Lernzettel

Spezifikations- vs. Implementierungsbasierte Entwicklung: Design by Contract, Assertions und Validierung

Universität: Technische Universität Berlin
Kurs/Modul: Algorithmen und Datenstrukturen
Erstellungsdatum: September 6, 2025

Lernzettel: Spezifikations- vs. Implementierungsbasierte Entwicklung

(1) Zielorientierung und Grundidee. Spezifikation beschreibt, *was* ein Baustein leisten soll, unabhängig davon, wie es intern umgesetzt wird. Implementierung beschreibt die konkrete Umsetzung und die Art, wie die Spezifikation erfüllt wird. Die getrennte Sicht erleichtert Korrektheit, Wartung und Austausch von Komponenten in Algorithmen und Datenstrukturen.

(2) Design by Contract (DbC). DbC führt Verträge zwischen Client und Supplier ein:

- *Präcondition* (Pre) – Voraussetzungen, die vor der Ausführung erfüllt sein müssen.
- *Postcondition* (Post) – Garantien nach Beendigung der Ausführung.
- *Invariants* – Zustands-Eigenschaften, die während der Ausführung (insbesondere in Schleifen/Klassen) stets gelten.

Formalität: oft per Hoare-Triple notiert

$$\{\, P \,\}\; C \;\{\, Q \,\},$$

dabei ist P die Präbedingung, C die auszuführende Anweisung bzw. der Block, und Q die Postbedingung.

(3) Assertions. Assertionsprüfungen setzen zu Laufzeit Bedingungen, die während der Entwicklung helfen, Korrektheit anzutesten:

$$\text{assert } \phi \quad \text{mit } \phi \text{ eine Bedingung über Variablen.}$$

Nützliches Werkzeug während der Entwicklung; oft während der Produktion deaktivierbar oder optimierbar. Beispiele:

- In Java: `assert x > 0;`
- In Java = DbC-Ansatz zusätzlich durch JML/Verträge ausdrückbar: `// requires x > 0; ensures result ≥ x;`

(4) Validierung vs. Verifikation.

- Validation (Richtige Software): Prüft, ob das System den tatsächlichen Anforderungen entspricht (Tests, Abnahmetests, Spezifikationsabgleich).
- Verifikation (Korrektheit): Beweist formell, dass ein System gemäß seiner Spezifikation funktioniert (Hoare-Logik, Beweisführung).

Beispiele:

$$\{P\}C\{Q\}, \quad \text{mit } P \Rightarrow Q \text{ im idealen Fall – formaler Beweis.}$$

(5) Praktische Muster in Algorithmen und Datenstrukturen. - Verträge für Datenstrukturen: Invariante z. B. für einen Stack

$$\text{Invariante: size} \geq 0 \quad \wedge \quad \text{top enthält das oberste Element, falls size} > 0.$$

Aktionen:

$$\text{Push}(x): \quad \text{size}' = \text{size} + 1, \quad \text{top}' = x.$$

$$\text{Pop}(): \quad \text{size}' = \text{size} - 1, \quad \text{neues Top entsprechend.}$$

- Prä-/Postbedingungen für Algorithmen:

$$\text{Pre: start} \in V, \text{ Zustand erlaubt}, \quad \text{Post: Ergebnis korrekt gemäß Spezifikation.}$$

(6) Beispielhafte DbC-Notation in Pseudocode.

```
function max(a,b)
  // requires: a,b
  // ensures: result = max(a,b)
  if a >= b then return a else return b
```

(7) Beispiel: Graph- oder Pfad-Algorithmus mit Contracts. Betrachten wir eine Funktion, die alle Knoten erreichbar von einem Startknoten ermittelt.

```
function reachable(Graph g, Node s)
  requires: s  g.nodes
  ensures:  v  ReachableFrom(g, s) -> v  g.nodes
```

Wichtige Vertragskomponenten: - Prä: Startknoten gehört zum Graphen. - Post: Rückgabe enthält genau die Menge der erreichbaren Knoten (Relation zur Transitive Closure).

(8) Hinweise zu Umsetzung und Best Practices. - Contracts sollten klein, nachvollziehbar und unabhängig von Implementierungsdetails sein. - Invariantensicherung ist zentral bei Datenstrukturen (z. B. Bäume, Mengen-Strukturen, Heaps). - Assertions helfen beim Debugging, aber Leistungs-Overhead beachten. - Kombination aus statischer Verifikation (Beweisen) und dynamischer Validierung (Tests, Property-Based Testing) ist sinnvoll. - Dokumentation der Verträge erleichtert Wartung und Weiterentwicklung von Algorithmen.

(9) Kurzvergleich: Spezifikation vs. Implementierung in der Praxis. - Spezifikation: Was soll das Ergebnis sein, unabhängig von Implementation. - Implementierung: Wie wird das Ziel technisch erreicht, inklusive Optimierungen. - DbC verbindet beide Welten durch klare Verträge, die sowohl Tests als auch Beweise erleichtern.