Formale Spezifikation

Andreas Zeller

Lehrstuhl Softwaretechnik
Universität des Saarlandes, Saarbrücken







Warum Spezifikation?

Spezifikation ist Teil des *Feinentwurfs:*

Für jede Funktion wird beschrieben, *was* sie tut
(aber nicht unbedingt, *wie* sie es tut)

Letzte Phase vor der Implementierung (eben jener Funktion)







Anforderungen

Jede Spezifikation soll

- *vollständig* sein – jeder Aspekt des Systemverhaltens wird abgedeckt
- *widerspruchsfrei* sein – damit klar ist, was implementiert werden soll
- auch *unvorhergesehene Umstände* beschreiben, um die Robustheit zu steigern.

⇒ Einsatz von *Spezifikationsverfahren*







Spezifikationsverfahren

Man unterscheidet folgende *Spezifikationsverfahren*:

- Informale Spezifikation
- Exemplarische Spezifikation
- Formale Spezifikation







Informale Spezifikation

Für jede Funktion wird in kurzer *Prosa* beschrieben, was sie tut.

Die Beschreibung sollte zumindest die Rolle der Parameter und des Rückgabewertes sowie ggf. Seiteneffekte enthalten.

- ✔ Weitverbreitetes Spezifikationsverfahren
- ✔ Gut für *Dokumentation* geeignet

- ✘ Unexakt
- ✘ Einhaltung der Spezifikation schwer nachweisbar







Informale Spezifikation – Beispiel

- `Reactor.cooking()` liefert `true`, wenn die Temperatur des Reaktors $100°$C erreicht.







Informale Spezifikation – Beispiel

- `Reactor.cooking()` liefert `true`, wenn die Temperatur des Reaktors $100°$C erreicht.

Probleme:

- Was bedeutet „erreicht“?
- Was passiert bei Temperaturen unter $100°$C?







Exemplarische Spezifikation

Testfälle beschreiben *Beispiele* für das Zusammenspiel der Funktionen samt erwarteter Ergebnisse

- ✔ Formales (da am Code orientiertes) Spezifikationsverfahren, dennoch leicht verständlich
- ✔ Nach der Implementierung dienen die Testfälle zur Validierung
- ✘ Nur exemplarische Beschreibung (und Validierung) des Verhaltens

Durch *Extreme Programming* populär geworden

Exemplarische Spezifikation – Beispiel


```
Reactor.set_temperature(99);
assert (!Reactor.cooking());

Reactor.set_temperature(100);
assert (Reactor.cooking());
```







Exemplarische Spezifikation – Beispiel


```
Reactor.set_temperature(99);
assert (!Reactor.cooking());

Reactor.set_temperature(100);
assert (Reactor.cooking());
```

Probleme:

- Deckt nicht alle Möglichkeiten ab
- Ausführung setzt ggf. echte (hier: kochende) Hardware voraus







Formale Spezifikation

Mittels einer *formalen Beschreibungssprache* wird die Semantik der Funktionen exakt festgelegt.

- ✔ Exakte Beschreibung der Semantik
- ✔ Ausführbare Spezifikationssprache kann als Prototyp dienen
- ✔ Möglichkeit des Programmbeweises

- ✘ Erhöhte Anforderungen an Verwender
- ✘ Aufwendig







Formale Spezifikation – Beispiel in Z

Das Schema *Reactor* beschreibt den Zustand eines Reaktors:

$$\begin{array}{|l} \textit{Reactor} \\ \hline \textit{temperature} : \mathbb{N} \\ \hline \textit{temperature} \geq 0 \end{array}$$

Formale Spezifikation – Beispiel in Z

Das Schema *Reactor* beschreibt den Zustand eines Reaktors:

Reactor

$$temperature : \mathbb{N}$$

$$temperature \geq 0$$

Das Schema *cooking* beschreibt eine *Eigenschaft* des Reaktors:

cooking

$$Reactor$$

$$temperature \geq 100$$

Warum formale *Spezifikation?*

Informale Spezifikation (= natürliche Sprache) ist

- verrauscht (*noisy*), d.h. Einsatz von verschiedenen Wörtern, die dasselbe bezeichnen – z.B. eine „nicht-leere Folge“ von Elementen vs. „ein oder mehr“ Elemente.
- mehrdeutig – z.B. „Ereignis A passiert nach Ereignis B“ (was ist „passiert“? „nach“?)
- widersprüchlich – da nicht (formal) nachweisbar

Dasselbe gilt auch für *semi-formale* Spezifikationen (= Bilder), sofern sie nicht von präzisen Annotationen begeleitet werden

Exemplarische Spezifikationen wiederum decken per Definition nur einen Teil der Anforderungen ab







Spezifikationsverfahren

Formale Spezifikationsverfahren werden in drei Kategorien unterschieden:

Eigenschaftsorientiert – entweder

- *axiomatisch* – Objekte werden aus Typen konstruiert und durch *logische Zusicherungen* spezifiziert
- *algebraisch* – Definiert *Algebren* über Objekte und deren Funktionalität

Modellorientiert – beschreibt ein *Modell* des Systems mit einer reichen formalen Sprache (VDM, Z)

Automatenorientiert – beschreibt Zustände und Übergänge des Systems (z.B. Petri-Netze, Statecharts)

Übersicht

- Modellorientierte Spezifikation mit Z
- Programmbeweise mit Z
- Von Z zu Code
- Axiomatische Spezifikation mit Hoare-Logik und temporaler Logik
- Model Checking