Software-Engineering II

Basiert auf Skript "Software-Engineering II" von Prof. Dr. Ludger Bölke.

1 Objektorientierte Analyse (OOA)

1.1 Anforderungsanalyse

(siehe Skript "Software-Engineering I")

1.2 Analysemodell

Umfasst ein erweitertes Klassendiagramm/Analyse-Modell-Diagramm (mit Steuerungs- und Schnittstellen-Klassen) und die Modellierung des Dynamischen Ablaufs.

- verwendet die Sprache des Entwicklers
- interne Darstellung des Systems
- redundanzfrei

Zu beschreiben sind System- und Anwenderschnittstellen sowie die Geschäftslogik.

1.2.1 Stereotypen

Klassifizierungsmöglichkeit für bestimme UML-Elemente zur Darstellung ihrer Bedeutung im Gesamtsystem.

- «entity»: Datenklasse: stellt fachlichen Sachverhalt dar
  - viele Attribute und primitive Operationen
  - kaum Zustände und komplexe Operationen
- «control»: Steuerungsklasse: Ablaufsteuerung und/oder Berechnungen
  - wenige oder keine Attribute
  - komplexe Operationen
  - u.U. komplexe Zustandsmodelle notwendig
- «interface»: abstrakte Definition rein funktionaler Schnittstellen
  - keine Attribute oder Operationen
  - wird von anderen Klassen implementiert
  - Hilfsmittel zur Arbeitsteilung
  - Platzhalter zur Kommunikation mit Systemen, noch nicht existieren
- «boundary»: Schnittstellenklasse: Benutzerschnittstelle zur Interaktion mit Akteuren
  - keine Logik, keine Datenhaltung
  - nicht persistent (nicht dauerhaft instanziert)
  - delegieren lediglich Operationen

1.2.2 Übergang zum Analysemodell

- Zwischen Akteuren und Anwendungsfall agiert ein Schnittellenobjekt
- Steuerung von Anwendungsfällen durch ein Control-Objekt

1.2.3 Dynamisches Modell

Beschreibung des Ablaufs des Systems. Dieser Bereich kann als Grauzone gesehen werden, da man viele Elemente ebenfalls in der Anforderungsanalyse findet.

1.2.4 Kommunikationsdiagramm

Zeigt die strukturelle Zusammenarbeit von Objekten anhand der zwischen ihnen ausgetauschten Nachrichten

1.2.5 Formale Beschreibung einer Kommunikation

- Bedingung
  - optional
  - Textform
- Nachrichtenname
  - Vorgängerbedingung: Botschaften vorher verschickt worden sein müssen
  - Sequenzausdruck (Nummerierung von Folgenachrichten)
  - Antwort (benannt, kann in weiteren Nachrichten als Argument übergeben werden)
  - Operationsname (inkl. Parameter)
- Synchronisation
  - synchron: Senderobjekt wartet auf Antwort
  - asynchron: Sender und Empfänger arbeiten unabhängig voneinander weiter
- Objektbeziehungen
  - «association»: Objekt-Beziehung besteht durch Assoziation im Klassendiagramm
  - «global»: Empfängerobjekt ist global bekannt
  - «local»: Empfänger ist innerhalb der sendenden Operationen bekannt
  - «parameter»: Empfängerobjekt ist dem Sender als Parameter übergeben worden
  - «self»: Senderobjekt empfängt die Nachricht
- Objektstatus
  - {new}: Objekt wird innerhalb des Szenarios erzeugt
  - {destroy}: Objekt wird innerhalb des Szenarios zerstört
  - {transient}: Erzeugung und Zerstörung innerhalb des Szenarios

1.2.6 Sequenzdiagramm

Zeigt Kommunikation/Interaktion zwischen Objekten unter Betonung des zeitlichen Ablaufs. Eignet sich jedoch nicht zur Darstellung komplexer Abläufe, auch wenn Kontrollstrukturen möglich sind.

1.2.7 Zustandsautomat

Einsatz dann, wenn die Reaktion eines Systems nicht unabhängig vom inneren Zustand ist. Ein Harel-Automat ermöglicht zusätzlich zu den Mealy- und Moore-Automaten

- bedingte Zustandsübergänge
- hierarchische Zustandsautomaten
  - Zusammenfassung von Zuständen in logische Blöcke zur besseren Übersicht
- nebenläufige Zustände
  - Nachrichten werden von mehreren unabhängigen Unterautomaten bearbeitet
  - aktueller Zustand besteht aus zwei Zuständen
  - Verlassen eines nebenläufigen Automaten erzwingt selbiges bei allen anderen
- Automaten mit Gedächtnis
  - Betreten eines (Unter-)Automation kann in den in dem Status erfolgen, in dem er verlassen wurde

Ereignisse, auf die der Automat reagiert, kann bspw der Empfang einer Nachricht oder das Ablaufen eines Intervalls sein.

Aktionen sind atomare (nicht unterbrechbare) Operationen die an Zustandsübergängen ausgeführt werden (u.a. entry, exit). Aktivitäten kommen während der "Verweildauer" zur Ausführung (do).

1.3 Überblick

1.3.1 Vorstudie

1. Grobbeschreibung des Systems
2. Projektbetroffene ermitteln
3. Geschäftsprozessmodellierung
4. Anforderungen von Nutzern und Controlling identifizieren

1.3.2 Anwendungsfall-Analyse

5. Geschäftsanwendungsfälle identifizieren
6. Anwendungsfälle essentiell beschreiben
7. Systemanwendungsfälle beschreiben
8. Anforderungen an Anwendungsfälle beschreiben
9. Geschäftsklassen identifizieren
10. Glossar anlegen
11. Anwendungsfall-/Ablaufmodell entwickeln
12. Systemschnittstellen beschreiben
13. Exploratives Schnittstellen Prototyping

1.3.3 Analyse des dynamischen Ablaufs

14. Erstellen der Kommunikationsdiagramme
15. Erstellen der Sequenzdiagramme
16. Erstellen der Zustandsdiagramme

2 Objektorienter Entwurf (OOD)

Soll Fehlerzahl reduzieren und Wiederverwendung fördern. Aber tiefe Vererbungsstrukturen sollten vermieden werden.

2.1 Komponenten

Aufteilung der Anwendung in fachlich zusammengehörende Komponenten mit einer definierten Schnittstelle nach außen. Optimalerweise sollte es später leicht möglich sein, ältere Komponenten durch kompatible neuere zu ersetzen.

Basis für Komponentenbildung sind Geschäftsklassendiagramm und Anwendungsfälle.

- Schnittellen sollten möglichst nur primitive Datentypen enthalten
- Verwendung fachlicher Standards
- Assoziationen durch Nachrichten- und Beobachtungsmechanismen ersetzen

2.2 Objektorientierte Entwurfskonzepte

2.2.1 Generische Klassen

Können sowohl einen Datentyp- als auch einen Konstantenparameter haben, die innerhalb der Klasse verwendet werden. Ein Beispiel wäre ein Stack, der immer nach dem selben Prinzip funktioniert jedoch zum Verwalten von Daten verschiedener Typen eingesetzt werden kann.

2.2.2 Container-Klassen

Anwendungsfall für generische Klassen. Sie dienen lediglich der Verwaltung von Objekten, was bei Klassendiagrammen immer stillschweigend angenommen wird.

2.2.3 Attribute und Operationen

Das Geheimnisprinzip kann durch die Sichtbarkeits-Modifikatoren umgangen werden. Trotzdem sollten Daten nur über die Standardisierten Schnittstellen ausgetauscht werden.

- public: sichtbar für alle Klassen
- protected: innerhalb der eigenen und aller Unterklassen
- private: innerhalb der eigenen Klasse
- friendly: (nur Java) innerhalb der Klassen eines Paketes

Zur Benennung von Attributen haben sich ebenfalls Standards durchgesetzt. Die vereinfachte Signatur eines Attributes ist bspw. als "Sichtbarkeit Attributname : Typ = Initialwert" definiert.

Bei Operationen werden zusätzlich zur Sichtbarkeit auch Parameter und Ergebnistypen angegeben.

2.2.4 Assoziationen

Die Lese-Richtung der Assoziation ist immer vom Ganzen zu den Teilen. Beim Spezialfall der Komposition wirken sich Operationen, die das Ganze betreffen, sich auf alle Teile aus.

3 Lösungsmuster

Ziel ist die Wiederverwendung existierender Lösungen (Aufwand sparen, Fehler vermeiden, Kosten minimieren).

3.1 Analysemuster

Ein Analysemuster ist eine Gruppe von Klassen mit feststehenden Verantwortlichkeiten unde Interaktionen.

3.1.1 Liste

- Komposition aus gleichartigen Teilen
- Teile sind dem Ganzen zugeordnet, können jedoch einzeln gelöscht werden

3.1.2 Exemplar

- einfache Assoziation (1:*)
- Existierte die Oberklasse nicht, so würden in den Objekten der untergeordneten Klassen viele Informationen redundant gespeichert
- Beispiel: Buch mit mehreren Ausgaben in einer Bibliothek

3.1.3 Baugruppe

- physische Objekte
- Komposition beteht über längeren Zeitraum, ein Teil kann jedoch ausgelöst udn einem anderen Ganze zugeordnet werden
- Beispiel: Fahrrad besteht aus Rad, Rahmen, Kette...

3.1.4 Rollen

- mehrere Assoziationen zwischen zwei Klassen
- Ein Objekt kann in Bezug auf unterschiedliche Objekte der anderen Klasse verschiedene Rollen einnehmen.
- Beispiel: Wegstück ist für Vorheriges Nachfolger, gleichzeitig für Nächstes Vorgänger

3.1.5 Wechselnde Rollen

Ein Objekt hat zu unterschiedlichen Zeitpunkten verschiedene Rollen mit anderen Attributen und Operationen.

3.1.6 Koordinator

Eine Koordinator-Klasse wird zur Beschreibung einer Situation eingesetzt und ersetzt eine mehrfache Assoziation. Sie hat jedoch selbst keine weiteren Attributen und Operationen.

3.2 Entwurfsmuster

Bewährte generische Lösung für ein immer wiederkehrendes Problem. Sie zeigen in rezeptartiger Weise das Zusammenspiel von Klassen, Objekten und Methoden.

3.2.1 Singleton

Muster für Klassen, von denen nur ein Objekt erzeugt werden soll.

Der Konstruktor wird als protected deklariert. Die Klasse enthält eine eine statische Variable zur Aufnahme eines Objektes vom jeweiligen Typ der Klasse. "Clients" holen sich über die statische Methode "instance" immer nur eine Referenz auf dieses Objekt, dass bei ersten "instance()"-Aufruf automatisch erzeugt wird.

3.2.2 Beobachter

Muster für Objekte (Subjekte), die andere Objekte (Observer) über Veränderungen informieren

Es werden zwei abstrakte Klassen Subjekt und Observer erstellt die abgeleitet werden. "Subject" bietet den Observern die Möglichkeit sich in eine Liste "einzutragen". Ändert sich beim Subject etwas ruft es seine Methode "notify" auf und Informiert hier alle Objekte in der Liste. Diese entscheiden dann, ob und wie sie reagieren.

3.2.3 Model-View-Controller

(nicht weiter beschrieben)

3.3 Frameworks

Im Gegensatz zu Klassenbibliotheken, die eher vorgefertigte Operationen - bspw. zum Drucken - zur Verfügung stellen, bieten Frameworks ein Anwendungs-Grundgerüst. Für bestimmte Problemstellungen, bspw. graphische Editoren, werden Operationen implementiert, z.B. "Symbol verschieben". Hierfür kann man nun nach Bedarf eigenen Code hinterlegen kann. Der wird dann vom Framework in dessen Ablauf "eingebaut".

Eines Frameworks schließt die gleichzeitige Verwendung einer Klassenbibliotheken natürlich nicht aus. In machen Fällen (Qt) ist die Grenze auch fließend.