Design by Contract

- Bertrand Meyer, ***Object-Oriented Software Construction***, Prentice Hall 1988 (2nd. Ed 1997)
- Bertrand Meyer, ***Applying Design by Contract***, IEEE Computer, vol. 25 #10, Oktober 1992

Grundprinzip

- Client-Server-Betrachtungsweise, wie RDD
  - *Kontrakt* - Voraussetzungen für Verwendung einer Klasse
  - Kontrakt schützt den Klienten (garantiert Erfüllung der Leistung) und den Anbieter (garantiert, daß sich der Klient an die Bedingungen hält)
- Formale Spezifikation, ausgehend von Spezifikation einzelner Operationen
  - Wenn die Funktion eines Moduls nicht spezifiziert wird, ist die Wahrscheinlichkeit gering, daß es die Funktion dann zur Verfügung stellt

Eigenschaften

- Basierend auf **Assertions** (logischen Zusicherungen)
  - Pre- und Postconditions für Operationen, sowie Invarianten (siehe Vererbungskapitel)
- Keine Richtlinien für erste Analysestufe
  - wie finde ich Klassen, Operationen
- Keine Richtlinien für Gliederung
  - welche Operationen wohin
  - Zerlegung oder Zusammenlegung von Klassen

Kontrakt

	Obligationen	Nutzen
Kunde	Am Flughafen 30 Minuten vor Abfahrt; Gepäck mitbringen, Ticket bezahlen	Kommt nach Chicago
Anbieter	Kunden nach Chicago bringen	Passagiere, die zu spät kommen, nicht transportierbares Gepäck mitbringen, oder nicht zahlen, müssen nicht befördert werden

Kontrakt für Software

Beispiel: Eintragen in Dictionary (kein Smalltalk-Dictionary!)

	Obligationen	Nutzen
Kunde	Garantieren, daß Tabelle nicht voll und Schlüssel kein Leerstring ist	Bekommt geänderte Tabelle mit neu eingetragenem Element (und passendem Schlüssel)
Anbieter	Gegebenes Element in Tabelle eintragen, unter dem gegebenen Schlüssel	Wenn die Tabelle voll oder der Schlüssel ungültig ist, muß nichts getan werden

Formelle Schreibweise

- Angabe von Pre- und Postconditions (siehe Kapitel Vererbung)
- Hier Eiffel-Syntax verwendet, aber Beschreibung ist eigentlich programmiersprachenunabhängig

```
put (x: ELEMENT; key: STRING) is
-- x eintragen, so daß über key zugreifbar
require
count <= capacity;
not key.empty
do ... Einfügealgorithmus ...
ensure
has (x);                    -- Inhaltsabfrage (#includes: in Smalltalk)
item (key) = x;             --item... Zugriff auf interne Datenstruktur
count = old count + 1       -- old ... Wert beim Aufruf
end
```


Beispiel

System für Chemiefabrik, mit Klassen TANK, LEITUNG, VENTIL, PUMPE, KONTROLLRAUM,...

Eigenschaften von TANK:

- Abfragen: ist_leer, ist_voll
- sonstige Abfragen: eingangsventil, ausgangsventil, füllstand, kapazität,...
- Befehle: füllen, leeren,...

Spezifikation

Operation `TANK.füllen:`

```
füllen is
-- Tank mit Flüssigkeit füllen
require
eingangsventil.offen;
ausgangsventil.geschlossen
deferred                    -- abstrakte Operation, keine Implementierung
ensure
eingangsventil.geschlossen;
ausgangsventil.geschlossen;
ist_voll
end
```

Bemerkungen

- Relativ genaue, formale Notation (logische Bedingungen, formuliert über Eigenschaften des Objekts)
- keine Vorgabe der Implementierung (aber Festlegung auf Eigenschaften der betroffenen Klasse)
- aber: eigentlich keine programmiersprachenspezifischen Konstrukte (Parameter, Instanzvariablen, Methoden gibt's überall)

Klasseninvarianten

- **Invariante**: Eine Bedingung, die für die Instanzen einer Klasse während ihrer gesamten Lebensdauer gilt
- Beispiel Dictionary:

```
invariant
0 <= count;
count <= capacity
```

- Beispiel Tank: Die Bedingung ist_voll wird als "ist beinahe voll" spezifiziert:

```
invariant
ist_voll =    (0.97 * kapazität <= füllstand) and
              (füllstand <= 1.03 * kapazität)
```

- Meyer: "eines der 3 oder 4 wichtigsten Konzepte in der OO Entwicklung"

Verwendung

- Dokumentation: Beschreibung der Situationen, in denen Operationen angewendet werden können
  - Implizit: Einschränkung von Operations*folgen* (selber Effekt wie Zustandsdiagramme in OMT)
- Testen: klare Vorgaben, welche Bedingungen für verschiedene Operationen getestet werden müssen
  - Idealzustand: Während Debugging werden die Programme mit "eingeschalteten" Assertions compiliert - automatische Laufzeitüberprüfung f. Spezifikationen

Vererbung

- Subklassenbildung entspricht **subcontracting:** der Kontrakt einer Subklassen muss alle Klauseln des Kontrakts der Oberklasse erfüllen (Vorsicht, nicht die deutsche Bedeutung des Worts Subkontraktor)
- Nur dann ist die Einhaltung des gesamten Kontrakts garantiert
  - Vorbedingungen: gleich oder schwächer
  - Nachbedingungen: gleich oder stärker
    ---> *Kontravarianz*
- Vererbung als Spezifikationsvererbung
  - Verwandt

Zusammenfassung

- Semantik der Operationen (Schnittstelle) steht im Vordergrund
- Textuell, keine aufwendige Notation, exakt im Ausdruck
- Prinzip ist programmiersprachenunabhängig
  - Unterstützung möglich (dzt. nur in Eiffel), aber nicht Voraussetzung (bei anderen Methoden ja auch nicht)
- Probleme:
  - keine Unterstützung für Analyse
  - keine Notation für Strukturierung im großen Rahmen
  - Keine Darstellung komplexer Abläufe, nur auf Ebene von Einzelaufrufen

Kontrakte in der Analyse

- Haim Kilov, Helen Mogill, Ian Simmonds, ***Invariants in the Trenches***, in Kilov, Harvey (Eds.) *Object-Oriented Behavioral Specifications,* Kluwer Publishing 1996
- *These*: Die Verwendung von Assertions in Zusammenhang mit "generic Relationships" (ER-Modell, OMT-Assoziationen) reicht als Analysemethode aus
- die Ergebnisse sind für die Benutzer (Kunden, Anwender, die den Modellierungsinput liefern) verständlich
  - verbesserte Kommunikation während der Analyse

Hintergrund: Haim Kilov, James Ross, ***Information Modeling: An Object-Oriented Approach***, Prentice Hall, 1994