Klausurrelevante Themen:

Softwareentwicklung & Softwarequalitäten:

- Software Engineering
  bezeichnet das ingenieurmäßige Vorgehen bei der Softwareentwicklung mit dem Ziel qualitativ hochwertige Software in kurzer Zeit zu entwickeln:
  – Erstellung eines organisatorischen Plans
  – Inhaltlicher Plan (Konzeption):
    Δ Pflichtenheft
    Δ Architekturbeschreibung
    Δ Entwurfsdokument
- Softwarequalitäten:
  – intern:
    für den Entwickler wichige Eigenschaften der Software
    Δ Wartbarkeit
    Δ Wiederverwendbarkeit
    Δ Portierbarkeit
    Δ Verständlichkeit
    Δ Interoperabilität
  – extern:
    für den Nutzer sichtbar/relevant
    Δ Korrektheit
    zu verstehen bezüglich der Spezifikation (d.h. bei korrekten Eingaben)!
    Δ Zuverlässigkeit
    lineare Größe; Wahrscheinlichkeit, dass die erwartete Funktionalität eingehalten wird
    Bei korrekter Spezifikation ein schwächere Form der Korrektheit
    Maße: rate of failure occurrence, mean time to failure, availability
    Δ Robustheit
    „vernünftiges" Verhalten unter nicht spezifizierten Umständen
    Ideal (vollständige Spezifikation): Robustheit = Korrektheit
    Δ Effizienz
    Δ Benutzerfreundlichkeit

Pflichtenheft:

- Anforderungen
  – funktional
  – nicht funktional
- Abschnitte
  1. Zielbestimmung
     Hauptaufgabe des Systems, Gründe für Systementwicklung, Zielgruppe (Vorwissen und Erfahrungen)
  2. Produkteinsatz (Einsatzbereich der Software)
     2.1. Beschreibung des Problembereichs als zusammenhängender Text – ggf. mit illustrierenden Grafiken
     2.2. Glossar
     2.3. Modell des Problembereichs
     Beschreibung durch UML **Klassendiagramm**
     2.4. Beschreibung des Geschäftsfeldes
     Überblick über die Abläufe im Einsatzbereich des Systems und die daran Beteiligten geben durch UML **Use Case Diagramm**, Rollen werden durch Actors dargestellt
     2.5. Beschreibung der Geschäftsprozesse
     Nähere Erläuterung der im Punkt zuvor identifizierten Geschäftsprozesse durch Tabelle und UML **Aktivitätendiagramm**
  3. Produktfunktionen
     3.1. Beschreibung der funktionale Anforderungen durch **Use Case-Diagramme** mit Angabe der Systemgrenze
     Use Cases unterstützen elementare Geschäftsprozesse
     3.2. Genauere Beschreibung jedes Use Cases im Einzelnen aus Nutzersicht
     3.2.1. Charakterisierende Informationen
     3.2.2. Szenario für den Standardablauf
     3.2.3. Szenarien für alternative Abläufe
     3.2.4. Beschreibung des allg. Ablaufs durch **Aktivitätendiagramm**
     3.2.5. Offene Fragen
  4. Produktcharakteristiken
     nicht-funktionale Anforderungen (z.B. Qualitäten, die das Produkt von vergleichbaren Systemen unterscheidet)
     Typen: USE, EFFIZIENZ, PFLEGE, SICHER, LEGAL, Zuordnung zu Use Cases in Tabellenform
- Qualitätskriterien
  Übertragung der Qualitätsmerkmale von Software auf das Pflichtenheft: Das Pflichtenheft hat nur interne Qualitäten, externe lassen sich als interne auffassen bzw. ergeben keine neue Qualität für das Pflichtenheft.
  – Korrektheit
    z.B. keine Vorwegnahme von Entwurfsentscheidungen
  – Änderbarkeit
  – Verständlichkeit

  zus, nach IEEE 830:
  – Eindeutigkeit
    Syntaktisch und semantisch korrekte Verwendung der UML
  – Vollständigkeit
    Ergibt die Summe der Ziele der Use Cases und der Produktcharakteristika die in der Zielbestimmung angegebenen Ziele?
    Sind Fachbegriffe im Glossar erklärt?
  – Konsistenz
  – Überprüfbarkeit
  – Nachvollziehbarkeit

Architekturbeschreibung:

- Übergang Pflichtenheft → Architekturbeschreibung
  PH 2.5 → AB 1.1; PH 3.2 → AB 1.2
- Abschnitte
  1. Analyse der Produktfunktionen (Übergang Anforderungen → Entwurf dokumentieren)
     1.1 Grobanalyse
     Segmentierung der **Aktivitätendiagramme in Swimlanes**: Zuordnung elementarer Produktfunktionen auf Pakete
     1.2 Feinanalyse
     Verfeinerung der Szenarien von Produktfunktionen durch **Sequenzdiagramme**
     a) nur Interaktion Actor ↔ System
     b) auch Interaktion zwischen System-internen Objekten
     Einhaltung der klassischen 3-Schichten-Architektur (GUI, Anwendung, Datenhaltung) wenig sinnvoll, da in Anwendungsteil technische und inhaltliche Aufgaben vermischt werden!
     Statt dessen: Unterscheidung von drei Sorten von Klassen:

a)  Übergangsklassen (boundary classes)
b)  Gegenstandklassen (entity classes)
c)  Steuerungklassen (control classes)

2. Softwarearchitektur (Ergebnis dokumentieren)
   2.1 Paketstruktur (ergibt sich aus 1.1)
   Pakete mit Aufgabenbeschreibung und zugeordneten Use Cases in Tabellenform
   Abhängigkeiten im **Paketdiagramm** (ergibt sich durch Nicht-lokale Transitionen in der Grobanalyse)
   2.2 Klassenstruktur (ergibt sich aus 1.2)
   Zuordnung von Klassen zu Paketen in **Analyse-Klassendiagramm** mit Operationen und Attributen
   Klassentabelle

- Qualitätsmerkmale
  Korrektheit, Vollständigkeit, Konsistenz, Nachvollziehbarkeit
  vertikale Konsitenz: zwischen Pflichtenheft und Architekturbeschreibung
  horizontale Konsistenz: innerhalb der Architekturbeschreibung

Entwurfsdokument:

- Übergang Architekturbeschreibung → Entwurfsdokument
  AB 2.1 → ED 1.1; AB 2.2 (+ED 1.1) → ED 1.1 und ED 2.x; AB 1.2 → ED 1.2;
- Abschnitte
  1. Grobentwurf
     1.1 Komponentenstruktur
     Eine Komponente hat vertraglich spezifizierte Schnittstellen und ausschließlich explizite Kontextabhängigkeiten. Sie kann unabhängig verwendet und leicht mit anderen Komponenten integriert werden.
     Komponenten z.B. als Pakete mit Schnittstellen
     Komponenten, Schnittstellen und Abhängigkeiten in **Komponentendiagramm**
     Schnittstellen und Parameterklassen in **Klassendiagramm mit Interfaces**
     1.2 Komponenteninteraktion
     Sequenzdiagramm auf Komponentenebene
     1.3 Protokolle für Komponentenbenutzung (abzuleiten aus 1.2)
     Zustandsdiagramm legt Folge erlaubter Operationsaufrufe der realisierten Interfaces fest
  2. Feinentwurf
     2.x Implementierung zu Komponente x (ergibt sich u.a. aus 1.1)
     Ggf. weitere Klassen hinzufügen, die zur Realisierung benötigt werden. Endgültige Festlegung auch unter Berücksichtigung der Besonderheiten der zu verwendenden Programmiersprache. Aufnahme von zu importierenden Bibliotheksklassen. Darstellung durch **Entwurfs-Klassendiagramme mit Sichtbarkeiten** (public, protected, private)
  3. Verteilungsentwurf
     3.1 Grobstruktur der Verteilung (abzuleiten aus 1.1 Komponentendiagramme)
     Beschreibung durch **Verteilungsdiagramme**, auf welchen Netzwerkknoten welche Komponenten installiert sind
     3.2 Technische Realisierung
     Klassendiagramme
- Qualitätsmerkmale
  Beschreibung: Analog zu Pflichtenheft
  Realisierung: Softwarequalitäten

Organisatorische Planung:

- Vorgehensmodelle:
  – Wasserfallmodell
  wenig flexibel, keine Dokumentation der Änderungen
  wenig Interaktion mit Auftraggeber und Nutzer
  – Rational Unified Process
  Δ Iterativer Prozess: Das Gesamtprojekt zerfällt in eine Reihe von Iterationen (Miniprojekten) mit unterschiedlichen Schwerpunkten, die mit einem Meilenstein abgeschlossen werden
  Δ Trennung von Phasen und Tätigkeiten: Jede Phase umfasst mehr oder weniger alle Tätigkeiten der Softwareentwicklung
  Δ Umfasst Vorgehensmodell, Methodologie und Templates
  Δ Phasen:
  1. Anfangsphase
  2. Ausarbeitungsphase
  3. Konstruktionsphase
  4. Übergangsphase
  Δ Meilensteine:
  1. Life-Cycle Objective (Vision)
  2. Life-Cycle Architecture (Grundlegende Architektur)
  3. Initial Operational Capability (Full Beta-Release)
  4. Produktrelease (Final Release)

Diagramme:

- Zurordnung UML-Diagramme Abschnitte aus Pflichtenheft/Architekturbeschreibung:

siehe UML-Notationselemente für TSE1 von Reiko

Qualitätssicherungsdokument:

- zu Qualitätskriterien:
  – Aufzählung der gefundenen Mängel
  – Begründung oder vorgenommene Änderung