Besprechung Übung 2

Requirements Engineering I
HS 2009

Reinhard Stoiber

Allgemein

- Die Übung wurde insgesamt wenig gut gelöst
  - Kreative / innovative Ideen und Lösungen waren gefragt
- Im Schnitt wurden nur 56,8 % der Punkte erreicht
- Punkte
  - Alle Gruppen erreichten die nötigen 60 Punkte (gesamt)
  - Ist keine Garantie für erfolgreiche Prüfung
    - Prüfung ist Einzelarbeit

2.1 Informationsquellen, Neue Anforderungen (1)

a) Direkte und indirekte Beteiligte (stakeholders) beim NYPD

- Definition “Stakeholder”
  - *“A stakeholder is a person or organization who influences a system’s requirements or who is impacted by that system.”*

    Reference: M. Glinz, R. Wieringa (2007). Stakeholders in Requirements Engineering. Special Issue, *IEEE Software, Vol 24 No 2*

- Sind i.d.R. nicht nur die Endbenützer, sondern auch
  - Sponsor(en) / Kunde(n), Architekt(en), Entwickler, Tester, Qualitätsingenieur(e), Projektmanager, Produktmanager, Operateur(e), Instandhalter

2.1 Informationsquellen, Neue Anforderungen (2)

University of Zurich

a)

- Beim NYPD - laut Angabe nur
  - Streifenpolizisten (indirekt)
    - Ausser Sie planten dass diese *direkt* auf das NY-FEDS zugreifen?
  - Dispatchers des NYPD (Benützer, direkt)
  - NYPD bzw. Clancy Wiggum (Sponsor / Kunde)

- Weiter sinnvoll ... (Annahmen treffen)
  - Projektmanager (Clancy Wiggum, Sie oder jemand anderes)
  - Architekt, Entwickler
  - Tester
  - Operateur (vermutlich ebenso die Dispatchers)
  - Instandhalter

2.2
Klassendiagramm (1)

Gefragt

- Detaillierte Analyse des *Problembereichs*
  - Klassen, Beziehungen, Attribute, Operationen
- Modellieren der Soll-Situation
- Kein Design einer möglichen Implementierung

Kriterien

- Vollständig, richtig, (ev. innovativ)
- Sind alle Daten enthalten die das System benötigt?

2.2
Klassendiagramm (2)

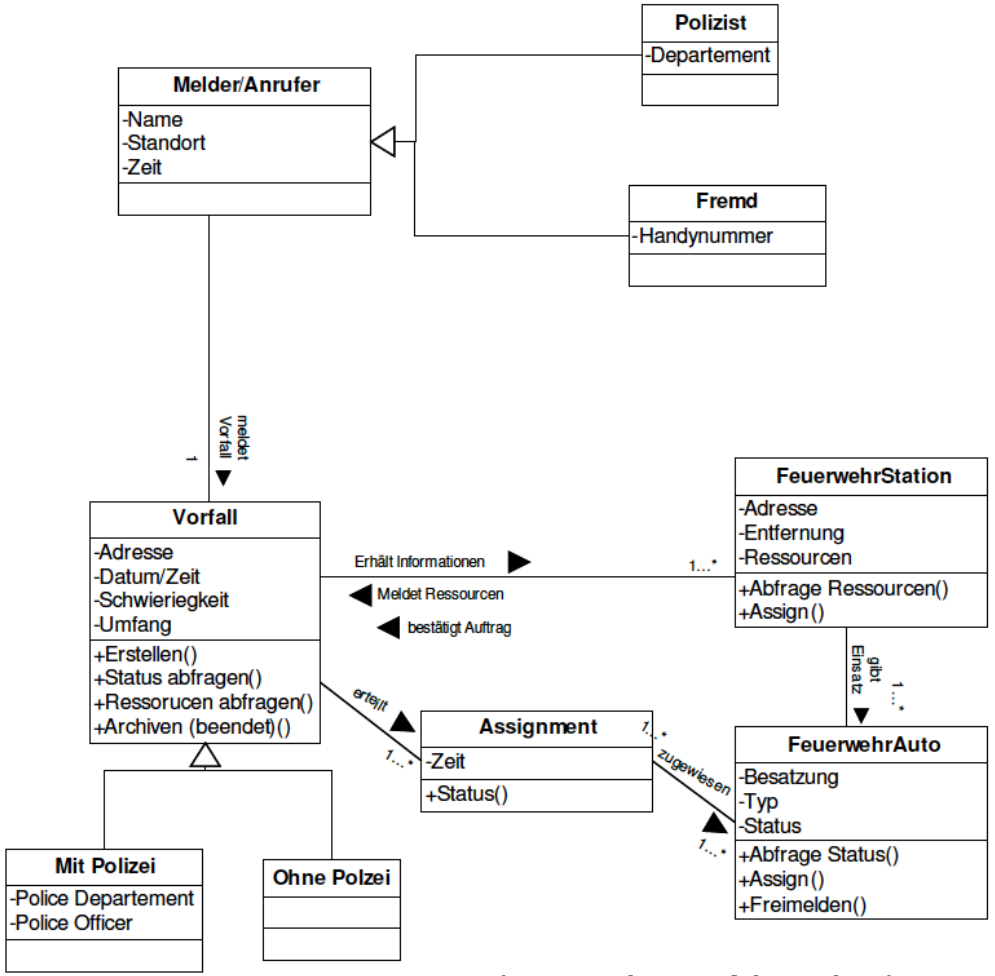
University of Zurich

Klassendiagramm

- teilweise fehlt noch Multiplizität
- Karte nicht explizit
- Dispatcher nicht modelliert

Fokus

Anwendungsdomäne,
bzw. Problembereich

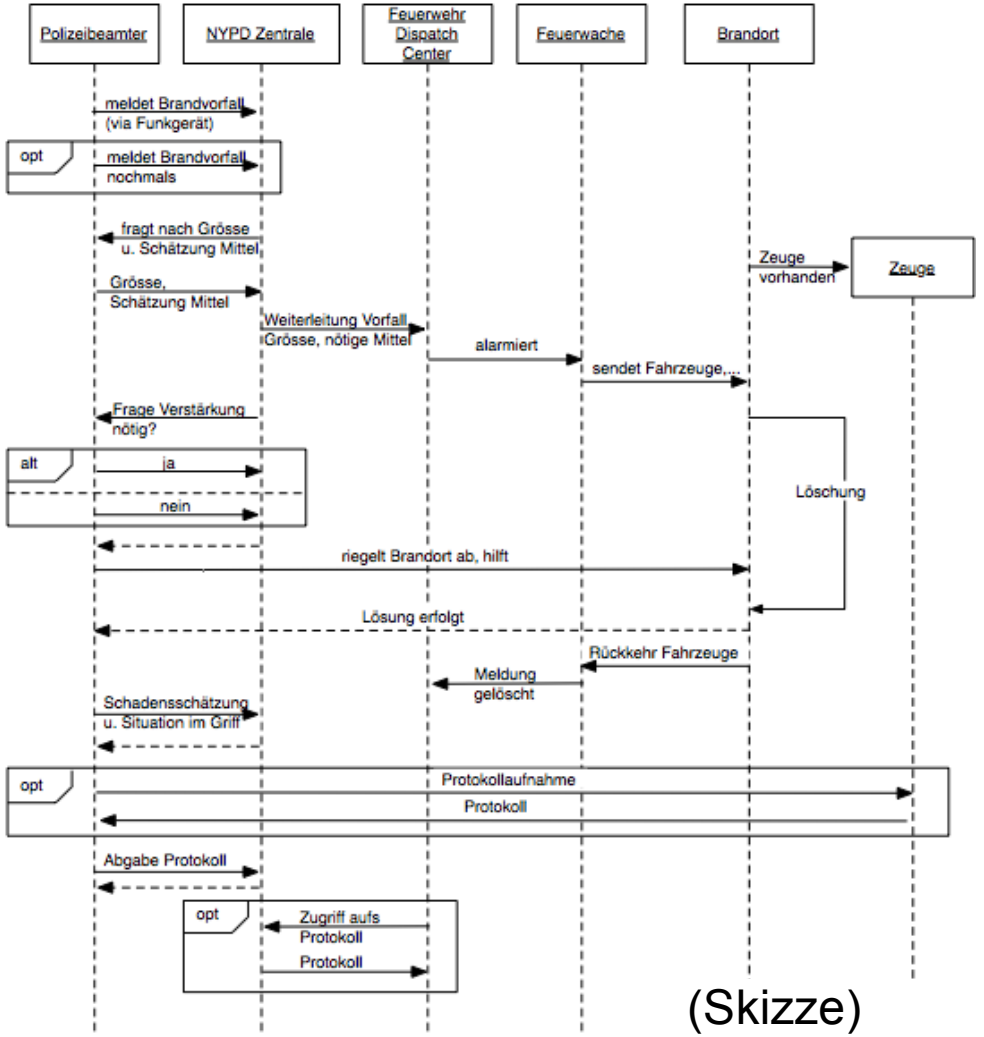


(aus einer Abgabe)

2.3 Szenarienanalyse (1)

a.) Interaktionsdiagramm

Mögliches Interaktionsdiagramm in UML 2 (mit Frames)

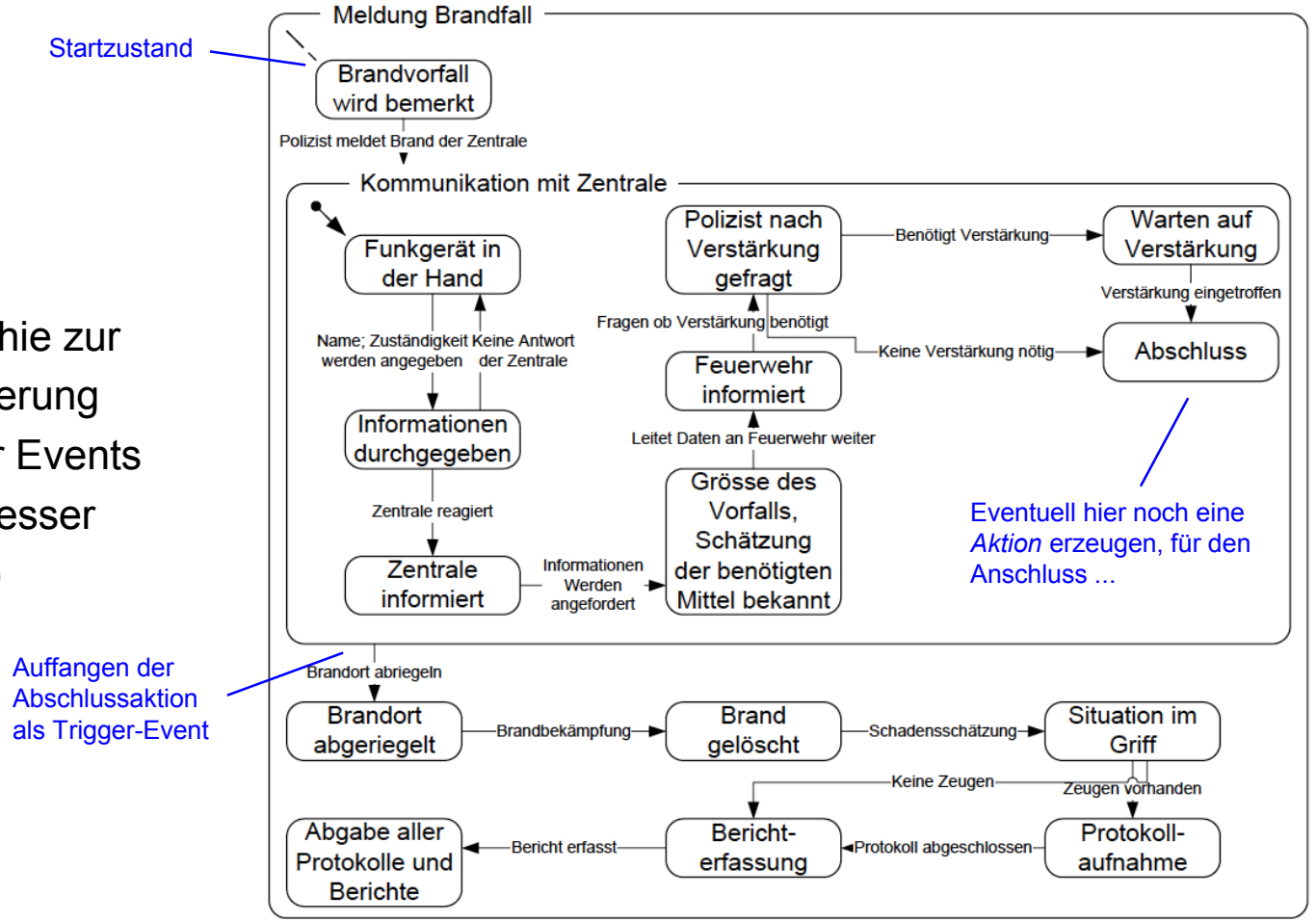


(Skizze)

2.3 Szenarienanalyse (2)

b.) Statechart

Ein Statechart
- ermöglicht Hierarchie zur besseren Strukturierung
- verwendet hier nur Events für Transitionen (besser auch mit Aktionen)



(aus einer Abgabe)

2.3
Szenarienanalyse (3)

c.) weitere gängige Darstellungsformen

- Unstrukturierter Text
- Strukturierter Text
- Aktivitätsdiagramm
- Use-Case-Diagramm

Vorteile, Nachteile wie im Skript

Kriterium

- Interpretation der Anwendbarkeit / Nützlichkeit am gegebenen Szenario

2.3
Szenarienanalyse (4)

d.) Unstimmigkeiten zwischen den Sichten

➔ Klassendiagramm vs. Szenarienmodellierung

- Klassendiagramm modelliert Daten, Zusammenhänge, Operationen
- Szenarienmodell modelliert
  - Interaktion (Aktionen und Reaktionen) zwischen Benutzer und System
  - Interaktionen zwischen den Systemteilen

- Systementwicklung erfordert Integration beider Sichten
- Konsistenz ist aber mit separaten Diagrammen nicht immer gegeben

2.4
Formale Spezifikation (1)

- Lösungen waren recht mangelhaft
- Syntax Probleme (Notation)
  - Fast bei allen Gruppen
- Kriterien
  - Spezifikation basierend auf axiomatischer Mengenlehre und Prädikatenlogik (vgl. Links Z Glossary bzw. Reference Manual)
  - Kein “Programmieren” der Funktionen wie in Java
    - Sondern definieren von Restriktionen, Invarianten, Beziehungen und Zustandveränderungen
  - Korrekte Syntax
  - Spezifikation der geforderten Funktionalität

2.4
Formale Spezifikation (2)

Funktion “assignFireEngine”

Definition Sets der verfügbaren Fahrzeuge in verschiedenen Entfernungskategorien

Zuweisung eines nahest-möglichen Fahrzeugs wie in der Entscheidungstabelle

Entscheidungstabelle

Engines near avail. Engines intermediate avail. Engines far avail.	J - -	N J -	N N J	N N N
Sent near sent intermediate sent far non available	X	X	X	X

Definition einer Zuweisung, falls ein Fahrzeug vorhanden ist

bleiben unverändert

assignFireEngine

Δ FireEngineAdministration
currentDate : Date
currentTime : Time
assignedFireEngine : FireEngine
enginesNear : ℙ FireEngine
enginesIntermediate : ℙ FireEngine
enginesFar : ℙ FireEngine
e? : EmergencyLocation
assignedDistance! : { sent_near | sent_intermediate | sent_far | none_available }

(∃ fh : FireHouses • (∃ f : FireEngine • (f, fh) ∈ basedAt ∧ (f, e?) ∉ isOnAssignment ∧ (fh, e?, DISTANCE=near) ∈ DISTANCES)) ⇒ enginesNear = ∅ ∪ ∀ f : FireEngine
(∃ fh: FireHouses • (∃ f : FireEngine • (f, fh) ∈ basedAt ∧ (f, e?) ∉ isOnAssignment ∧ (fh, e?, DISTANCE=intermediate) ∈ Distances)) ⇒ enginesIntermediate = ∅ ∪ ∀ f : FireEngine
(∃ fh: FireHouses • (∃ f : FireEngine • (f, fh) ∈ basedAt ∧ (f, e?) ∉ isOnAssignment ∧ (fh, e?, DISTANCE=far) ∈ Distances)) ⇒ enginesFar = ∅ ∪ ∀ f : FireEngine

((enginesNear ≠ ∅) ⇒ (assignedDistance! = sent_near ∧ ∃ f : FireEngine | f ∈ enginesNear • assignedFireEngine = f)) ∨
((enginesNear = ∅ ∧ enginesIntermediate ≠ ∅) ∧ (assignedDistance! = sent_intermediate ∧ ∃ f : FireEngine | f ∈ enginesIntermediate • assignedFireEngine = f)) ∨
((enginesNear = ∅ ∧ enginesIntermediate = ∅ ∧ enginesFar ≠ ∅) ∧ (assignedDistance! = sent_far ∧ ∃ f : FireEngine | f ∈ enginesFar • assignedFireEngine = f)) ∨
((enginesNear = ∅ ∧ enginesIntermediate = ∅ ∧ enginesFar = ∅) ∧ (assignedDistance! = none_available))

(assignedDistance! = none_available ∧ Assignments’ = Assignments ∧ isOnAssignment’ = isOnAssignment) ∨
(assignedDistance! ≠ none_available ∧ Assignments’ = Assignments ∪ {(assignedFireEngine, currentDate, currentTime, e?)} ∧ isOnAssignment’ = isOnAssignment ∪ {(assignedFireEngine, e?)})

Returns’ = Returns
FireHouses’ = FireHouses
FireEnginesOutOfOrder’ = FireEnginesOutOfOrder
Distances’ = Distances
basedAt’ = basedAt

(Skizze)

2.4
Formale Spezifikation (3)

University of Zurich

Funktion “returnFireEngine”

returnFireEngine

Δ FireEngineAdministration
currentDate : Date
currentTime : Time
assignedLocation : EmergencyLocation
f? : FireEngine

∃ e : EmergencyLocation | (f?, e) ∈ isOnAssignment • assignedLocation = e

Returns’ = Returns ∪ {(f?, currentDate, currentTime, assignedLocation)}
Assignments’ = Assignments \ {(f?, currentDate, currentTime, assignedLocation)}

isOnAssignment’ = isOnAssignment \ {(f?, assignedLocation)}

FireHouses’ = FireHouses
FireEnginesOutOfOrder’ = FireEnginesOutOfOrder
Distances’ = Distances
basedAt’ = basedAt

Definieren des zugewiesenen Orts für dieses Fahrzeug

Definieren des zusätzlichen Returns und des abzüglichen Assignments

Das Fahrzeug ist nicht mehr auf einem Assignment

bleiben unverändert

(Skizze)

2.4
Formale Spezifikation (4)

University of Zurich

Funktion “calculateOperatingTime”

calculateOperatingTime

Ξ FireEngineAdministration
currentDate : Date
startTime, endTime : Time
f? : FireEngine
d? : Date
e? : Location
assignmentDuration! : Time

---

d? ≤ currentDate

∃ assign : Assignments | EmergencyDate = d? ∧ AssignedFireEngine = f?
AssignmentLocation = e? • startTime = AssignmentTime(assign)
∃ ret : Returns | ReturnDate = d? ∧ AssignedFireEngine = f?
AssignmentLocation = e? • endTime = ReturnTime(assign)

assignmentDuration! = endTime - startTime

Das Datum muss in der Vergangenheit liegen

Definition der startZeit und endZeit für den Einsatz des gegebenen Fahrzeugs, Datums und Ort

Ausgabewert = Dauer

(Skizze)

Danke für die Aufmerksamkeit.