeFertigungsplanzumFertigungsauftrag(FertigungsauftragNr fNr) throws NotFoundException, TechnicalException {
        String fPlanNrStr = pServ.read(FERTIGUNGSAUFTRAG, fNr.getNR());
        String fPlanStr = pServ.read(FERTIGUNGSPLAN, Integer.parseInt(fPlanNrStr));
        Fertigungsplan fPlan = Fertigungsplan.fromString(fPlanStr);
        return fPlan;
    }
}
```

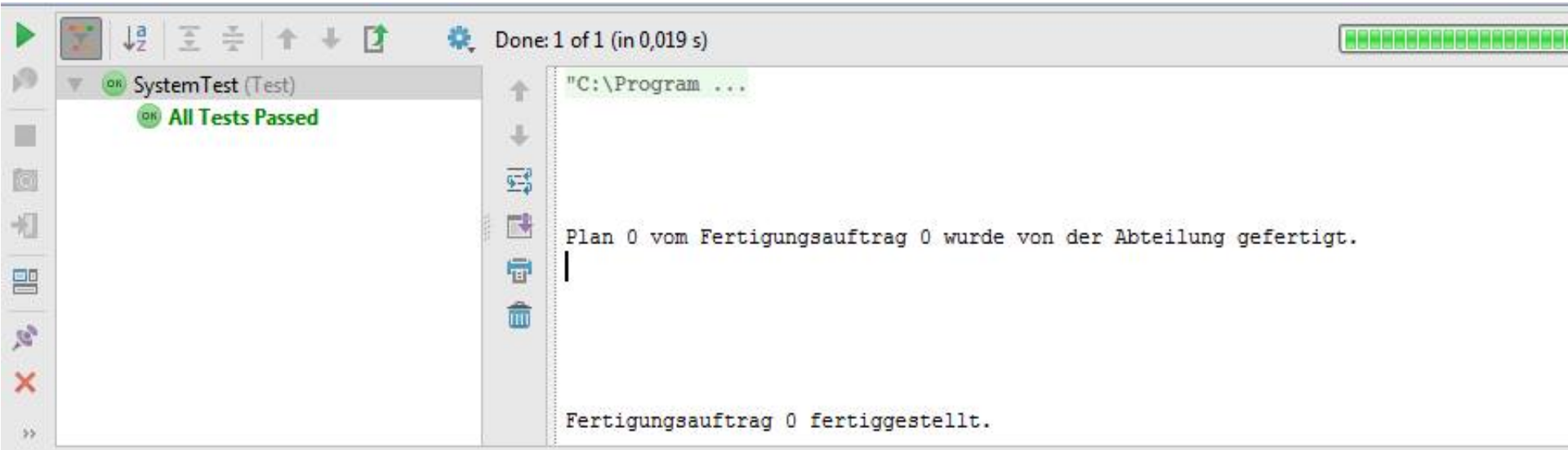
Implementation

Test

```
    @Test
    public void testAuftragserstellungUndFertigung() throws Exception {
        // Auftrag aus einem Angebot erstellen
        aKomp.erstelleAuftrag(anNr);

        String s = pServ.read(AUFTRAG, 0);
        assertEquals(s, "15,2,7,14.2");

        // Auftrag erteilen
        fServ.fertigeAn(AuftragsNr.auftragsNr(0));

        FertigungsplanNr fpNr = fUI.erhalteneAuftraege.get(0);
        String fPlan_str = pServ.read(FERTIGUNGSPLAN, fpNr.getNR());
        assertEquals(fPlan_str, "0,-2,1,-1,2,2");

        fVermFui.stelleFertig(FertigungsauftragNr.fertigungsauftragNr(0));

        int anz = mServ.zeigeInventar(BauteilNr.bauteilNr(bKomplex));
        assertEquals(anz, 7);


        FertigungsauftragNr erwartet = FertigungsauftragNr.fertigungsauftragNr(0);
        FertigungsauftragNr erhalten = aKomp.fertiggestellteFertigungsauftraege.get(0);
        assertEquals(erwartet, erhalten);
    }
}
```

Implementation

Test



Offene Fragen & Annahmen

?

Wie soll das System gewartet werden?

Welche Robustheit ist gefordert?

Welche Daten sind für die Berechnung eines Angebots relevant?

!

Kundeninteraktion findet nur über das Callcenter statt

Die Fertigung produziert nur für Aufträge, i. e. nicht auf Vorrat

Nichtkomplexe Bauteile sind stets vorrätig

Die Datenbank lässt sich einfach auf String basierten CRUD Interface anbinden