University of Zurich
Department of Informatics

RE
RG

Software Qualität Übung 2

Debugging

1 Informationen

1.1 Daten

- Ausgabe: 06.04.2009
- Abgabe: 20.04.2009 24:00
- Besprechung: 04.05.2009

1.2 Formales

Die Dateien, welche zu Ihrer Abgabe gehören, müssen in eine .zip-Datei gepackt werden (Diagramme und andere Dokumente als PDF, Quellcode als Textdateien in einem separaten Unterverzeichnis). Die Abgabe erfolgt per Email an stoiber@ifi.uzh.ch.

1.3 Gruppen

Die Übung ist in 2er Gruppen zu lösen. Falls die Aufgaben aufgeteilt werden, muss klar ersichtlich sein, wer welchen Teil bearbeitet hat. Alle Gruppenmitglieder müssen über alle Teile Auskunft geben können.

2 Aufgabenstellung

A: Statische Analyse

Die statische Analyse des Programmcodes kann zum Finden von Hypothesen für die Defektlokalisierung verwendet werden. Das folgende Java-Programm soll deshalb auf Daten- und Kontrollabhängigkeiten hin untersucht werden. Beachten Sie dazu auch *Kapitel 11: Statische Analyse* aus der Grundvorlesung Software Engineering.

```
1   public class static_analysis {
2     public static void main(String[] args) {
3       String input = "278 k1lometers; \n6221 24.";
4       System.out.println("Input: "+input);
5       boolean[] digits = new boolean[10];
6       for (boolean x : digits) { x = false; }
7       int nrDiffDigits = 0, nrDigits = 0, sumOfDigits = 0;
8       int nrLines, nrWords = 0, i = 0;
9       if (input.length() >= 0) { nrLines = 1; } else { nrLines = 0; }
10      while ( i < input.length()-1 ) {
11        Character cur = input.charAt(i);
12        if ( cur.isDigit(cur) ) {
13          int curDigit = ((int)cur) - 48; //ascii to decimal
14          nrDigits++;
15          sumOfDigits += curDigit;
16          if ( digits[curDigit] == false ) {
17            nrDiffDigits++;
18            digits[curDigit] = true;
19          }
20        }
21        if ( cur.equals(' ') ) { nrWords++; }
22        if ( cur.equals('\n') ) { nrLines++; }
23        i++;
24      }
25      System.out.println("Nr of digits: "+nrDigits);
26      System.out.println("Nr of different digits: "+nrDiffDigits);
27      System.out.println("Sum of digits: "+nrDiffDigits);
28      System.out.println("Nr of words: "+nrWords);
29      System.out.println("Nr of lines: "+nrLines);
30      System.out.println("These numbers were observed:");
31      for (int j=0; j<digits.length; j++) {System.out.println(j+": "+digits[j]);}
32    }
33  }
```

a) Welche Set-Use, Use-Set, Set-Set Beziehungen gibt es im Programmstück?

b) Definieren Sie einen Program Dependency Graphen (PDG), welcher diese Beziehungen darstellt.

c) Definieren Sie den statischen Backward Slice für Zeile 27 (System.out.println(”Sum of digits: ”+nrDiffDigits);). Suchen Sie dazu in Ihrem Program Dependency Graph den entsprechenden rückgerichteten Pfad und zeichnen Sie diesen ein. Kennzeichnen Sie alle Elemente im Quellcode, die nicht erreicht werden.

d) Wie können Sie Code, der nie aufgerufen wird, mit Hilfe des Program Depedency Graphen erkennen? Welche weiteren möglichen Probleme können mit Hilfe des Program Dependency Graphen erkannt werden?

B: Rekonstruktion von Fehlern

Probleme müssen reproduzierbar sein, damit Fehler gefunden und behoben werden können. Verschiedene Faktoren können einen Einfluss auf diese Reproduzierbarkeit haben. Ein möglicher Einflussfaktor ist Zeit.

Wenn ein Fehler nur zu gewissen Zeitpunkten eintritt, müssen diese Zeitpunkte simuliert werden, um den Fehler rekonstruieren zu können. Die unten stehende “Bug Story” zeigt ein Beispiel, wie die Systemzeit das Auftreten eines Fehlers beeinflusst.

> Bug Story: Program Only Works on Wednesday (Eisenstadt, 1997)
>
> I once had a program that worked properly only on Wednesdays. The documentation claimed the day of the week was returned in a double word (8 bytes). In fact, Wednesday is nine characters long, and the system routine actually expected 12 bytes of space for the day of the week. Because I was supplying only 8 bytes, it was writing 4 bytes on top of the storage area intended for another purpose. As it turned out, that space was where a y was supposed to be stored for comparison with the user’s answer. Six days a week the system would wipe out the y with blanks, but on Wednesdays a y would be stored in its correct place.

Geben Sie zwei weitere mögliche Einflussfaktoren auf die Reproduzierbarkeit von Problemen an. Für jeden dieser Einflussfaktoren geben Sie jeweils ein Beispiel eines möglichen Fehlers, der durch diesen Einflussfaktor beeinflusst wird. Geben Sie dabei auch an, wie der Fehler reproduziert werden kann.

C: Debuggen

Auf der Webseite der Übungen finden Sie die Klasse Quicksort.java, welche die einzelnen Wörter einer eingegebenen Zeichenkette lexikographisch sortiert und ausgibt. Das Programm ist mit einer Version des Quicksort Algorithmus implementiert.

Wird ein Array, das die einzelnen Wörter des folgenden Strings enthält, übergeben, so wird eine Exception geworfen.

> Quicksort (von engl. quick - schnell, to sort - sortieren) ist ein schneller, rekursiver, nichtstabiler Sortieralgorithmus, der nach dem Prinzip ’teile und herrsche’ (lat. Divide et impera!, engl. Divide and conquer) arbeitet. Er wurde ca. 1960 von C. Antony R. Hoare in seiner Grundform entwickelt und seitdem von vielen Forschern verbessert.

Verwenden Sie für diese Aufgabe Eclipse und den Eclipse Debugger.

a) Finden Sie mit einer geeigneten Methode den einfachsten Input. Der einfachste Input ist die kleinste Menge von Strings, welche den Fehler verursacht. Erklären Sie, nach welcher Methode Sie vorgegangen sind, und zeigen Sie, wie ihr Input nach dem ersten, zweiten und dritten Schritt ausgesehen hat.

Beginnen Sie nun mit der Fehlerisolation. Setzen Sie hierzu im Code an geeigneten Stellen Breakpoints (vier Stellen wurden durch *//Debug-Punkt* bereits vorgeschlagen).

b) Überlegen Sie sich für Ihren Input, welche Werte die Variablen *start*, *end*, *untergrenze*, *obergrenze* und *vergleichemit* und das Array *elemente* an den verschiedenen Debug-Punkten bei korrekter Implementierung des Quicksort-Algorithmus haben müssen. Testen Sie dazu die folgenden in diesem Code möglichen Hypothesen:

1. das Inkrementieren der Untergrenze ist falsch (erste for-Schleife)
2. das Dekrementieren der Obergrenze ist falsch (zweite for-Schleife)
3. das Vertauschen der Elemente an der Untergrenze und an der Obergrenze ist falsch
4. die Abbruchbedingung der while-Schleife ist falsch
5. die Rekursion ist falsch

Geben Sie Ihre Testresultate in einer Tabelle an.

c) Benutzen Sie jetzt den Debug-Modus von Eclipse um die Werte der zu beobachtenden Variablen an den Breakpoints zu überwachen. Wenn Abweichungen eintreten, tragen Sie diese in Ihrer Tabelle ein.

d) Bilden Sie dort wo eine Hypothese aus Aufgabe b) zutrifft weitere Hypothesen bis Sie den Fehler genau eingegrenzt haben. Nachdem Sie den Defekt im Programm so lokalisiert haben, geben Sie an, wie der Defekt korrigiert werden kann. Hierzu müssen Sie evtl. die Teilaufgaben b) und c) wiederholen.

e) Analysieren Sie den gegebenen Programmcode auch bezüglich des Programmierstils. Wo können Sie Verbesserungen im Programmierstil vorschlagen? Begründen Sie Ihre Aussagen.