Prüfer: Martin Jung
Beisitzer: Keine Ahnung
Dauer: 30 Minuten
Note: 1,3

Definiere Software Architecture.
Es gibt diverse Definitionen, die alle in die gleiche Richtung gehen.

Riehle: Die Architektur eines Software-Systems ist die Aggregation von allen strukturellen und dynamischen Eigenschaften der Elemente, die von Relevanz für ihre Stakeholder sind.

~Zsfsg:
1. Übersichtsplan des betrachteten Systems, der zeigt:
    1.1 Welche logischen Bausteine es gibt, und welche Dienste diese anbieten
    1.2 Welche physischen Bausteine es gibt
    1.3 Wie diese miteinander interagieren
    1.4 Wie aus den Bausteinen das System aufgebaut ist
    1.5 Wie das fertige System in einer Ausführungsumgebung eingesetzt wird
2. Basis übergreifender Entwurfsentscheidungen mit systemweiten Konsequenzen
3. vergröberte Darstellung der Lösung für ein Projekt
4. Dokumentation, wie Anforderungen auf die Lösung abgebildet ist

Um die Flut an Informationen zu filtern gibt es Sichten. Diese legen weniger wert auf manche und mehr Wert auf andere, um bestimmten Stakeholdern als gute Übersicht in ihren Zuständigkeitsbereichen zu dienen.

Was sind Sichten, warum gibt es sie?
Ich habe Kruchtens 4+1 Sichten genannt:
logische Sicht
- zeigt Funktionen des Systems unabhängig von HW und SW
- Kruchten nennt Klassendiagramm

Prozesssicht
- zeigt dynamische Systemaspekte
- Sequenz- / Ablaufdiagramm

Entwicklungssicht
- zeigt relevante Infos für SW-Entwickler
- Komponentendiagramm (man sieht gut, welche Komponenten Teil wovon sind und deren Verbindungen)

Physische Sicht
- zeigt wie Artefakte auf HW abgebildet werden
- Deployment-Diagramm

Szenariosicht
- Zeigt, wie alle Use-Cases mit gebautem System gelöst werden
- Use-Case Diagramm

Male mir ein einfaches Beispiel zu jeder Sicht.
Hier hat er nacheinander gesagt, welche Sicht ich als nächstes malen soll.
Währenddessen hat er die Begriffe der UML-Sichten au der VL genannt, also
- logische Sicht
- dynamische Sicht
- Verteilungs-Sicht
- Realisierungs-Sicht
(- Szenariosicht hat er ausgelassen)

M2M, M2T – warum braucht man das, was kann das?
Hier habe ich ein bisschen verkackt.

Model to Model scheint wirklich über stark verschiedene Modelle Transformieren zu können

Model to Text ist das wichtigere in der Realität, damit kann man z.B. aus Petri-Netzen auch .jars erzeugen und eben mit Generierung viel Code bauen.

Was sind Styles und Muster? Was für Beispiele gibt es?
Style:
Wiederverwendbare Pakete von Design-Entscheidungen und Constraints, welche auf eine Architektur angewandt werden, um gewählte und gewünschte Qualitäten einzuführen.

(meine) Beispiele: Pipes and Filters, Blackboard, Repository (auch wenn letzteres nicht so krass ist)

Muster (Pattern):
Wiederverwendbare Lösungsvorschriften, die in unterschiedlichen technischen Anwendungsgebieten eingesetzt werden können um wiederkehrende Aufgaben zu lösen.

(meine) Beispiele: Mediator, Broker

Erkläre einen Style genauer (kein Pattern, wahrscheinlich weil Zeit)
meine Wahl: Pipes and Filters

Daten verarbeiten und weiterleiten.
Filter: verarbeitet Daten (rechnen etc.)
Pipes: Leitet Daten weiter
Beispiel: Unix Shell (|)

Vor-/Nachteile:
+ Lose Kopplung
+ Wiederverwendung
+ Evolution (Erweiterbarkeit)
+ Durchsatz/Deadlockanalyse
- Benutzerinteraktion – Datenkonversion (Inputformat ← Outputformat?)
- Fehlerbehandlung schwierig
- Steuerkomponente erforderlich

Ist das eine gute Architektur?
legt Bild von (aktuell noch) V02, F22 vor. Das Bild zeigt Rover-Komponenten, die recht wild miteinander verbunden sind. Teile sind z.B. Expertenschnittstelle, Fernbedienung, Bewegungssteuerung, Routenplanung oder Aktorik

Gesagt habe ich meine subjektive Meinung: Alles ist sehr wirr und es gibt viele Verbindungen. Es sieht also nach einer eher Schlechten Architektur aus.

Was er wollte war vor allem Kopplung – die ist nämlich schlecht. Eben wegen den vielen Verbindungen zwischen den Komponenten

Kann man auf dem Bild angewandte Styles oder Pattern erkennen?
Mir sind keine aufgefallen.
Er: "Wenn überhaupt, dann Layer, da die Komponenten vertikal angeordnet sind. Aber das ist auch eher nur Erfahrungswert"

→ Ich glaube nicht dass diese Frage meine Note beeinflusst hat

Andere quantitative Metriken neben Kopplung?
(Ich habe nicht mehr als 2 oder 3 Metriken – nicht die Punkte hier – jeweils genannt)

Codemetriken
1. Größe (LOC, Story/Function Points)
2. Komplexität (McCabe, Halstead)
3. Aufwand (COCOMO)
4. Objektorientierte Metriken
   (DIT, NOC, CBC, LCOM)

Architekturmetriken
1. Komponenten pro User Story
2. Schachtelungstiefe der Komponenten
3. Nutzer pro angebotener Schnittstelle
4. Szenarien pro User Story
5. Artefakte pro Komponente
6. Zustandsmodell der Komponente
7. Betroffene Artefakte/Knoten pro User Story

McCabe: #Pfade, die man durch Kontrollflussgraphen eines Moduls legen kann
COCOMO: Formeln basierend auf Auswertung vergangener Projekte (regression formulas)
DIT: Depth of Inheritance Tree
NOC: Number Of direct Children (Vererbung)

Stichwort zu qualitativen Metriken?
ATAM
(mehr wollte er nicht, wahrscheinlich auch wegen Zeit. Und die Schritte von ATAM will er nicht fragen, da das ein Fall von "Ich weiß, wo ich nachschauen muss" ist.)

Was steht auf erster Ebene des Qualitäsbaums und was für Beispiele gibt es? (Wahrscheinlich wegen Zeitgründen keine tieferen Fragen)
Quality attributes:
- Performance
- Modifiability
- Availability

Wie kommt man zu relevanten Metriken?
Goal Question Metric

Wie funktioniert GQM?
1. Man überlegt sich, wozu man Metriken messen will
2. Man überlegt sich Fragen dazu (die man auch an der Architektur messen können sollte)
3. Man findet passende Metriken

Denke dir Fragen aus, die gut für eine Architektur wäre, die häufig kleine Updates ausliefern wird.
(hab erst ein bisschen rumgelabert und Fragen genannt, die man nicht gut an der Architektur abmessen kann – das wollte er nicht)
Wie viele Komponenten werden in einem Artefakt realisiert?
Wie viele User-Stories pro Komponente?

Alles in allem eine angenehme Prüfungsatmosphäre, man konnte auch mit ein bisschen witzeln den Prüfer zum lachen bringen 😉