

Softwareentwicklung in Großprojekten

Klausur WS 2011/12 Braindump

6. März 2017

Fehler oder fehlende Fragen bitte melden: he29heri@stud.informatik.uni-erlangen.de

Nach einer dreiviertel Stunde fertig, wenn man alles wusste.

1 Wissensfragen

Unterschied von beabsichtigen zu tatsächlichem Verhalten failure

Unterschied von spezifizierten zu realisiertem Verhalten fault

4 weitere Eigenschaften von Software-Anforderungen (außer Eindeutigkeit, Konsistenz, Verfolgbarkeit) siehe Skript

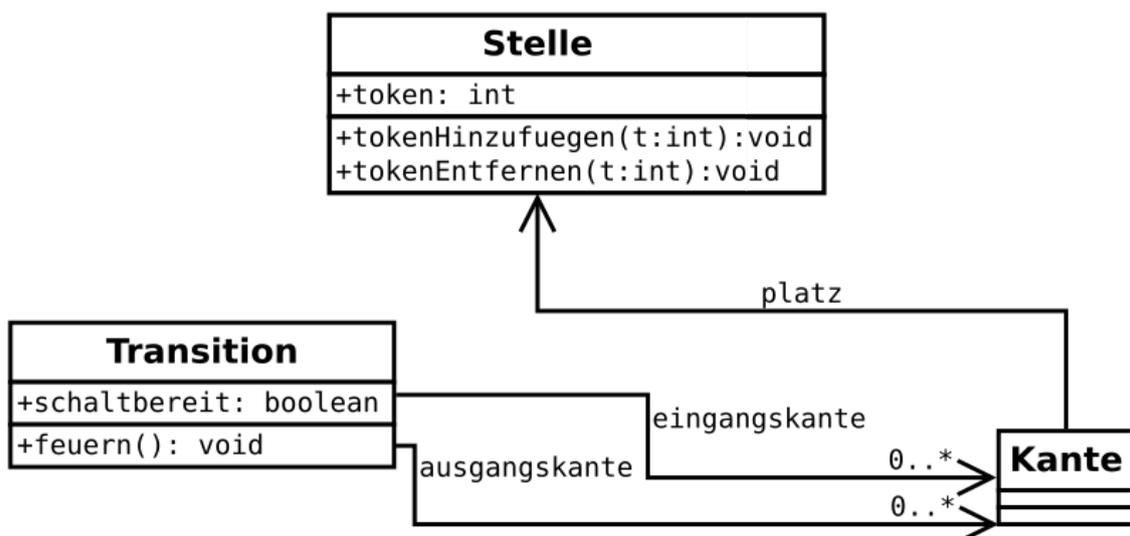
3 weitere Methoden zur Anforderungsermittlung (außer Interview) siehe Skript

4 weitere Punkte der Volerekarte (außer Anforderungsnummer und Urheber) siehe Skript

3 weitere Grundsätze der Softwareergonomie (außer Aufgabenangemessenheit und Individualisierbarkeit) siehe Skript

Benennen Sie die Kohäsionsart folgender Definition: Eine Reihe von Aktionen auf unterschiedlichen Daten in zeitlicher Abfolge. siehe Skript

2 OCL



Anzahl der Tokens an einer Stelle muss größer oder gleich 0 sein.

```
context Stelle:
inv: self.token >= 0
```

Eine Transition kann nur feuern wenn sie schaltbereit ist, d. h. `schaltbereit = true`.

```
context Transition::feuern(): void
pre: self.schaltbereit = true
```

Wird `tokenHinzufuegen(t: int)` mit $t > 0$ aufgerufen, dann wird `token` um t erhöht.

```
context Transition::tokenHinzufuegen(t : int): void
post: t > 0 implies token = token@pre + t
```

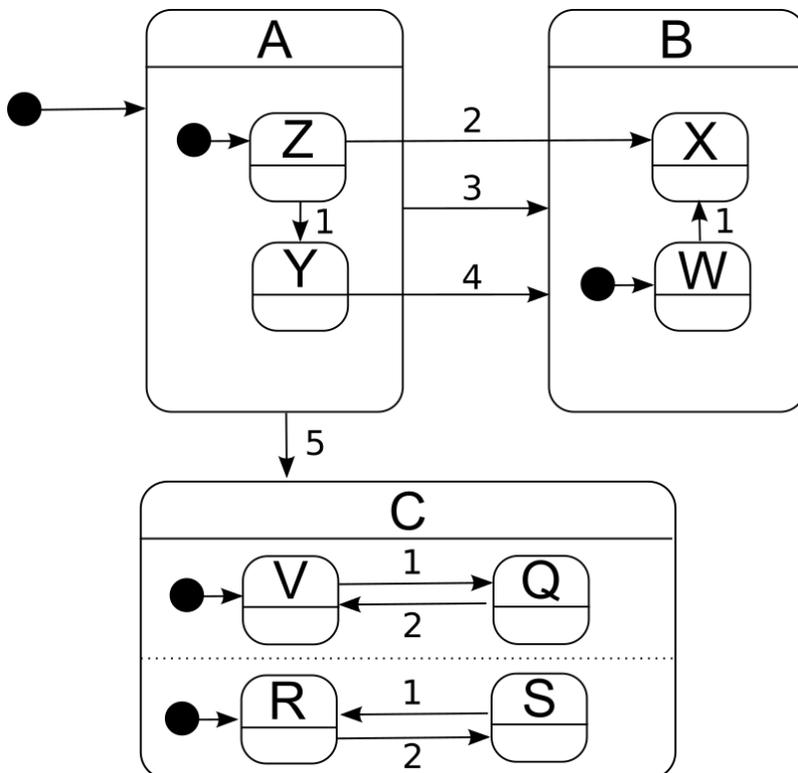
Jede Kante ist einer Stelle und einer Transition zugeordnet.

```
context Kante:
inv: self.platz->notEmpty() AND
    Transition.allInstances()->exists(t : Transition |
        t.einkangskante->includes(self)
    OR t.ausgangskante->includes(self))
```

Das Attribut `schaltbereit` ist nur dann auf `true` gesetzt, wenn auf jedem Eingabepplatz der Transition mindestens so viele Token liegen, wie es Eingangskanten zur Transition gibt.

```
context Attribut:
inv: schaltbereit = true implies
    eingangskante->forAll(k : Kante |
        k.platz.token >= eingangskante->select(p : Kante |
            p.platz = k.platz)->size())
```

3 Statechart



Zeichnen Sie einen äquivalenten endlichen Automaten.

4 Entwurfsmuster

Abbildung mit 3D-Objekten gezeigt.

Entwurfsmuster Kompositum

Kategorie objektorientiertes Strukturmuster

UML-Diagramm zeichnen siehe Skript

Sequenziagramm zeichnen siehe Skript

5 Testen

2 weitere Arten von Black-Box-Tests (ohne Äquivalenzklassentest)

2 weitere Arten von Systemtests (ohne Stresstest)

Folgender Code gegeben:

```
function foo(int x, int y, int z) {
    int erg = z;

    if (x < z && y < z) {
        if (x < y) {
            erg = x;
        } else {
            erg = y;
        }
    }

    return erg;
}
```

(x, y, z) mit $x, y, z \in \{1, \dots, 5\}$

Tests für Anweisungsüberdeckung (1, 2, 3), (1, 1, 3)

Tests für Verzweigungsüberdeckung (1, 2, 3), (1, 1, 3), (5, 5, 1)

Tests für Pfadüberdeckung (1, 2, 3), (1, 1, 3), (5, 5, 1)

6 Refactoring

Es waren zwei UML-Diagramme gegeben und es sollte der Refactoring-Typ angegeben werden.

1. Diagramm Klasse extrahieren, siehe Skript

2. Diagramm Methode integrieren, siehe Skript

Nennen sie zwei weitere Typen (ohne Diagramm, nur angeben) siehe Skript