Ausarbeitung: Die Kunst des Debuggens

Allgemeines Seminar

Blockseminar Informatik
01.03.2005 – 03.03.2005

Katja Reitz

Inhaltsverzeichnis

1 Debuggen

Definitionen: *„Fehler in einem Programm oder einer Software aufspüren und entfernen.“* [2]

> *„Bei dem EDV Begriff Debuggen bzw. Debugging handelt es sich um einen Vorgang der Fehlersuche und der Fehlerbeseitigung in einem Programm oder Quellcode. Diese Fehlersuche und Fehlerbeseitigung kann manuell oder mit Hilfe eines Anwendungsprogramm, dem sogenannten Debugger vollzogen werden. Die wichtigsten Hilfsmittel für das Debuggen bzw. Debugging unter Zuhilfenahme eines Debuggers sind die sogenannten Breakpoints (wörtlich: Haltepunkte, Programmpausierungen), Watches (wörtlich: Überwachungen) und das Tracing (wörtlich: Verfolgung).“* [3]

Debugging heißt übersetzt ‚Entwanzung’. In der Informatik bedeutet Debuggen, das Fehlersuchen und Fehlerbeheben bei Softwareprogrammen.

Debuggen ist eine bei Programmierern eher ungeliebte Aufgabe. Gründe dafür sind einerseits das Eingestehen von Fehlern seitens des Programmierers. Zum zweiten bedeutet es ein großer Zeitaufwand und eine gewisse mentale Anstrengung. Des weiteren muss man beim Debuggen zumeist die Fehler alleine finden, im Gegensatz zu vorhergegangenen Entwicklungsphasen, in denen man im Team arbeitet.

Man beginnt mit dem Debuggen, wenn ein Test Case erfolgreich war, das heißt es sind ein oder mehrere Fehler aufgetreten. Hierbei ist zu beachten, dass das Debuggen nicht mit dem Testen von Software gleichgesetzt werden kann. Beim Testen wird die komplette Software auf Fehler geprüft. Erst wenn Fehler aufgetreten sind, fängt man an diesen Fehler zu debuggen.

Das Debuggen unterteilt man in zwei Stufen:

1. Die Ermittlung der Art des Fehlers und an welcher Stelle im Programm er aufgetreten ist.
2. Anschließend muss der Fehler behoben werden.

Der meiste Inhalt dieser Ausarbeitung befasst sich mit der Stufe 1 des Debuggens, da 95% des Aufwandes des Debuggens diesem Punkt zufällt. Außerdem hält sich die Ausarbeitung weitgehend an das Buch „The Art of Software Testing“ von J. G. Myers [1].

Im folgenden werden einige Methoden des Debuggens vorgestellt und anschließend einige allgemeine Techniken, die man beim Debuggen beachten sollte. Zum Schluss wird noch auf die Fehleranalyse nach Abschluss eines Projektes eingegangen.

2 Methoden des Debuggens

2.1 Methode: ‚Brute Force’

‚Brute Force’ heißt übersetzt ‚rohe Gewalt’. Damit wird die Herangehensweise an das Debuggen dieser Art beschrieben. Diese ‚Brute Force’-Methode ist die übliche Vorgehensweise des Debuggens, die auch am meisten geläufig ist. Charakteristisch für diese Methode ist, dass wenig mentale Anstrengung notwendig ist. Allerdings ist sie auch sehr ineffizient und oft nicht sehr erfolgreich.

Hier werden drei Kategorien von ‚Brute Force’-Methoden unterschieden:

- Debuggen mit einem Speicher Dump
- Print-Anweisungen werden an diversen Stellen im Programm platziert.
- Automatische Debugging-Werkzeuge

Debuggen mit einem Speicher Dump

Mit einem Speicher Dump wird der Speicherinhalt an einer bestimmten Stelle im Programm oder zu einem bestimmten Zeitpunkt abgebildet. Diese Methode ist am meisten ineffizient. Dies hat mehrer Gründe. Zum einen ist es schwierig eine Verbindung zwischen Speicherort und Variable im Quellprogramm herzustellen. Des weiteren entsteht dadurch bei großen Programmen eine riesige Menge an Daten, von denen die meisten nicht gebraucht werden. Außerdem ist der Speicher Dump nur eine statische Abbildung des Programms. Um aber Fehler zu finden, sollte man die Dynamik eines Programms studieren. Dazu kommt noch, dass der Speicher Dump meistens nicht exakt an die Stelle des Fehlers platziert wird, so dass nicht der Programmstatus zum Zeitpunkt des Fehlers gezeigt wird. Es kann sogar passieren, dass Hinweise auf den Fehler verdeckt werden.

Print-Anweisungen werden an bestimmten Stellen im Programm platziert

Diese Methode ist etwas besser als die Methode mit den Speicher Dumps, da sie besser die Dynamik des Programms zeigt.
Ein Nachteil ist zum Beispiel, dass man zum Probieren verleitet wird, anstatt zum Denken, indem man an beliebigen Stellen eines Programms Print-Anweisungen streut. Des weiteren erzeugt auch diese Methode eine große Datenmenge, die analysiert werden muss. Ein weiteres Problem ist, dass durch diese Methode das Programm geändert werden muss. Dies kann zum einen Fehler überdecken, zum anderen auch weitere Fehler einleiten.
Diese Methode funktioniert noch ganz gut bei kleineren Programmen, bei größeren Programmen dagegen sind die Kosten zu groß. Außerdem ist diese Art des Debuggens bei bestimmten Programmtypen nicht möglich, wie zum Beispiel bei Betriebssystemen.

Automatische Debugging-Werkzeuge

Das Verwenden von Debugging-Werkzeuge ist vergleichbar mit dem platzieren von Print-Anweisungen, nur wird hier das Programm nicht verändert. Eine übliche Methode ist hier das setzten von Haltepunkten an Stellen, an denen man Fehler vermutet oder das verwenden von Traces. Auch hier entsteht wie bei den anderen beiden Methoden eine große Menge an irrelevanten Daten. Auch hier ist es eher ein Zufall einen Fehler zu finden.

Generell kann man zu allen drei Methoden sagen, dass sie eine große Menge an Daten produzieren, die analysiert werden muss und die zu meist nicht zur Fehlersuche beiträgt. Des weiteren ist zu bemängeln, dass man nicht dazu ermutigt wird zu Denken, um eine Lösung zu finden, sondern das Ausprobieren wird gefördert. Oft kann es sogar passieren, dass Fehler überdeckt werden oder sogar neue Fehler eingeleitet werden.

2.2 Methode: Induktion

Induktion: *„Schlussfolgerung vom Besonderen auf das Allgemeine.“* [2]

Die meisten Fehler werden durch Denken gefunden, ohne an den Computer zu gehen. Die Methode der Induktion fordert solch einen Denkprozess. Der Prozess sieht vor vom speziellen zum generellen zu gehen. Es werden Hinweise (Symptome von Fehlern, Resultate von Test Cases) gesammelt und wenn möglich Beziehungen zwischen ihnen hergestellt. Die folgende Grafik stellt die Methode der Induktion da:

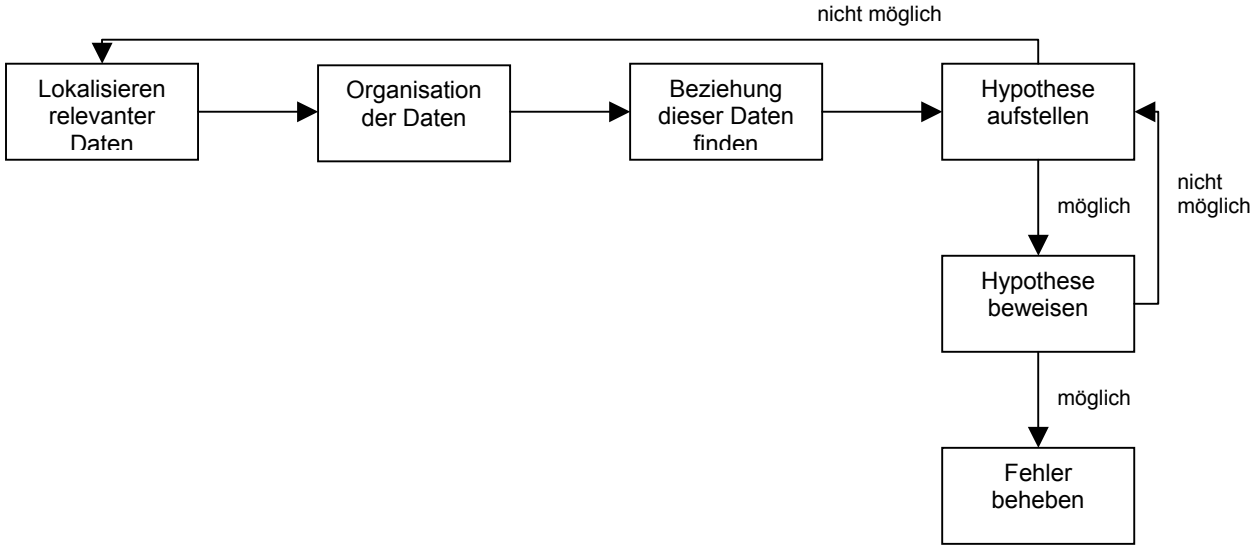

Abbildung 1: Methode Induktion [1]

Die Induktion läuft in folgenden Schritten ab:

1. Lokalisieren der relevanten Daten: Hier sollen die Symptome der Fehler aufgezählt werden, was korrekt funktioniert und was nicht korrekt funktioniert.
2. Organisation der Daten: In diesem Schritt sollen die Daten strukturiert werden. Es sollen Muster und Widersprüche erkannt werden. Dazu kann das folgende Formular hilfreich sein.

	korrekt	nicht korrekt
Was?		
Wo?		
Wann?		
In welchem Umfang?		

Abbildung 2: Formular zur Datenstrukturierung [1]

In den zwei Spalten wird notiert, was korrekt läuft und was nicht korrekt läuft. In die Zeile ‚Was?' werden die generellen Symptome eingetragen. Danach gibt man an, an welcher Stelle die Symptome aufgetreten sind (Zeile ‚Wo?'). Unter ‚Wann?' wird notiert, zu welchem Zeitpunkt die Symptome aufgetreten sind. Zu letzt wird unter ‚Umfang?' aufgelistet, welches Ausmaß die Symptome haben.

3. Hypothese aufstellen: Jetzt wird versucht Beziehungen zwischen den Hinweisen herzustellen. Kann keine Hypothese aufgestellt werden, dann müssen neue Daten gesammelt werden. Ergeben sich mehrere Theorien, wird die wahrscheinlichste zu erst geprüft.
4. Beweisen der Theorie: Hier wird die Hypothese mit den ursprünglichen Daten verglichen und auf Korrektheit überprüft. Oft wird der Fehler gemacht, diesen Schritt zu überspringen und gleich mit der Fehlerkorrektur zu beginnen. Dies hat zu Folge, dass meist nur das Symptom des Fehlers korrigiert wird, aber nicht der Fehler selbst.
5. Ist die Hypothese bewiesen, kann mit der Fehlerbehebung begonnen werden. Könnte die Hypothese dagegen nicht bewiesen werden, muss einen neue Hypothese aufgestellt werden.

2.3 Methode: Deduktion

Deduktion: *„Ableitung des Besonderen aus dem Allgemeinen.“* [2]

Der Prozess der Deduktion sieht vor von einer generellen Theorie über Ausschließen und Verfeinern zur Lokalisierung des Fehlers zu kommen. Die nachfolgende Abbildung verdeutlicht den Prozess der Deduktion:

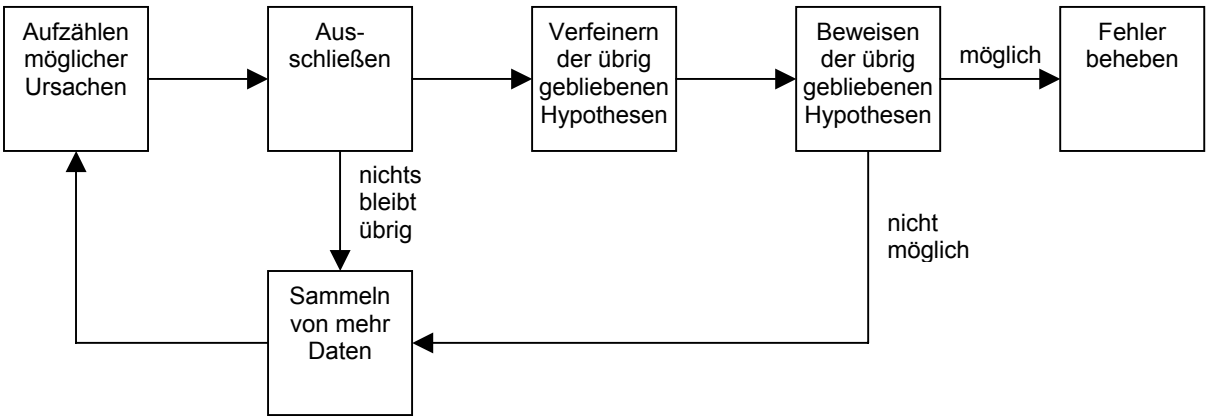

Abbildung 3: Methode Deduktion [1]

Die Deduktion läuft folgendermaßen ab:

1. Aufzählen von möglichen Ursachen und Hypothesen: Es soll ein Liste mit denkbaren Ursachen für den Fehler erstellt werden. Dies soll keine vollständige Erläuterung der einzelnen Fehler sein, sonder nur eine kurze Auflistung.
2. Ausschließen von möglichen Ursachen: Hier sollen die Daten zum Beispiel nach Widersprüchen untersucht werden. Ursachen die ausgeschlossen werden können, werden von der Liste gestrichen. Bleiben keine Ursachen übrig, müssen neue Theorien aufgestellt werden. Bleiben mehrere Ursachen übrig, wird die wahrscheinlichste als erstes geprüft.
3. Verfeinerung der übrig gebliebenen Hypothesen: Dies ist nötig um den Fehler genauer zu spezifizieren. Hierzu kann das Formular aus Abbildung 2 der Induktion genommen werden.

4. Beweisen der übrig gebliebenen Hypothesen: Wie auch bei der Induktion müssen hier die Hypothesen auf Korrektheit überprüft werden, bevor man versucht den Fehler zu beheben.
5. Kann eine Hypothese bewiesen werden, kann der Fehler behoben werden, wenn keine Hypothese bewiesen werden kann, müssen auch hier neue Daten gesammelt und analysiert werden.

2.4 Methode: Backtracking

Beim Backtracking startet man die Suche nach dem Fehler an der Stelle, wo sich die letzte richtige Anweisung befindet und die darauffolgende ein falsches Resultat liefert. Anschließend wird der falsche Wert mit dem Wert verglichen, der an dieser Stelle eigentlich richtig wäre, um rückverfolgend die Ursache für den Fehler zu finden.
Diese Methode eignet sich allerdings nur für kleine Programme. Da ist sie aber sehr effektiv.

2.5 Methode: Testen

Hier werden Test Cases verwendet. Bei Test Cases wird zwischen Test Cases zum Debugging und Test Cases zum Testen unterschieden. Hier ist die Rede von Test Cases zum Debuggen. Da diese Test Cases sehr klein sind, eignen sie sich für das Debuggen. Das Schreiben von Test Cases ist keine eigenständige Methode des Debuggens, sie verbindet mehrere Ansätze. Zum Beispiel kann zur Charakterisierung des Fehlers erst die Methode der Induktion oder der Deduktion angewendet werden, um dann einen entsprechenden Test Case für den aufgetretenen Fehler zu schreiben.

3 Prinzipen beim Debuggen

Es gibt einige allgemeine Prinzipien beim Debuggen, die teilweise psychologischer Natur sind. Dies sind zu meist Dinge, die jeder kennt, die aber oft vergessen werden. Da das Debuggen aus zwei Phasen besteht, das Fehler lokalisieren und das Fehler beheben, werden die Prinzipen auch in diese beiden Bereiche aufgeteilt.

Prinzipen bei der Fehlerlokalisierung

- Denken:
  Es ist erwiesen, dass sich die meisten Fehler beim Programmieren durch Denken beheben lassen, ohne auch nur an den Computer zu gehen.
- Kommt man bei einem Fehler nicht weiter, sollte man darüber schlafen:
  Wenn man nach einer gewissen Zeit der Fehlersuche (bei kleinen Programmen ca. 30 Minuten; bei größeren mehrere Stunden) keine Lösung des Problems gefunden hat, dann sollte man das Thema für eine Weile vergessen und sich mit etwas anderem beschäftigen. Oft löst unser Unterbewusstsein das Problem in dieser Zeit.
- Kommt man bei einem Fehler nicht weiter, sollte man das Problem jemanden anderen beschreiben:
  Dies hat meist den Effekt, dass man durch das reine Erklären des Problems selbst die Lösung findet.
- Debugging-Werkzeuge sollten nur als zweite Möglichkeit verwendet werden:
  Debugging-Werkzeuge sollten erst verwendet werden, wenn andere Methoden zu keinem Erfolg geführt haben und dann auch nur als Zusatzwerkzeug. Auch hier sollte man nicht vergessen über das Problem nachzudenken. Experimente zeigen, dass Programmierer, die Debugging-Programme meiden, bei der Fehlersuche erfolgreicher sind als die, die diese Werkzeuge benutzen.
- Experimente sollen vermieden werden oder nur als letzte Möglichkeit verwendet werden:
  Eine experimentelle Fehlersuche hat eine geringe Chance auf Erfolg. Außerdem werden dadurch oft noch weitere neue Fehler in das Programm eingefügt.

Techniken zur Fehlerbehebung

- Wo ein Fehler ist, sind oft noch weitere:
  Die Praxis zeigt, dass sich in der Umgebung eines Fehlers oft noch weitere Fehler befinden. Fehler treten oft im Verbund auf.
- Den Fehler beheben und nicht nur das Symptom von ihm:
  Es kommt oft vor, dass nur das Symptom des Fehlers oder nur eine Instanz des Fehlers korrigiert werden, anstatt des Fehlers selbst.
- Die Wahrscheinlichkeit, dass die Fehlerkorrektur richtig ist, ist nicht 100%:
  Die Korrektur eines Fehlers muss genauso getestet werden, wie das ursprüngliche Programm, vielleicht sogar noch mehr.
- Die Wahrscheinlichkeit, dass die Fehlerbehebung korrekt ist, nimmt ab, je größer das Programm ist.
- Es kann vorkommen, dass eine Fehlerkorrektur neue Fehler erzeugt, zum Beispiel Seiteneffekte, die neue Fehler erzeugen.
- Der Prozess zur Fehlerbehebung bringt einem zeitweilig in die Designphase zurück:
  Fehlerkorrektur ist eine Form von Programmdesign. Es werden ähnliche Prozeduren, Methoden und Formalismen verwendet.
- Die Änderungen müssen im Quellcode erfolgen, nicht im Objektcode.

4 Fehleranalyse

Die Analyse nach Abschluss eines Projektes dient den nächsten Projekten. Durch eine Fehleranalyse können weitere Projekte effektiver und weniger fehlerreich durchgeführt werden. Folgende Fragestellungen müssen analysiert werden:

- Wo wurde der Fehler gemacht?
  Wurde ein Fehler zum Beispiel bei der Korrektur eines früheren Fehlers gemacht oder gab es mehrdeutige Angaben in der Spezifikation oder gab es Missverständnisse bei den Anforderungen des Endanwenders. Diese Fragestellung ist sehr umfangreich und nur mit einem gewissen Aufwand zu beantworten. Hierzu muss man unter anderem noch einmal die Geschichte des Projektes studieren, sprich Dokumentationen etc.
- Wer hat den Fehler gemacht?
  Hier kann zum Beispiel festgestellt werden, dass 60% der Fehler von einer Person gemacht wurden oder dass ein Programmierer dreimal so viele Fehler macht wie die anderen. Dies dient nicht dazu einen Schuldigen zu finden, sondern um es das nächste Mal besser zu machen.
- Was wurde falsch gemacht?
  Dies können zum Beispiel Tippfehler, Defizite bei der Programmiersprache oder falsche Annahmen sein.
- Wie können solche Fehler vermieden werden?
  Hier geht es darum, was beim nächsten Projekt besser gemacht werden kann.
- Warum wurden Fehler nicht früher entdeckt?
- Wie könnte der Fehler früher entdeckt werden?

5 Fazit

Es ist vor allem festzuhalten, dass das Debuggen in erster Linie ein Denkprozess ist, indem Debugging-Werkzeuge nur als Ergänzung dienen. In dem Buch von J. G. Myers [1] wird immer wieder darauf hingewesen, das die meisten Fehler ohne Hilfe des Computers gelöst werden können, durch Denken. Des weiteren wird immer wieder gesagt, dass Debuggen mit Hilfe von Werkzeugen oder durch Programmausgaben das Denken nicht fördert, sondern zum Ausprobieren verleitet. Das Problem bei Debugging-Werkzeugen liegt vor allem in der großen Datenmenge, die produziert wird und die zum größten Teil irrelevant ist. Gerade bei größeren Projekten ist dies kaum noch analysierbar. Diese Werkzeuge können aber eingesetzt werden, wenn man den Fehler schon lokalisiert und charakterisiert hat, um dann den Fehler gezielt zu beheben.
Weiterhin ist zu beachten, dass man sich beim Debuggen generell an die richtige Reihenfolge hält. Das heißt man bestimmt erst Art des Fehlers und die Stelle an die der Fehler auftritt und erst dann behebt man den Fehler. Dies ist eins der größten Probleme beim Debuggen, dass man versucht den Fehler zu beheben, ohne wirklich zu wissen, um was es sich eigentlich handelt. Dadurch kann zum Beispiel nur ein Symptom des Fehlers behoben worden sein oder der Fehler wurde sogar überdeckt.

6 Quellenverzeichnis

[1] Myers, G. J., The Art of Software Testing, John Wiley, 2004
[2] Wörterbuch der deutschen Sprache, Bertelsmann, 2004
[3] http://www.bullhost.de/d/debuggen.html