DESIGN BY CONTRACT

Studientag 01853

Moderne Programmiertechniken und -methoden

WARUM DBC?

- Nachteil des interfacebasierten Ansatzes
  - es werden keine semantischen, sondern nur syntaktische Vorgaben gemacht
- Prinzip des DbC betrachtet Programme als eine Menge kommunizierender Komponenten, deren Interaktion auf genauer Spezifikation der gegenseitigen Obliegenheiten beruht

➡ Verträge

FORMALER HINTERGRUND

- P {A} Q
- lies: wenn P vor der Ausführung von A wahr ist, dann ist danach auch Q wahr
- P ist Vor- Q ist Nachbedingung, beides sind Zusicherungen
- A ist eine Folge von Anweisungen (also ein Programm)

RECHTE UND PFLICHTEN

- Einhaltung der Vorbedingungen ist Pflicht des Aufrufers
- Einhaltung der Nachbedingungen ist Pflicht des Aufgerufenen
- beide werden in der Analysephase konkret auf Klassen und Methoden angewendet

INVARIANTEN

- Zusicherungen die durchgängig erfüllt sein müssen (meist Klasseninvarianten)
- können bei Ausführung einer Methode temporär verletzt werden (mangels Transaktionskonzept)
- Änderungen der Klasse darf keine Auswirkung auf Invariante haben

INVARIANTEN II

- Bezug auf internen Zustand: Implementationsinvarianten
  - bei Definition muss i.d.R. auf die Werte vor dem Aufruf zugegriffen werden können

ZUSICHERUNGEN I

- in einfachen Fällen übernehmen Vor- und Nachbedingungen die Funktion eines Typsystems
- effektiv: Beschränkung des Wertebereichs von Parametern
- kann statisch vom Compiler überprüft werden

ZUSICHERUNGEN II

- Zusicherungen gehen aber darüber hinaus:
  - Abhängigkeiten zwischen Parametern
  - Abhängigkeiten zwischen Ein- und Ausgabe
    - z.B. Ungenauigkeit bei Näherungsalgorithmen

UMSETZUNG VON DBC

- de facto werden Bedingungen heute im Quellcode notiert
  - keine Redundanz
  - Vermischung mit Code
    - kann fehlerhaft sein
    - kann Seiteneffekte haben
    - kann zirkuläre Abhängigkeiten haben

PRÜFUNG DER ZUSICHERUNGEN

- dynamisch
  - liefert keinen allgemeinen Beweise sondern nur fallweise Bestätigung
  - muss aufgrund erhöhter Ausführungskosten Abschaltbar sein
- Fernziel: statische Programmverifikation zur Übersetzungszeit (mit Theorembeweiser)

  ➡ Verified DbC

PRÜFUNG VON INVARIANTEN

- Invarianten sind Vor- und Nachbedingungen zugleich
- müssen nach jeden externen Methodenaufruf geprüft werden
- Problem Aliasing: Attribute können sich ggf. ohne Zutun der Klasse ändern:
  - Prüfung der Invarianten vor dem Methodenaufruf zur Absicherung
- Problem in Java: private-Aufrufe durch andere Objekte

PRÜFUNG VON VORBEDINGUNGEN

- defensiv: Prüfung durch aufgerufene Klasse
- aktiv: Prüfung durch Aufrufer
- er ist nach DbC für Einhaltung verantwortlich
- kann auf Fehler entsprechend reagieren
- benötigt dann aber Zugriff auf zugrunde liegende (Prüf-)Methoden

ZUSICHERUNGEN UND VERERBUNG

- Substituierbarkeit setzt voraus, dass Vertrag bei Vererbung Gültigkeit behält
  - Subklasse darf Vorbedingungen aufweichen
  - Subklasse darf Nachbedingungen einschränken

LOGISCHE GRUNDLAGE

- stärkere Bedingungen implizieren schwächere
- $\text{VorB}_{\text{Superklasse}} \rightarrow \text{VorB}_{\text{Subklasse}}$
- $A \rightarrow A \vee B$ (wenn A wahr ist, ist auch A oder B wahr)
- $\text{VorB}_{\text{Superklasse}} \rightarrow \text{VorB}_{\text{Superklasse}} \vee \text{VorB}_{\text{Subklasse}}$
- Subklasse darf Vorbedingungen aufweichen

LOGISCHE GRUNDLAGE II

- $\text{NachB}_{\text{Superklasse}} \leftarrow \text{NachB}_{\text{Subklasse}}$
- $A \leftarrow A \wedge B$ (wenn A und B wahr sind, ist auch A wahr)
- $\text{NachB}_{\text{Superklasse}} \leftarrow \text{NachB}_{\text{Superklasse}} \wedge \text{NachB}_{\text{Subklasse}}$
- Subklasse darf Nachbedingungen einschränken

ZUSICHERUNGEN UND VERERBUNG II

- Invarianten werden bei Vererbung mittels Konjunktion zusammengeführt

- Achtung: wenn Vorbedingung der Subklasse restriktiver als die der Superklasse ist, wird diese Verschärfung durch die oder-Verknüpfung aufgehoben

SPEZIFIKATIONSSPRACHEN ALLGEMEIN

- basieren i.d.R. auf Prädikatenlogik erster Stufe (Prädikate sind quasi Aussagen über Entitäten)
- sind atemporal und daher für DbC ungeeignet
  - zu prüfendes Verhalten ändert sich über die Zeit
- Sichtweise der Spezifikation: deskriptiv (schreiben be)
- Sichtweise des Programms: präskriptiv (schreiben vor)

DBC IN EIFFEL

- Vor- und Nachbedingungen mittels require und ensure
- Problem: Methoden mit Seiteneffekten sind in den Klauseln zwar nicht erlaubt, wird aber vom Compiler nicht geprüft
- Meyer fordert, dass Vorbedingungen nicht verschärft werden dürfen
  - damit auch Unveränderbarkeit von Invarianten
  - Umgehen mittels programmatischem Prädikat

DBC IN JAVA

- seit Version 1.4: assert
  - kein Schutz vor Aufruf von Methoden mit Seiteneffekten
    - aber: ggf. werden Seiteneffekte zur Speicherung alter Attributwerte benötigt
- alternativ: Nutzen von org.junit.Assert.*
  - vor JUnit 4 inkompatibel mit assert, da eigene AssertionFailedError geworfen wurden

DBC IN JAVA II

- Java Modelling Language (JML)
  - erlaubt formale, semantische Spezifikation von Interfaces
  - berücksichtigt Seiteneffekte (Freiheit von solchen durch pure-Statement)
  - abstrakte Spezifikation mit Modellvariablen und -methoden
    - diese erhalten ihren Wert in Implementierungen durch Abstraktionsfunktionen

DBC UND TESTS

- aus den Zusicherungen können automatisiert Testfälle erzeugt werden
- Tests und DbC können nur Fehler nachweisen und keine Fehlerfreiheit

PROBLEME DES DBC

- Überprüfung der Einhaltung einer Spezifikation ist nicht trivial
  - erfolgt i.d.R. nur Fallweise
- Formulierung der Zusicherungen kann fehlerhaft sein
- die den Zusicherungen zugrunde liegenden Anforderungen können falsch sein