



Auf Fehlersuche mit dem Debugger

Die Fehlersuche **in Programmen** ist eine wichtige Aufgabe **beim Entwickeln** von Software. Durch Debugger lässt sich **diese Aufgabe** einfacher erfüllen.



Kammerjäger

OLIVER MÜLLER

Mal abgesehen von Spielzeugimplementierung, wie "Hallo Welt!", gibt es wohl kein Programm, dass von Anfang an fehlerfrei ist. Der Fehlersuche kommt daher in der Softwareentwicklung eine wichtige Aufgabe zu.

Das Debugging beruht darauf, ein Programm nicht einfach bis zum Fehler und/oder Absturz durchlaufen zu lassen, sondern es an bestimmten Stellen anzuhalten und schrittweise weiter ab-

zuarbeiten. Stellen Sie sich vor, dass in einer Methode, die durch den Klick auf eine Schaltfläche ausgeführt wird, ein Fehler auftritt. Sie können sich diesen Fehler jedoch nicht erklären. Sie starten also den Debugger und laden Ihr Programm.

Jetzt teilen Sie dem Debugger mit, dass er beim Ausführen des Programms beim Eintritt in die Methode anhalten soll. Sowie Sie also beim Ausführen des Programms auf die berüchtigte Schaltfläche klicken und das Programm in die Methode eintritt, soll der Debugger anhalten. Dies erreichen Sie durch einen *Breakpoint* oder *Haltepunkt*, den Sie am Anfang der Methode setzen.

Ist das Programm durch den Debugger angehalten, haben Sie die Möglichkeit die Inhalte von Variablen und Objekte unter die Lupe zu nehmen. Sie können also feststellen, ob irgendeine Abfrage wider Erwarten nicht das Resultat liefert, das Sie erwartet hatten.

Den Programmablauf selbst können Sie jetzt schrittweise - also Anweisung für Anweisung - abarbeiten. Sie sehen jetzt, wie sich die Inhalte der Variablen und Objekte ändern und wie sich der Programmlauf bei *if* und *switch* verzweigt.

■ Vorarbeiten

Bevor Sie sich auf die Suche nach Fehlern in Ihrem Programm machen können, müssen Sie diese beim Compilieren/Linken entsprechend vorbereiten. Eine ausführbare Datei enthält nur noch Maschinencode. Es befinden sich

also nur noch binäre Befehle für die CPU darin. Den ursprünglichen Quelltext können Sie in diesem Wirrwarr von Einsen und Nullen nicht mehr erkennen.

Durchspezielle Optionen beim Compilieren können Sie aber in der ausführbaren Datei "Debugging-Informationen" ablegen. Diese Informationen ordnen den Prozessorinstruktionen die Zeilen und Bezeichner des Quelltextes zu. Damit kann der Debugger dem Programmierer den Quelltext zeigen und diesen scheinbar Schritt für Schritt abarbeiten. In Wahrheit arbeitet der Debugger aber über dem Maschinencode des ausführbaren Programms. Durch die Zuordnung zwischen den binären Instruktionen und den Zeilen des Source-Codes kann er jedoch dem Programmierer die Illusion vermitteln, dass er den Quelltext debuggt.

Damit der Compiler die Debugging-Informationen in das Executable aufnehmen und die Fehlersuche ermöglichen, müssen Sie dies diesem explizit mitteilen. Für den GNU C++-Compiler geschieht dies über die Option *-g*.

Wenn Sie beispielsweise den Quelltext *foo.cc* compilieren, verwenden Sie zum Compilieren diesen Befehl:

```
c++ -g -c foo.cc
```

Die Methoden und Objekte, die Sie in *foo.cc* definiert haben, können Sie danach später im Debugger untersuchen.



Damit Sie das ganze Programm debuggen können, müssen Sie *-g* beim Compilieren jedes Quelltextes angeben.

QUICK INDEX

- ▶ **Wie funktioniert Debugging?**
Debugging beruht darauf Programme anzuhalten und schrittweise weiter auszuführen.
- ▶ **Vorarbeiten**
Bevor Sie debuggen können, müssen Sie Ihr Programm vorbereiten.
- ▶ **Auf in den Kampf**
DDD bietet Ihnen eine ansprechende Oberfläche, über die Sie alle Debugging-Funktionen per Mausklick erreichen.
- ▶ **Stoppl**
Über Breakpoints können Sie die Ausführung von Programmen gezielt anhalten.
- ▶ **Step by Step**
Die Ausführung können Sie Schritt für Schritt verfolgen.
- ▶ **Was steht denn da?**
Um den Status eines Programms zu beurteilen, müssen Sie die Variablen untersuchen.



Auf in den Kampf

Unter Linux bietet sich der "Data Display Debugger" (DDD) für die Fehlersuche an. Dieser Debugger ist - im Gegensatz zu seinen spröden Unix-Pendants - in eine ansprechende grafische Oberfläche gekleidet. Wie Sie dieses Programm installieren, lesen Sie im Kasten "DDD installieren". Das Starten von DDD erfolgt über das Kommando

```
ddd
```

aus dem Shell-Fenster.

Nachdem DDD sich auf dem Bildschirm präsentiert, können Sie das Programm, das Sie debuggen möchten, über File / Open laden. In dem folgenden Dialogfeld wählen Sie das ausführbare Programm aus. Nach dem Laden zeigt sich der Quelltext des Programms. Besteht Ihre Software aus mehreren Modulen (Quelltexten), so wird der Quelltext angezeigt, der die main()-Funktion enthält.

Wenn Sie den Fehler in einem anderen Modul vermuten, öffnen Sie den betreffenden Quelltext durch File / Open Source. Damit können Sie zwischen den einzelnen Quelltexten wechseln.

Unter dem Quelltext sehen Sie die Konsole des GNU Debuggers gdb. DDD ist "nur" ein grafisches Frontend für den gdb. Es verwendet also für die eigentlichen Debugging-Funktionen dieses textbasierten GNU-Tool. Über diesen Fensterabschnitt unter dem Quelltext können Sie gdb-Kommandos eingeben, um direkt die gdb-Features zu nutzen. In der Regel ist dies aber nicht notwendig, da Sie die betreffenden Funktionen über die grafische Oberfläche von DDD erreichen.

Das kleine Fenster mit den hervorstechenden Schaltflächen *Run* und *Interrupt* ist das Kontrollzentrum von DDD. Über dieses Werkzeugfenster erreichen Sie die Debugging-Funktionen über einen simplen Mausklick. Sie können damit beispielsweise Breakpoints setzen oder das Programm schrittweise abarbeiten.

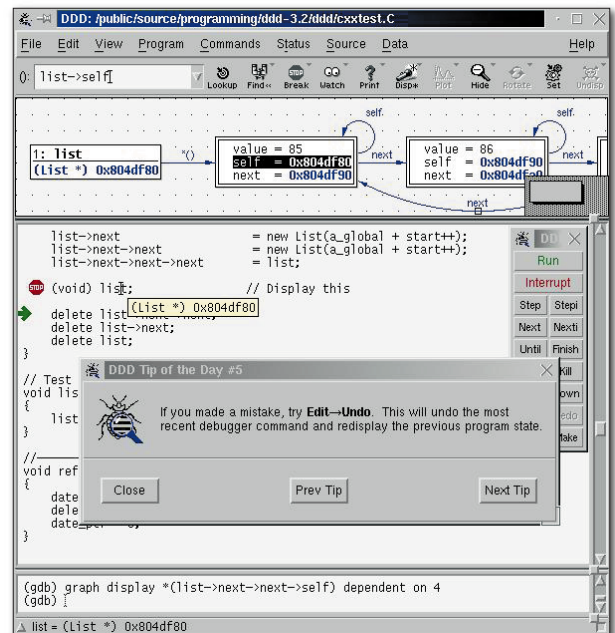
Stopp!

Zum Anhalten des Programms an bestimmten Stellen, benötigen Sie Haltepunkte. Ein solcher Breakpoint markiert eine Anweisung bzw. Zeile im Quelltext, an deren Stelle der Debugger anhalten soll, wenn der Programmlauf dort angenommen ist.

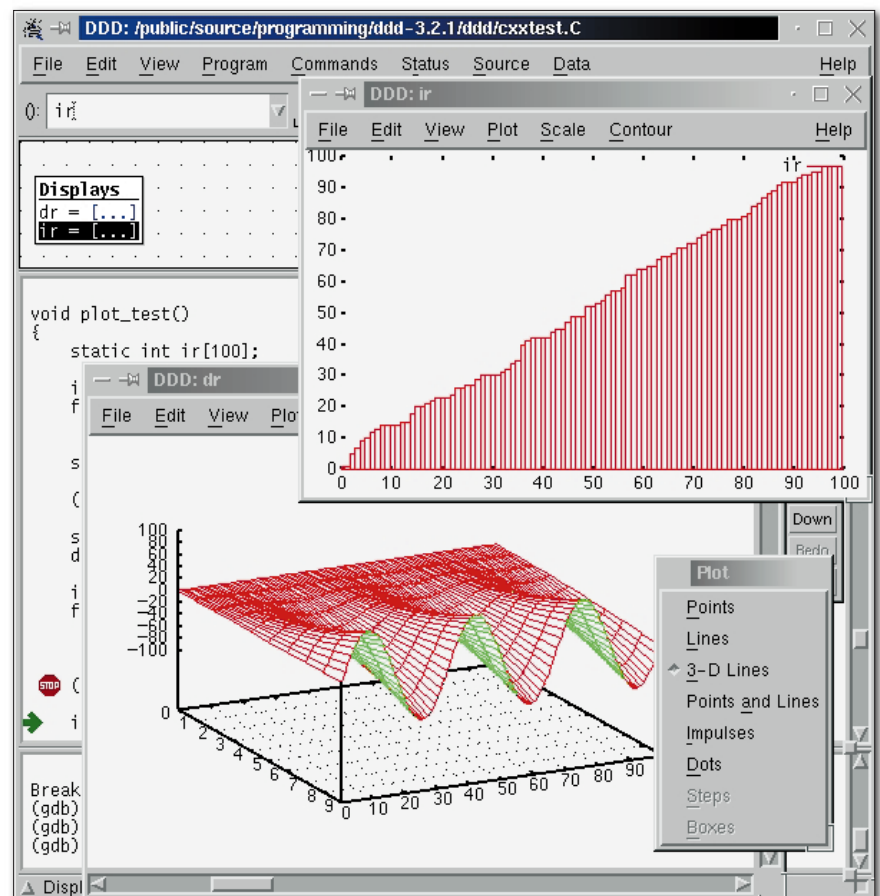
Um einen Breakpoint zu setzen, gehen Sie mit dem Cursor in die erste Spalte der Zeile, ab der Sie den Programmlauf unterbrechen wollen. Anschließend klicken Sie auf die Schaltfläche *Break* des Werkzeugfensters. Alternativ können Sie auch den Eintrag *Set Breakpoint* des Kontextmenüs, das Sie über einen Klick mit der rechten Maustaste im Quelltextfenster erhalten, verwenden. Ein Stop-Schild vor der Zeile zeigt an, dass hier ein Haltepunkt gesetzt wurde.

Nachdem Sie den Breakpoint gesetzt haben, können Sie das Programm starten. Klicken Sie hierzu auf die Schaltfläche *Run* des Werkzeugfensters. Das Programm wird jetzt solange ausgeführt, bis die Zeile mit dem Haltepunkt er-

reicht wird. In diesem Fall stoppt der Debugger die Ausführung, und Sie können mit der Inspektion Ihres Programms beginnen, sei es Schritt für Schritt oder Break für Break.



DDD IN AKTION, eine Wanze ist zu sehen.



GRAFISCHE AUSWERTUNGEN erleichtern die Fehlersuche.



Step by Step

Ist Ihr Programm angehalten, hat es recht wenig Sinn danach auf Run zu klicken. Sie würden damit nur wieder das Programm von neuem starten. Sie können jetzt über die Schaltflächen Next springt immer zur nächsten Anweisung im gerade aktuellen Unterprogramm. Selbst wenn diese Anweisung eine andere Methode aufruft, verzweigt das Programm nicht in die betreffende aufgerufene Methode.

Die Schaltfläche *Step* hat zunächst die selbe Aufgabe wie *Next*. Auch Sie setzt die Ausführung schrittweise fort. Gelingt die Ausführung jedoch an einen Methodenaufruf, verzweigt *Step* in diese Methode. Sie finden sich also bei der Anweisung der aufgerufenen Methode wieder. Wurde die Methode, in die Sie gerade verzweigt haben durchlaufen, landen Sie wieder beim ursprünglichen Aufruf.

Wollen Sie den Programmlauf wieder in einem Fluss - also nicht schrittweise - fortsetzen, klicken Sie auf *Cont*. Danach wird die Ausführung wieder "normal" aufgenommen, bis der nächste Breakpoint erreicht wurde.

Was steht denn da?

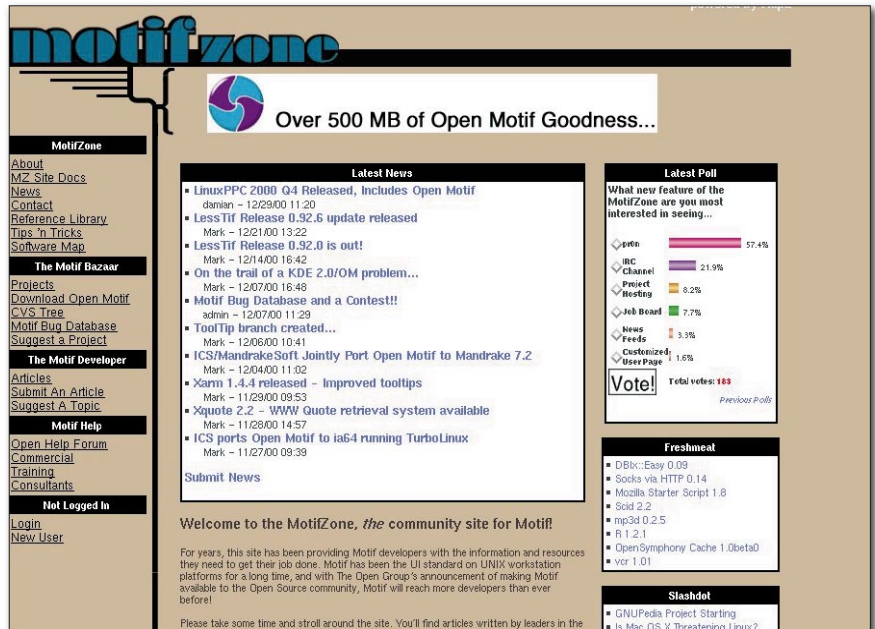
Damit Sie den Zustand eines Programms beurteilen können, müssen Sie die Daten untersuchen. Konkret heisst das, dass Sie den Inhalt der Variablen ansehen müssen.

Um eine Variable zu untersuchen, klicken Sie auf eine Variable im Quelltext und verwenden die Schaltfläche *Display*. Über dem Quelltext erscheint nun ein zusätzlicher Abschnitt, der Ihnen die Variable als Grafik veranschaulicht.

DDD ist bei der Datenvisualisierung sehr leistungsfähig. Angenommen Sie haben eine Zeigervariable *ptr*, die auf einen Datenbereich *data* zeigt. Wenn Sie beide Variablen per *Display* anzeigen, zeigt Ihnen DDD auch den Zusammenhang dieser beiden Variablen. Von *ptr* geht ein Pfeil aus, der auf *data* zeigt.

Über die rechte Maustaste können Sie zu jeder angezeigten Variablen ein Kontextmenü aufrufen. Mit den darin enthaltenen Befehlen können Sie die Ansicht der Variablen beeinflussen.

Der Eintrag *Undisplay* des Kontextmenüs lässt die dargestellte Variable wieder verschwinden.



FÜR DDD BENÖTIGEN Sie Motif von www.openmotif.org.

DDD INSTALLIEREN

Auf der Heft-CD finden Sie DDD im gleichnamigen Verzeichnis. Darin sind zwei Unterverzeichnisse *RedHat* und *tarball*. Das erste Verzeichnis enthält ein Binary- und ein Source-RPM für Red Hat Linux. Unter Red Hat installieren Sie lediglich *ddd-3.2.93-1.i386.rpm* mit *kpackage*, *gnorpm* oder *rpm*. Sie müssen allerdings Open Motif installieren, um dieses RPM verwenden zu können. Für alle anderen Linux-Distributionen finden Sie einen Tarball (.tar.gz-Archiv) im anderen Verzeichnis. Um dieses zu installieren, benötigen Sie entweder LessTif Version 0.89 oder später, oder (besser) OSF/Motif Version 1.2 oder später.



TIP OSF/Motif erhalten Sie als "Open Motif" unter <http://www.openmotif.org>. Dort können Sie RPMs, DEBs und Tarballs für Ihre Distribution herunterladen und installieren.

Um DDD aus dem Tarball zu installieren, benötigen Sie die Development-Packages LessTif oder Open Motif. Laden Sie sich deshalb aus dem Internet die Motif-Laufzeitumgebung und das Entwicklungspaket herunter. Loggen Sie sich zuerst als "root" ein. Wenn Sie unter X11 arbeiten, öffnen Sie ein Shell-Fenster (häufig auch Terminal genannt). Legen Sie die CD ins Laufwerk ein und mounten Sie sie durch den Befehl

```
mount /mnt/cdrom
```

bzw. unter SuSE durch

```
mount /cdrom
```

Anschließend wechseln Sie in das HOME-Verzeichnis:

```
cd
```

Jetzt entpacken Sie den Tarball:

```
tar xzf /mnt/cdrom/DDD/tarball/ddd-3.2.1.tar.gz
```

bzw. unter SuSE:

```
tar xzf /cdrom/DDD/tarball/ddd-3.2.1.tar.gz
```

Danach können Sie die CD-ROM wieder unmounten und aus dem Laufwerk nehmen:

```
umount /mnt/cdrom
```

bzw. unter SuSE:

```
umount /cdrom
```

Zunächst wechseln Sie in das neu entstandene Verzeichnis:

```
cd ddd-3.2.1
```

Hier angekommen, starten Sie den Build durch folgende Kommandos:

```
./configure  
make
```

Nachdem der Build ohne Fehlermeldungen durchgelaufen ist, installieren Sie DDD durch dieses Kommando:

```
make install
```