Software Qualität Übung 2

Debugging

1. Informationen

Formales

Abgabetermin: **Freitag 18.Mai 2007**, 18.00 CET (Central European Time)
Abgaben per e-Mail an cramer@ifi.unizh.ch und fricker@ifi.unizh.ch
Das abzugebende Material ist in den Teilaufgaben erwähnt. Die Teilaufgabe D soll mit Eclipse gelöst werden.

Bei Fragen: e-Mail an Uniboard 3514 oder cramer@ifi.unizh.ch
Die Computerräume am IFI haben Eclipse installiert.

Literatur: Zeller, Why Programs Fail: A Guide to Systematic Debugging, Dpunkt Verlag, 2005, ISBN 3-89864-279-8.
Das Buch ist bei cramer@ifi.unizh.ch verfügbar (bitte voranmelden).

Der Source Code ist als ZIP-Datei auf der Kurs-Homepage verfügbar.

Struktur der Übung

Ziel der Übung ist, ausgewählte Debugging-Methoden zu beherrschen.

Teil A: Program Slicing
Teil B: Strukturiertes Erstellen von Hypothesen
Teil C: Theoretische Überlegungen zu Rekonstruktion von Fehlern
Teil D: Lokalisierung eines Fehlers mit systematischer Verwendung von Hypothesen über Programmzustände.

Gruppen

Die Aufgaben sollen individuell oder in Gruppen von 2 Personen gelöst werden. Eine Gruppe à 3 Personen kann als Ausnahme bewilligt werden.

2. Aufgabestellung

Teil A

Gegeben sei folgendes C-Programm:

```
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?

Abgabe: Antworten zu den Aufgaben a) bis e).

Teil B

In der Zip-Datei befindet sich eine .jar Datei einer Klasse namens NumberTranslator.

Die Klasse „übersetzt“ positive Zahlen in das entsprechende Wort in englischer Sprache. 21 wird demnach übersetzt zu „twenty one“.

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 23: Hypothesen aufstellen und testen) vor, um eine möglichst korrekte Diagnose zu erstellen, welche Eingaben einen Fehler verursachen und wie sich der Fehler manifiestiert.

Abgabe: komplette Historie Ihrer Diagnosen als Tabelle. Eine solche Diagnose soll mit Hypothese, Testfällen und Resultaten der Tests tabellarisch dargestellt werden.

NB: In Eclipse können Sie der Werte übergeben indem Sie im Menu ***Run*** das Kommando ***Run…*** auswählen und im gegeneben Fenster die Registerkarte Arguments auswählen. Dort könnten Sie bei Program Arguments Werte eingeben. Mehrere Werte müssen durch ein Leerzeichen getrennt werden.

Teil C

Probleme müssen reproduziert werden können, 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.

Abgabe: Antworten zu obiger Frage.

Teil D

In der Zip-Datei befindet sich eine Klasse Quicksort, welche Strings nach einer Version des Algorithmus Quicksort sortieren soll.

Wird ein Array mit den hier gegebenen Strings (unten mit Leerschlag abgetrennt) übergeben, so wird eine Exception geworfen.

```
Quicksort Von engl. Quick – Schnell To Sort – Sortieren Ist Ein
Schneller Rekursiver Nicht-stabiler Sortieralgorithmus Der Nach
Dem Prinzip Teile Und Herrsche lat. Divide,et,impera!, engl.
Divide And Conquer arbeitet. Er Wurde 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 der 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.

Abgabe: Antworten zu den Aufgaben a) bis d), wobei nur die Antworten zu b) und c) geliefert werden sollen, welche zur erfolgreichen Fehleridentifikation geführt haben.