University of Zurich
Department of Informatics

RERG

Software Qualität Übung 2

Debugging

Informationen

Daten

- Ausgabe: 07.04.2008
- Abgabe: 21.04.2008 24:00
- Besprechung: 05.05.2008

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 reinhard@ifi.uzh.ch.

Gruppen

Die Übung ist in 2er (und evtl. einer 3er) 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.

1 Statische Analyse

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

```
1   #define YES 1
2   #define NO 0
3   main()
4   {
5     int c, nl, nw, nc, inword;
6     inword = NO;
7     nl = 0;
8     nw = 0;
9     nc = 0;
10    c = getchar();
11    while ( c != EOF ) {
12       nc = nc + 1;
13       if ( c == '\n' )
14          nl = nl + 1;
15       if ( c == ' ' || c == '\n' || c == '\t' )
16          inword = NO;
17       else if ( inword == NO ) {
18          inword = YES;
19          nw = nw + 1;
20       }
21       c = getchar();
22    }
23    printf("%d \n", nl);
24    printf("%d \n", nw);
25    printf("%d \n", nc);
26 }
```

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 Forward Slice für inword=NO (Zeile 6). Suchen Sie dazu in Ihrem Program Dependency Graph den entsprechenden Pfad und zeichnen Sie diesen ein. Kennzeichnen Sie alle Elemente im Quellcode, die nicht erreicht werden.

d) Definieren Sie den statischen Backward Slice für printf(”%d \n”, nw) (Zeile 24). 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.

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

2 Strukturiertes Erstellen von Hypothesen

Auf der Übungswebseite finden Sie das Java Programm NumberTranslator.jar. Dieses Programm "übersetzt" positive Zahlen in das entsprechende Wort in englischer Sprache. 21 wird demnach übersetzt zu "twenty one". Aus der Konsole starten Sie das Program mit folgendem Kommando (wobei 9 die zu übersetzende Zahl ist):

```
java -cp NumberTranslator.jar NumberTranslator 9
```

Finden Sie heraus, in welchen Situationen das Programm nicht die erwarteten Werte zurückliefert. Gehen Sie dabei nach der Wissenschaftlichen Methode (im Skript Kapitel 5, Seite 22: Hypothesen aufstellen und testen) vor, um eine möglichst korrekte Diagnose zu erstellen, welche Eingaben einen Fehler verursachen und wie sich der Fehler manifiestiert.

Geben Sie die komplette Historie Ihrer Diagnosen als Tabelle ab. Ihre Diagnose soll die Hypothese, Testfälle und entsprechenden Resultate enthalten.

Hinweis: In Eclipse müssen Sie die jar Datei über "Project" → "Properties" → "Java Build Path" → "Libraries" → "Add External JARs" als Bibliothek hinzufügen. Danach können Sie das Programm über "Run" → "Run…" → "Java Application" mit "NumberTranslator" als "Main class" laufen lassen. Die zu übersetzende Zahl können die im Tab "Arguments" angeben.

3 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, muss dieser Zeitpunkt 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.

4 Hypothesen über Programmzustände

Auf der Übungswebsite finden Sie die Klasse Quicksort.java, welche Strings nach einer Version des Quicksort Algorithmus sortieren soll.

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.

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 (4 Stellen wurden durch *//Debug-Punkt* 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 müssten. Geben Sie diese Werte in einer Tabelle an.

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

d) Lokalisieren Sie den Defekt im Programm und geben Sie an, wie der Defekt korrigiert werden kann. Hierzu müssen Sie evtl. die Teilaufgaben b) und c) wiederholen.