24 Spezielle Fragen des Architekturentwurfs

- Querschnittsaufgaben
- Zu dokumentieren als aspektorientierte Teilkonzepte
- Zum Beispiel:
  - Benutzerschnittstelle
  - Datenhaltung
  - Fehlertoleranz
  - Sicherheit

24.1 Benutzerschnittstelle

Grundsätze

- Boehm's Prinzip: “Do unto others as you would have others to unto you – *if you were like them*”
- Nievergelt/Venturas Regel: Jederzeit wissen
  - Wo bin ich
  - Was kann ich hier tun
  - Wie bin ich hergekommen und wohin kann ich gehen
- Verlässlichkeit und Durchschaubarkeit: System verhält sich wie ein verlässlicher menschlicher Partner – Reaktionen sind nachvollziehbar, in analogen Situationen analog, keine bizzarren, unvorhersehbaren Reaktionen
- Metaphernkonformität: Schnittstelle folgt konsequent einer Präsentations-/Interaktionsmetapher (zum Beispiel Schreibtischmetapher, Instrumenten-brettmetapher, ...)
- Rückkopplung: Jede Bedienungshandlung hat eine unmittelbare Wirkung

Abgrenzung zur Spezifikation

Zwitterrolle:

- Benutzerschnittstelle ist mitentscheidendes Kriterium für die Brauchbarkeit eines Systems
  → gehört zu den Anforderungen
- Bedienung ist nur Mittel zum Zweck: Funktionen verwalten/auslösen
  → gehört in den Entwurf
- ⇨ Fallweise zu klären, wann was gemacht wird

- Immer mit Benutzerbeteiligung
- Im Zweifelsfall: Prototypen

Was ist zu gestalten?

- Medien für Ein- und Ausgabe
- Ausgaben: Anzeigen, Ausdrucke, Ton...
- Dialogabläufe
  - geführt – frei
  - modal – nichtmodal
- Eingaben: Masken, Zeigeinstrumente, Sprache,...
  - standardisiert – anwendungsorientiert

Einbettung in die Gesamtarchitektur

- Getrennt vom übrigen System (MVC-Muster)
- Heute in der Regel ereignisgesteuert
- Häufig unter Verwendung eines gegebenen Rahmens

24.2 Datenhaltung

- ☆ Teilweise Bestandteil der Gesamtarchitektur
- ☆ Teilweise als aspektorientiertes Teilkonzept: Schemata, Integrität, Zugriffsschutz, etc.

Zu treffendende Architekturentscheidungen:

- ❍ Flüchtig – persistent
- ❍ Dateien – Datenbanken
- ❍ Homogen – heterogen
- ❍ Physisch zentral – verteilt
- ❍ Logisch zentral – föderiert – autonom
- ❍ Steuerung/Koordination (z.B. durch TP-Monitor)
- ➪ Festlegung der Grundarchitektur

Aspektorientiertes Teilkonzept für Datenhaltung

- Festlegung von Datenmodell(en)
- Entwurf der konzeptuellen und externen Schemata
- Verteilung
  - was – wo
    - Daten
    - Anwendungslogik
  - Replikation
- Sicherstellung der Integrität
- Zugriffsschutz

24.3 Fehlertoleranz

Zur Terminologie

- Eine Person begeht einen *Irrtum (mistake)*.
- Als mögliche Folge davon enthält die Software einen *Defekt (defect, fault)*.
- Wird der Defekt durch Inspizieren der Software gefunden, so ergibt das einen *Befund (finding)*.
- Bei der Ausführung von Software mit einem Defekt kommt es zu einem *Fehler (error)*: Die tatsächlichen Ergebnisse weichen von den erwarteten / den richtigen ab.
- Dies kann zum *Ausfall (failure)* eines softwarebasierten Systems führen.
- *Fehlertoleranz* – Auftreten von Fehlern führt nicht zum Ausfall.

Hardware-Fehlertoleranz

- System überlebt Ausfälle von Hardware-Komponenten
- Maßnahmen zu treffen
  - primär gegen Material- und Energieprobleme
  - sekundär gegen Entwurfsfehler
- Mittel:
  - Vervielfachung von Komponenten, zum Beispiel
    - Dreifach-Redundanz
    - Datenspiegelung
  - Transaktionen mit Log/Wiederaufsetzen
- Software muss teilweise angepasst werden

Software-Fehlertoleranz

Maßnahmen gegen logische Fehler

- ☆ in den Daten
- ☆ in Programmen

Mittel zur Tolerierung von Datenfehlern

- ❍ Datenredundanz
- ❍ ggf. Wiederholen / Ignorieren / Interpolieren / Verwendung voreingestellter Werte

Mittel zur Tolerierung von Programmfehlern

- Wiederholen mit anderen Algorithmen (Recovery Blocks)
- N-Versionsprogrammierung: Mehrere, unabhängige Realisierungen der gleichen Software

  Probleme:

  - Datensynchronisation in den n Versionen
  - präzise gemeinsame Spezifikation notwendig
  - Defekte in mehreren Versionen sind nicht stochastisch unabhängig

  Daher:

  - Die Ausfallwahrscheinlichkeit des Gesamtsystems ist nicht gleich dem Produkt der Ausfallwahrscheinlichkeiten der Einzelversionen (sondern größer!)
  - N-Versionsprogrammierung hilft nicht gegen Spezifikationsfehler.

24.4 Sicherheit

Zwei Bedeutungen:

- Schutz von Information gegen missbräuchliche Verwendung (Security)
- Schutz von Personen und Sachen gegen Schäden, die durch den Einsatz von Informatiksystemen verursacht oder mitverursacht werden (Safety)

Sicherheit und Zuverlässigkeit sind nicht das Gleiche:

Zuverlässigkeit (reliability) – Die Fähigkeit eines Systems, die verlangte Funktionalität unter gegebenen Randbedingungen für eine gegebene Zeit zu erfüllen (d.h. in dieser Zeit ohne Ausfälle zu funktionieren).

24.4.1 Sicherheit im Sinn von Security

Bedrohungen

Verlust / Bruch / Verweigerung von

- Vertraulichkeit (Abhören, Spionieren)
- Integrität (Datenmanipulation, Datenverlust)
- Authentizität (Erschleichen von Zugriff, Erschleichen von Leistungen, Verfälschen der Urheberschaft)
- Verfügbarkeit (Nichtverfügbarkeit oder Verweigerung von Leistungen)
- Verbindlichkeit (Bestreiten der Urheberschaft oder des Empfangs)

Mögliche Maßnahmen

- Verschlüsselung
- Identifikation
- Transaktionen
- Logging
- Auditing
- Zertifikate
- Physische Maßnahmen (Einschließen, Zutrittskontrolle)

24.4.2 Sicherheit im Sinn von Safety

Problemerkennung und -analyse

- Minimierung des Risikos, dass ein System in einen verbotenen Zustand gerät (zum Beispiel, dass eine Lokomotive ein rotes Signal überfährt, ohne dass die Steuersoftware der Lok dies erkennt und eine Schnellbremsung auslöst)
- Identifikation gefährlicher verbotener Zustände
- Gefahrenanalyse: Beurteilung der Gefährlichkeit der verbotenen Zustände, d.h. wie hoch sind Eintretenswahrscheinlichkeit und Schadenhöhe eines resultierenden Schadens (zum Beispiel eines Zugunglücks beim Überfahren eines roten Signals)

Maßnahmen

- Beweis der Unerreichbarkeit gefährlicher Zustände
- Nachweis der Ungefährlichkeit erreichbarer verbotener Zustände
- Analyse möglicher Wege zu den kritischen verbotenen Zuständen
  a) Vorwärts: mögliche Folgen von Ereignissen konstruieren
  b) Rückwärts: Ausgehend vom Gefahrenzustand Ermittlung direkter Ursachen (und-oder-Kombinationen)
     - Rekursive Bestimmung der Ursachen für die Ursachen
     - Resultierenden Fehlerbaum soweit wie möglich zurückverfolgen
- Entwurf so ändern, dass alle möglichen Pfade zu einem Gefahrenzustand unterbrochen oder mit hinreichend kleinen Wahrscheinlichkeiten belegt sind