technische universität dortmund

Fakultät für Informatik
Lehrstuhl 14 für Software Engineering
Prof. Dr. Falk Howar
Simon Dierl, M. Sc.
Softwarekonstruktion WiSe 18 / 19

# Übungsblatt 5

**Ausgabe:** 10.12.2018 **Abgabe:** 20.12.2018, 12:00 Uhr

## Organisatorisches

Über die vorlesungsfreien Tage um Weihnachten und Neujahr werden wir eine optionale Zusatzübung zum Meta-Modellierungswerkzeug *Cinco* anbieten. Durch eine erfolgreiche Bearbeitung können zusätzliche Punkte erworben werden, die zur Zulassung notwendige Schwelle von 58 Punkten (50 % der in regulären Übungen erreichbaren Punkte −2) bleibt jedoch unberührt. Ansonsten gelten die Hinweise von Übungsblatt 4.

## Hausübung

### Aufgabe 5.1 (Qualitätsmetriken) 4 P.

Betrachten Sie den Quellcode dieser Java-Klasse zum Kanalmanagement eines Fernsehers unter den verschiedenen Qualitätsmetriken, die in der Vorlesung besprochen wurden. Betrachten Sie den Konstruktor als normale Methode.

```
1  package edu.udo.ls14.swk;
2
3  public final class TVChannels {
4
5    private final double[] carrierFrequency;
6    private final String[] channelName;
7    private final boolean[] encrypted;
8
9    public TVChannels(int nChannels) {
10     this.carrierFrequency = new double[nChannels];
11     this.channelName = new String[nChannels];
12     this.encrypted = new boolean[nChannels];
13   }
14
15   public void autodiscoverChannels(TVTuner tuner, double start, double end) {
16     int currentChannel = 0;
17     for (double freq = start; freq <= end && currentChannel <
18         this.carrierFrequency.length; freq += tuner.RESOLUTION) {
19       tuner.tuneTo(freq);
20       if (tuner.dataReceived()) {
```

```
21          this.carrierFrequency[currentChannel] = freq;
22          this.encrypted[currentChannel++] =
23              tuner.estimateNoise() > .95d;
24        }
25      }
26    }
27
28    public void nameChannel(int channel, String name) {
29      this.channelName[channel] = name;
30    }
31
32    public void configureChannel(int channel, double freq,
33        byte[] encryptionKey) {
34      this.carrierFrequency[channel] = freq;
35      this.encrypted[channel] = encryptionKey != null;
36    }
37
38    public String getChannelName(int channel) {
39      if (channel < 0 || channel >= this.channelName.length ||
40          this.channelName[channel] == null) {
41        return "";
42      }
43      return this.channelName[channel];
44    }
45
46    public void swap(int one, int other) {
47      if (one == other) { return; }
48
49      if (this.encrypted[one] || this.encrypted[other]) {
50        throw new UnsupportedOperationException(
51          "TVT-00000124: key manager does not support channel change");
52      }
53
54      this.carrierFrequency[one] ^= this.carrierFrequency[other];
55      this.carrierFrequency[other] ^= this.carrierFrequency[one];
56      this.carrierFrequency[one] ^= this.carrierFrequency[other];
57      String s = this.channelName[one];
58      this.channelName[one] = this.channelName[other];
59      this.channelName[other] = s;
60    }
61
62    public void tuneToChannel(TVTuner tuner, int channel,
63        byte[] encryptionKey) {
64      if (this.encrypted[channel] && encryptionKey == null) {
65        throw new MissingKeyException();
```

```
66      }
67      tuner.tuneTo(this.carrierFrequency[channel]);
68      if (this.encrypted[channel]) {
69        tuner.supplyKey(encryptionKey);
70      }
71    }
72
73    @Override
74    public String toString() {
75      StringBuilder sb = new StringBuilder(this.getClass().getName())
76        .append("(");
77      for (int i = 0; i < carrierFrequency.length; i++) {
78        sb.append(this.carrierFrequency[i]).append(" Hz, \"")
79          .append(this.channelName[i]).append("\", ")
80          .append(this.encrypted[i] ? "paid; " : "free; ");
81      }
82      return sb.append(")").toString();
83    }
84  }
```

(a) Ihr Vorgesetzter hat in einem Buch gelesen, dass Klassen mit *wenig* Zeilen Code wünschenswert sind. Mit welchem „billigen“ Trick können Sie mit minimalem Aufwand die LOC reduzieren? 1 P.

(b) Bestimmen Sie jetzt *mit der in der Vorlesung vorgestellten Formel* die LCOM der Klasse. 2 P.

(c) Die LCOM offenbart, dass die Funktionalität der Klasse nicht sehr kohäsiv zu sein scheint. Welche Funktionalität sollte in eine andere Klasse ausgelagert werden, um die LCOM zu verringern? 1 P.

## Aufgabe 5.2 (Kontrollflussgraphen) 8 P.

Betrachten Sie jetzt diese zwei Methoden zum Sortieren von Daten. Die kurzen Hilfsmethoden können Sie für die folgende Analyse ignorieren.

```
1  package edu.udo.ls14.swk;
2
3  public final class Sorting {
4
5    private Sorting() { throw new UnsupportedOperationException(); }
6
7    public static void quickSort(long[] data) { quickSort(data, 0, data.length - 1); }
8
9    private static void swap(long[] data, int i, int j) {
```

```
10      data[i] ^= data[j];
11      data[j] ^= data[i];
12      data[i] ^= data[j];
13    }
14
15    private static void quickSort(long[] data, int left, int right) {
16      if (left < right) {
17        int l = left;
18        int r = right - 1;
19        long pivot = data[right];
20
21        while (l < r) {
22          for (; data[l] < pivot && l < right - 1; l++) {}
23          for (; data[r] >= pivot && r > left; r--) {}
24          if (l < r) { swap(data, l, r); }
25        }
26        if (data[l] >= pivot) { swap(data, l, right); }
27
28        quickSort(data, left, l);
29        quickSort(data, l + 1, right);
30      }
31      return;
32    }
33
34    public static void mergeSort(long[] data) {
35      long[] from = data;
36      long[] to = new long[data.length];
37
38      for (int run = 2; run <= data.length * 2; run *= 2) {
39        for (int start = 0; start < data.length; start += run) {
40          final int end = Math.min(start + run, data.length);
41          final int mid = Math.min(start + run / 2, end);
42          int left = start;
43          int right = mid;
44          int target = start;
45
46          while (target < end) {
47            if ((left < mid && right < end && from[left] < from[right]) || right >= end) {
48              to[target++] = from[left++];
49            } else {
50              to[target++] = from[right++];
51            }
52          }
53        }
54
```

```
55        long[] swap = to;
56        to = from;
57        from = swap;
58      }
59      System.arraycopy(from, 0, data, 0, data.length);
60      return;
61    }
62  }
```

(a) Intuitiv können wir einen Kontrollflussgraphen vereinfachen, indem wir Abfolgen von nicht-verzweigenden Operationen in einem Knoten zusammenfassen. Diese Intuition lässt sich formalisieren. Sei dazu $G = (V, E)$ ein Kontrollflussgraph. 2 P.

- $\mathbf{V} = (v_1, \ldots, v_n)$ heißt *Pfad* in $G$, wenn

$$\{v_1, \ldots, v_n\} \subseteq V \text{ und } \{(v_1, v_2), \ldots, (v_{n-1}, v_n)\} \subseteq E.$$

- Ein Pfad heißt *verzweigungsfrei*, wenn $v_2, \ldots, v_n$ Ein- und Ausgrad 1 haben und $v_1$ Ausgrad 1 hat[1].
- Ein verzweigungsfreier Pfad $\mathbf{V}$ heißt *maximal*, wenn es keinen längeren verzweigungsfreien Pfad gibt, der $\mathbf{V}$ enthält.
- Die Menge aller maximaler verzweigungsfreier Pfade in $G$ sei $\{\mathbf{V}^1, \ldots, \mathbf{V}^k\}$. Das erste Element eines verzweigungsfreien Pfades $\mathbf{V}$ sei $i(\mathbf{V})$, das letzte $\ell(\mathbf{V})$.
- *Verschmelzen* wir einen solchen Pfad $\mathbf{V}$ in $G$, erhalten wir

$$m^{(\mathbf{V})}(G) = ([V \setminus \mathbf{V}] \cup \{i(\mathbf{V})\},\\ (E \setminus \{(v, w) \mid v \in \mathbf{V}, (v, w) \in E\}) \cup \{(i(\mathbf{V}), w) \mid (\ell(\mathbf{V}), w) \in E\}).$$

- Ein *reduzierter* Kontrollflussgraph sei dann

$$G' = m^{(\mathbf{V}^1)}(m^{(\mathbf{V}^2)}(\cdots(m^{(\mathbf{V}^k)}(G))\cdots)).$$

Beweisen Sie jetzt, dass die zyklomatische Komplexität $c(G) = c(G')$, d. h. dass ein reduzierter Graph genügt, um die Komplexität zu untersuchen. In einem solchen Graphen beschriften wir einen Knoten mit allen Statements, die in ihn verschmolzen wurden.

[1] Es ist möglich, das Kriterium zu lockern, indem wir für $v_n$ nicht den Ausgrad 1 fordern. Dies wird hier aber *nicht* betrachtet.

(b) Zeichnen Sie den reduzierten Kontrollflussgraphen (d. h. mit allen möglichen Verschmelzungen) für die Methode `quickSort(long[], int, int)`. Um einen Kontrollflussgraphen für eine Java-Programm zu erhalten, gehen Sie wie folgt vor: 2 P.

- Eine Zuweisung wie `i = 3`, `i /= 10` oder `a[i] = i` entspricht einer Zuweisung in While-Programmen.
- Aufrufe und Befehle wie `doIt()` und `return` werden ebenfalls wie Zuweisungen behandelt.
- Die Behandlung von `if` und `while` ist auch bekannt.
- Eine For-Schleife der Form `for (assignment; test; update) { code; }` lässt sich in der Form `assignment; while(test) { code; update; }` als While-Schleife ausdrücken.

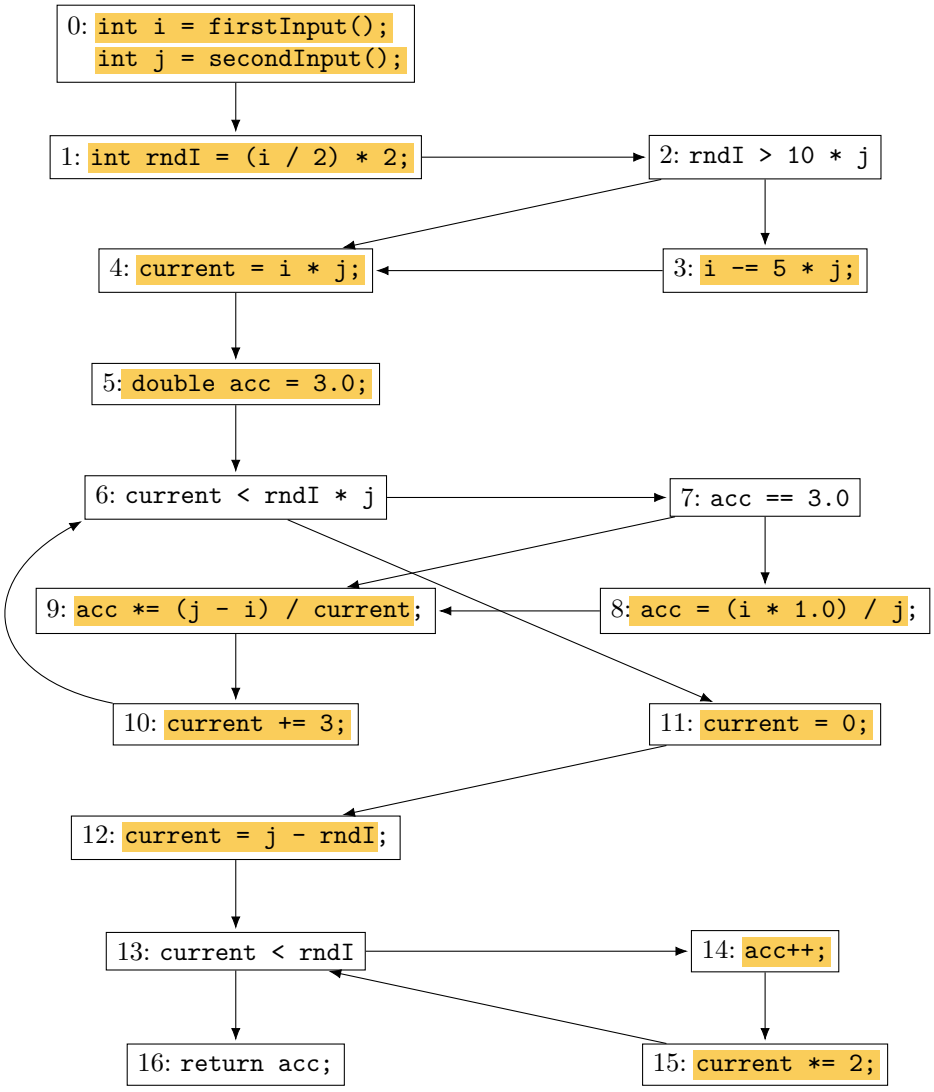
(c) Nutzen Sie den Graphen, um die zyklomatische Komplexität der Methode auszurechnen. 1 P.

(d) Bestimmen Sie ebenfalls den reduzierten Kontrollflussgraphen für die Methode `mergeSort(long[])`. 2 P.

(e) Nutzen Sie den Graphen, um auch die zyklomatische Komplexität dieser Methode auszurechnen. 1 P.

### Aufgabe 5.3 (Fixpunktsuche) 8 P.

Betrachten Sie folgenden Kontrollflussgraph für eine Methode:




Jede Zuweisung ist eindeutig durch die Nummer ihres Knotens identifiziert. Auf diesem Graphen sollen jetzt eine Reaching Definitions- und eine Live-Variables-Analyse durchgeführt werden. Erstere ist aus der Vorlesung bekannt. Die Eingabeparameter werden nicht von dieser Analyse erfasst.

Eine Live-Variables-Analyse untersucht, welche Variablen in einer Zeile lebendig sind, d. h. ihr Wert wird später im Programm noch benutzt. Wir gehen ähnlich wie bei der Reaching-Definitions-Analyse vor, betrachten jedoch nur Mengen von Variablen. Im letzten Knoten sind *keine* Variablen lebendig.

Ein Knoten tötet Variablen, denen ein Wert zugewiesen wird und generiert Variablen, die ausgelesen werden (auch in einem `return`-Statement). Variablen, die geschrieben *und* gelesen werden, werden keiner der Mengen hinzugefügt. Die Menge lebendiger Variablen in einem Knoten bestimmt sich aus der Vereinigung lebendiger Variablen aller Nachfolger sowie der lokal generierten Variablen, minus der lokal getöteten Variablen.

(a) Bestimmen Sie die Kill-Menge für eine Reaching Definitions-Analyse für jeden Knoten im Kontrollflussgraphen. 1 P.

(b) Bestimmen Sie ebenfalls die Gen-Menge für eine Reaching Definitions-Analyse für jeden Knoten. 1 P.

(c) Führen Sie die Reaching Definitions-Analyse durch und berechnen Sie die erreichenden Definitionen für jeden Knoten ($RD_{exit}$ nach dem letzten Schritt). 2 P.

(d) Bestimmen Sie die Kill-Menge für eine Live-Variables-Analyse für jeden Knoten im Kontrollflussgraphen. 1 P.

(e) Bestimmen Sie ebenfalls die Gen-Menge für eine Live-Variables-Analyse für jeden Knoten. 1 P.

(f) Führen Sie die Live-Variables-Analyse durch und berechnen Sie die lebendigen Variablen für jeden Knoten. 2 P.