Flussdiagramme vs. Pseudocode

Ausarbeitung zum Vortrag im Seminar
"Empirische Forschungsmethoden in der Softwaretechnik"
WiSe 2004

© 2004 Marc 'BlackJack' Rintsch
marc@rintsch.de

7. April 2004

Inhaltsverzeichnis

1 Worum geht es?

Die beiden Papiere, welche diese Ausarbeitung zum Gegenstand hat, beschäftigen sich mit der Frage, welche der beiden Darstellungsarten für Algorithmen – Flussdiagramme oder (Pseudo)code – besser geeignet sind, um Algorithmen zu verstehen.

Einleitend folgen zwei Abschnitte über Flussdiagramme und Pseudocode. Ausserdem eine ganz kleine Übersicht über FORTRAN und eine kurze Erklärung des *Grade Point Average* – kurz *GPA*.

1.1 Flussdiagramme

Flussdiagramme (Flowcharts, Programmablaufpläne) sind eine grafische Notation zur Darstellung von Programmen bzw. Teilprogrammen. Der Informatikduden [InfDuden93] sagt dazu auf Seite 542:

> **Programmablaufplan** (*Ablaufplan*, *Ablaufdiagramm*, *Flussdiagramm*): Normierte Methode zur graphischen Darstellung von Programmen. [. . .]

Die Anweisungen oder ganze Anweisungsblöcke werden als Symbole dargestellt, die durch Linien verbunden sind, welche die möglichen Programmabläufe darstellen. In Abbildung 1 sind die wichtigsten Symbole aufgelistet.

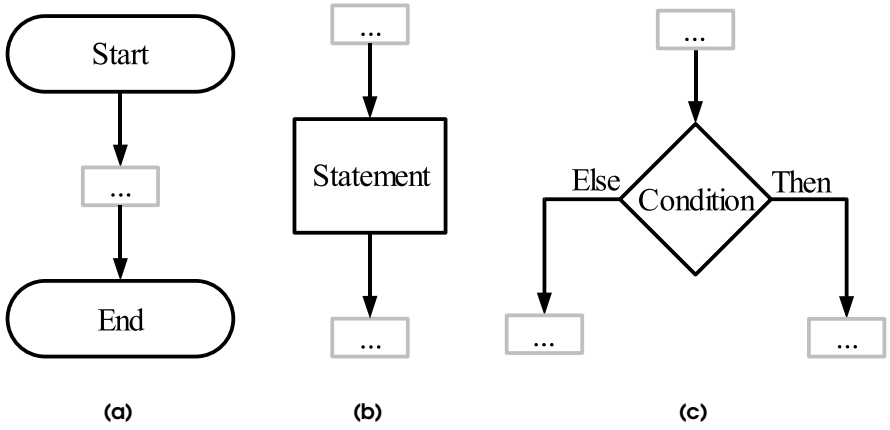

Abbildung 1: Die grundlegenden Symbole in einem Flussdiagramm. (a) Start und Ende des Algorithmus. (b) Einfache Anweisung(en). (c) Bedingte Verzweigung.

In Flussdiagrammen müssen Schleifen-Konstrukte wie `for`- oder `while`-Schleifen durch einzelne Aktionen und Bedingungen nachgebildet werden. Ferner gibt es keine Möglichkeit rekursive Aufrufe darzustellen.

Die Ablaufbeschreibung ist sehr "maschinennah" und wird deshalb auch gerne für die Beschreibung von kleinen Algorithmen auf Maschinensprache- oder sogar Hardwareebene benutzt. Im "Patterson/Hennessy" [PatHen98, S. 208 ff] finden sich in Kapitel 4 zum Beispiel die mathematischen Grundoperationen Addition, Subtraktion, Multiplikation und Division für Integer- und Fliesskommazahlen jeweils als Flussdiagramm

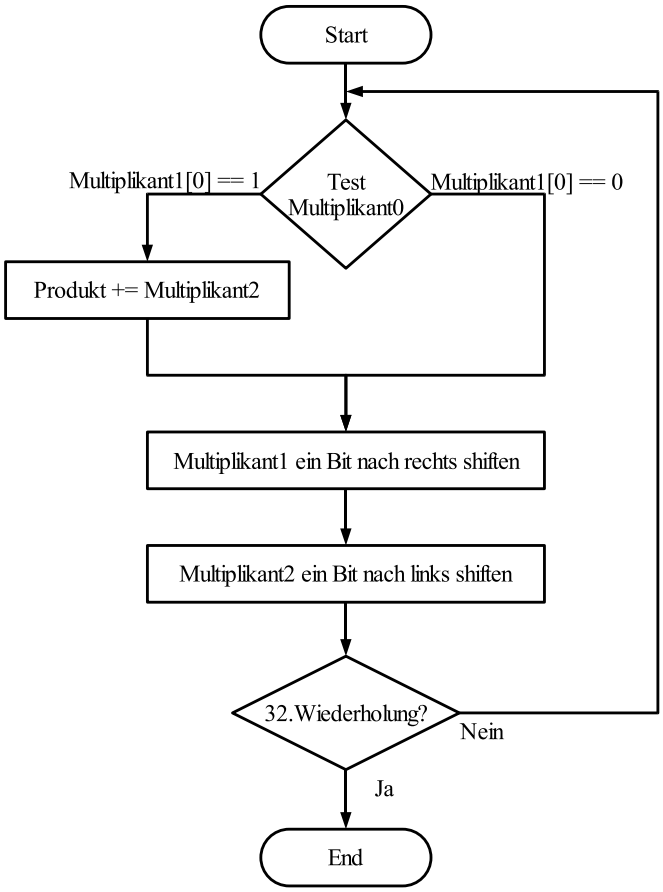

Abbildung 2: Flussdiagramm: Ganzzahlige Multiplikation von zwei 32 Bit Zahlen.

dargestellt. Ein Beispiel für die Multiplikation von zwei ganzen Zahlen in Hardware ist in Abbildung 2 auf der vorhergehenden Seite zu sehen.

Auf der anderen Seite kann man Flussdiagramme auch zur Beschreibung von sehr abstrakten, übergeordneten Abläufen verwenden. Sogar über den Computer hinaus, zum Beispiel um Geschäftsabläufe zu visualisieren.

Flussdiagramme haben zwar seit ihrer frühen Einführung Ende der 1940er an Bedeutung verloren, werden aber immer noch an einigen Stellen eingesetzt. Zum Beispiel als "visuelle Programmiersprache" bei Lego Mindstorms, wo man sich Programme aus Flussdiagramm-Legobausteinen am Bildschirm zusammenklicken kann. Sehr eng verwandt mit Flussdiagrammen sind auch UML Aktivitätsdiagramme.

1.2 Pseudocode

Pseudocode ist nicht normiert, wie Flussdiagramme. Meistens sehen Algorithmen in Pseudocode-Notation Quelltextfragmenten von "echten" Programmiersprachen sehr ähnlich, allerdings oft mit Teilen in normalem Fliesstext umgangssprachlich verfasst. Diese beschreiben, was an der entsprechenden Stelle im Algorithmus passieren soll.

Im allgemeinen gibt es die üblichen Schleifenkonstrukte (`for`, `while`) und Entscheidungsstrukturen (`if`, `case`) als Schlüsselwörter der fiktiven Programmiersprache. Entweder werden diese in einleitenden Textabschnitten eingeführt, oder als allgemein verständlich vorausgesetzt, wenn man z.B. davon ausgehen kann, dass der Leser bereits Erfahrungen mit einer höheren Programmiersprache besitzt und diese Elemente wiedererkennt und versteht.

Manchmal werden im Pseudocode auch mathematische Notationen wie z.B. Mengenoperationen verwendet.

Eine grosse Anzahl von Algorithmen als Pseudocode dargestellt, findet man beispielsweise im "Cormen" [CorLeiRiv90].

1.3 GPA

Der *Grade Point Average* ist ein Wert, der die Note einer Veranstaltung mit den benötigten Stunden in Beziehung setzt, die der erfolgreiche Abschluss dauerte. Dabei wird die erreichte Note durch die Anzahl der Stunden dividiert. Je länger man für einen Schein braucht, z.B. wenn man eine Veranstaltung wiederholen muss, um so niedriger wird der GPA für diese Veranstaltung.

Der GPA von Teilnehmern wird in beiden Papieren erwähnt, die in dieser Ausarbeitung behandelt werden.

1.4 Fortran 77

Shneiderman verwendet in seinen Experimenten FORTRAN[1] statt Pseudocode. Da die Sprache mittlerweile nicht mehr so gebräuchlich ist, von "Nischen" einmal abgesehen, ist in Listing 1 auf Seite 8 ein kleines Beispielprogramm abgedruckt, damit sich der geneigte Leser ein Bild machen kann.

[1] **For**mular **Tran**slator

FORTRAN stellt ein paar recht willkürlich wirkende, historisch bedingte Anforderungen an die Formatierung des Quelltextes. In den ersten sechs Spalten dürfen keine Anweisungen beginnen, dieser Platz ist für numerische (Sprung)marken reserviert. Ausnahme sind Kommentarzeilen, die in der ersten Zeile mit einem 'C' beginnen. Ferner dürfen Zeilen nicht länger als 72 Zeichen sein.

Man muss immer die numerischen Sprungmarken benutzen. Sogar bei Schleifen muss man das Ende der Schleife mit einer Marke versehen und diese am Schleifenbeginn angeben.

Strukturierte Programmierung ist nur bedingt möglich und kann auch sehr einfach durchbrochen werden. Es gibt z.B. Unterprogramme, denen man Rücksprungadressen in Form von Sprungmarken mitgeben kann, so dass ein '`RETURN` *`Marke`*' als `GOTO` agiert. Die Unterroutine `FOO()` in Listing 1 ist ein Beispiel für diese Technik. Umgekehrt kann man in Unterprogrammen auch mehrere Einstiegspunkte mit '`ENTRY`' angeben, so dass man auch mitten in ein Unterprogramm hineinspringen kann. Zusammen mit den nicht besonders aussagekräftigen Sprungmarken, kann man auf diese Weise sehr optimalen, aber ebenso unübersichtlichen "Spaghetti"-Code erstellen.

Listing 1: Beispiel für FORTRAN 77

```
1  C! ------------------------------------------------------------
2  C! Small test program in Fortran 77.
3  C! ------------------------------------------------------------
4        PROGRAM TEST
5        INTEGER ANSWER
6        ANSWER = 42
7        CALL FOO(ANSWER, *10, *20)
8   10   PRINT *, 'not printed...'
9   20   PRINT *, 'Hello World!'
10       PRINT 100, ANSWER    ! formatted output
11
12       CALL HEXTABLE()
13
14       STOP                 ! stop program execution here
15
16  100  FORMAT ('*The* answer is ', I 3,'!!!')
17       END
18
19 C! ------------------------------------------------------------
20 C! Return to first given label if X = 0, to second label otherwise.
21 C! ------------------------------------------------------------
22       SUBROUTINE FOO(X, *, *)
23       INTEGER X
24
25       PRINT *, X
26       IF (X .EQ. 0) THEN
27          RETURN 1          ! return to label #1
28       ELSE
29          RETURN 2          ! return to label #2
30       END IF
31       END
32
33 C! ------------------------------------------------------------
34 C! Print 0-255 as decimal and hex values.
35 C! ------------------------------------------------------------
36       SUBROUTINE HEXTABLE()
37       INTEGER J
38       DO 10, J = 0, 256, 1
39          PRINT 100, J, J
40  10   CONTINUE
41  100  FORMAT ('decimal:', I 4, ' hex:', O 4) !BUG: prints octal, not hex
42       END
```

2 Zusammenfassung der Quellen

Es folgt eine Zusammenfassung der beiden Artikel

- “Experimental Investigations of the Utility of Detailed Flowcharts in Programming” von Ben Shneiderman, Richard Mayer, Don McKay und Peter Heller [ShnMayMcK77]
- und “Structured Flowcharts Outperform Pseudocode: An Experimental Comparison” von David A. Scanlan. [Scanlan89]

2.1 Shneiderman

Ben Shneiderman, Richard Mayer, Don McKay und Peter Heller haben im Jahr 1977 insgesamt fünf Experimente durchgeführt.

Die Grundtätigkeiten beim Programmieren sind Komposition, Verständnis, Fehlersuche und Modifikation von Programmen. Flussdiagramme sollen eine nützliche, grafische Darstellung der Programmlogik darstellen. Shneidermans Ziel war die Überprüfung dieser Nützlichkeit bei den genannten Tätigkeiten.

Flussdiagramme sind seit der Einführung von Computern in den 40er Jahren Bestandteil des Programmierens. Sie waren so verbreitet, dass 1963 in den USA ein nationaler Standard vorgeschlagen wurde, der später bei der Einführung neuer Sprachkonstrukte (z.B. `DO`-Schleifen in FORTRAN) überarbeitet werden musste.

Eine Betrachtung von Texten über FORTRAN brachte die in Abbildung 3 ersichtliche verbreitete Nutzung von Flussdiagrammen als Darstellungsmittel.

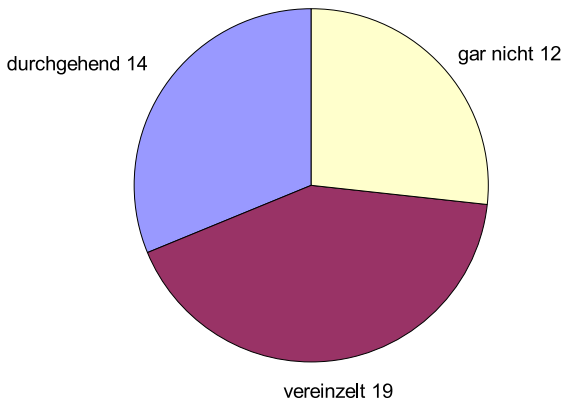

Abbildung 3: Flussdiagramme wurden in über $\frac{2}{3}$ der Texte über FORTRAN zur Darstellung von Algorithmen verwendet.

In Texten, die unhabhängig von einer konkreten Programmiersprache in das Themengebiet “Programmierung” einführen, sind Flussdiagramme das Haupttransportmittel für Algorithmen.

Ferner sollen automatisch generierte Flussdiagramme von bestehenden Programmen den Programmierern eine Hilfe bei der Fehlersuche- und Beseitigung, sowie beim Modifizieren von eigenem und fremdem Quelltext hilfreich sein. Diese Nützlichkeit war sehr umstritten, da Flussdiagramme zwar den logischen Programmablauf zeigen, aber nicht die wichtigen Zusammenhänge und Beziehungen zwischen Funktionen im Programm, welche für das Verständnis auch sehr wichtig sind.

Shneiderman und seine Kollegen erwähnen in der Einleitung Befürworter, Kritiker und vorhergehende Experimente zum Thema "Flussdiagramme".

So hat eine Studie behauptet, dass Regeln und Dienstordnungen in Flussdiagrammform weniger anfällig für Fehlinterpretationen und mit weniger Zeitaufwand umsetzbar sind.

Dies wurde in einem anderen Experiment bestätigt, bei dem eine Anleitung zum Telefonieren mit diversen Vor- und Durchwahlnummern an "Hausfrauen" und *Bell Telephony Laboratory* Mitarbeitern getestet wurden. Die "Hausfrauen" benötigten mit Flussdiagrammen, anstelle von Anweisungen in Fliesstext, weniger Zeit und selbst die "Profis" von *Bell* machten weniger Fehler beim Wählen.

Die Empfehlung von Flussdiagrammen in solchen Fällen wurde mit folgenden Punkten begründet:

- Hauptentscheidungskriterien werden an den Anfang verschoben.
- Die Komplexität des Textes wird reduziert.
- Wichtige und unwichtige Information wird klarer unterschieden.
- Die Informationsmenge wird beschränkt.

Wright und Reid [WriRei73] haben in einem ähnlichen Experiment gezeigt, dass sich Anweisungen in Fliesstext-Form dafür besser im Gedächtnis verankern als Flussdiagramme.

Diese beiden Experimente aus dem Bereich "Ergonomie" sind laut Shneiderman aber aus mehreren Gründen nicht ohne weiteres direkt auf das Programmieren übertragbar.

- Programme sind nicht in mehrdeutiger, menschlicher Sprache, sondern in klar definierten Programmiersprachen geschrieben.
- Die Experimente hatten nur einfache Aufgaben und nicht Komposition, Fehlersuche oder Modifikation von Programmen zum Gegenstand.
- Die meisten Flussdiagramme waren auf eine (Papier)Seite beschränkt.
- Die Testpersonen bekamen entweder ein Flussdiagramm *oder* eine Anleitung als Fliesstext, aber nicht *beides*.

2.1.1 Experiment I – Kompositon / Entwurf

An vielen Stellen in der Literatur wurde behauptet, dass Flussdiagramme nützlich beim Entwurf eines Programms sind, da sie dem Programmierer helfen, das Problem zu erfassen. Das soll mit diesem Experiment untersucht werden.

Testpersonen sind Studenten eines einführenden Kurses in FORTRAN. Sowohl der Dozent als auch das verwendete Buch nutzen Flussdiagramme. Das Experiment war in

den zweiten von drei Tests eingebunden, deren Note den Hauptteil der Benotung des Kurses ausmachte.

Die Studenten wurden in zwei Gruppen aufgeteilt, von denen eine Gruppe (34 Studenten) sowohl ein Flussdiagramm als auch ein Programm zu einem vorgegebenen Problem abgeben sollten, während die andere Gruppe (28 Studenten) nur das Programm anfertigen brauchte. Die Gruppen hatten so viel Zeit zur Verfügung, wie sie benötigten.

Beide Gruppen schnitten sehr gut ab, was nahelegte, dass alle Studenten mit der gestellten Aufgabe gut zurechtkamen. Der Punkteunterschied von 1% macht statistisch gesehen keinen signifikanten Unterschied, so dass die Anforderung ein Flussdiagramm vor der Implementation zu erstellen, weder Vor- noch Nachteile zu haben scheint. Und das, obwohl das Problem so ausgewählt wurde, dass ein relativ kompliziertes Verzweigungsmuster umgesetzt werden musste.

2.1.2 Experiment II – Verständnis

Flussdiagramme sollen das Verständnis von vorhandenem Quelltext erleichtern. Da Experiment I keinen Vorteil von Flussdiagrammen bei einfachen Programmen nachweisen konnte, wurden im zweiten Experiment komplexere Programmstrukturen verwendet.

Testpersonen waren wieder Studenten eines Einführungskurses in FORTRAN und das Experiment war Teil des dritten von fünf Tests, die zusammen den Hauptteil der Benotung des Kurses ausmachten. Von den 100 erreichbaren Punkten fielen 60 auf das Experiment.

Die 53 Studenten wurden in zwei etwa gleichgrosse Gruppen aufgeteilt, von denen beide zwei FORTRAN-Programme mit 27 bzw. 24 Anweisungen bekamen. Die eine Gruppe bekam zusätzlich ein Flussdiagramm zum ersten Programm, während die andere eines zum zweiten Programm erhielt.

Die Fragen, die bei beiden Gruppen identisch waren, verlangten von den Studenten die Angabe von Ausgabewerten für bestimmte Eingaben und das Nachvollziehen des Kontrollflusses. Es durfte wieder soviel Zeit aufgewendet werden, wie nötig war um die Fragen zu beantworten.

Die Bewertung war bei diesem Experiment einfacher, da die Ergebnisse eindeutig richtig oder falsch waren.

Eine Varianzanalyse der Ergebnisse ergab, dass die einzige signifikante Aussage war, dass das zweite Programm schwieriger zu verstehen war, d.h. mehr Zeit beanspruchte.

Das Ergebnis fand Shneiderman überraschend, weil die Programme viele Verzweigungen bzw. Sprünge enthielten und somit bei der Verwendung von Flussdiagrammen eigentlich Vorteile zu erwarten waren. Beobachtungen während des Tests haben gezeigt, dass die Flussdiagramme nicht besonders häufig zu Rate gezogen wurden.

2.1.3 Experiment III – Verständnis & Fehlersuche

Dieses Experiment untersuchte an fortgeschrittenen Anfängern den Effekt, den Flussdiagramme beim Verständnis und bei der Fehlersuche in Programmen haben.

Die Testpersonen waren Studenten aus FORTRAN-Kursen, in denen Flussdiagramme behandelt wurden und auch im Buch zum Kurs Anwendung fanden. Allerdings gab es zwei Gruppen von Studenten: Eine (43 Studenten) musste in ihren Kursen keine Flussdiagramme zu jedem Programm in den Hausaufgaben abgeben (NFC-Gruppe), die andere Gruppe (27 Studenten) hingegen musste zu jedem selbstgeschriebenen Programm

ein Flussdiagramm anfertigen (FC-Gruppe). Das Experiment hatte diesmal keinen Einfluss auf die Benotung und beide Gruppen wurden separat getestet.

Gegenstand des Experiments war ein in FORTRAN verfasstes Tic-Tac-Toe-Spiel mit 81 Zeilen Hauptprogramm und 43 und 23 Zeilen langen Unterprogrammen, also insgesamt 147 Zeilen. Das Hauptprogramm war kommentiert. Ferner erhielten die Studenten noch eine Ausgabe eines Programmablaufs und die Information, dass der Quelltext *mindestens* einen Fehler enthält, und dass dieser sich auch in der Ausgabe niederschlägt. Tatsächlich enthielt der Quelltext noch zwei weitere semantische Fehler, die an der Ausgabe nicht ersichtlich waren.

Die Gruppen wurden jeweils nochmal in drei, etwa gleichgrosse Untergruppen eingeteilt, von denen die erste Gruppe ein vierseitiges, ausführliches Flussdiagramm ("mikro"), die zweite ein einseitiges, abstrakteres Flussdiagramm ("makro") und die dritte *kein* Flussdiagramm, zusätzlich zum Quelltext erhielt.

Der Ablauf war in drei Schritte unterteilt. Im ersten Schritt wurden Quelltext, Flussdiagramme und Anweisungen ausgeteilt. Die Studenten erfuhren, dass es sich um ein Tic-Tac-Toe-Spiel handelt, welches nicht verlieren soll und das der Quelltext mindestens einen Fehler enthält. Sie sollten den Quelltext und das Flussdiagramm oder wenn gewünscht, nur das Flussdiagramm (sofern vorhanden) studieren. Danach sollten sie den oder die Fehler finden und beseitigen. Abgabe sollte die fehlerhafte Zeilennummer zusammen mit der korrigierten Zeile sein. Die NFC-Gruppe bekam 40 Minuten Zeit, währen die FC-Gruppe 10 Minuten länger arbeiten durfte.

Im zweiten Schritt wurden die Studenten über die drei Fehler und die nötigen Korrekturen informiert und mussten danach 11 Multiple-Choice Fragen beantworten, die Verständnis des Programms, grundlegende Programmierkenntnisse und die manuelle Simulation des Spiels betrafen. Die NFC-Gruppe hatte 20 Minuten und die FC-Gruppe wiederum 10 Minuten länger Zeit.

Im dritten und letzten Schritt sollten die Studenten einen Fragebogen ausfüllen, der ermittelte, wie die Studenten ihre Antworten auf die vorhergehenden Aufgaben auf einer Skala von 0 bis 9 einstufen würden. Die Gruppen, die ein Flussdiagramm zur Verfügung hatten, sollten zusätzlich noch Fragen zur Nützlichkeit beim Lösen der Aufgaben beantworten.

Die Daten der beiden Gruppen (NFC und FC) stammten von Testpersonen aus zwei unterschiedlichen Personenkreisen, und wurden unter unterschiedlichen Testbedingungen ermittelt. Deshalb konnte Shneiderman die erreichten Punkte nicht direkt vergleichen. Wohl aber die Trends innerhalb der Gruppen. Bei den Studenten der NFC-Gruppe schnitten die ohne Flussdiagramm am besten ab, gefolgt von denen mit dem "mikro"-Flussdiagramm und die wenigsten Punkte erhielten die mit dem "makro"-Flussdiagramm. Bei der FC-Gruppe bekamen die Studenten mit dem "mikro"-Flussdiagramm im Schnitt die meisten Punkte, gefolgt von denen mit dem "makro"-Flussdiagramm und das Schlusslicht bildeten hier die Studenten ohne grafische Unterstützung.

Diese Trends scheinen einen Zusammenhang zwischen dem Hintergrund (NFC/FC) und der Nützlichkeit von Flussdiagrammen nahezulegen. Die statistische Analyse der Ergebnisse zeigt aber, dass die Abweichungen nicht signifikant genug sind, um eine Aussage zu treffen.

Die Befragung über die Nützlichkeit ergab, dass die Testpersonen, die genau *einen* Fehler fanden, das Flussdiagramm als nützlich erachteten, während die, die keinen, zwei oder alle drei Fehler fanden, es als weniger hilfreich ansahen.

Das liesse sich damit erklären, dass der eine, mehr oder weniger offensichtliche Fehler, der auch in der Programmausgabe seine Spuren hinterliess, durch das Flussdiagramm

gefunden werden konnte – die beiden anderen aber nur mittels Durchgehens des Quelltextes.

2.1.4 Experiment IV – Modifikation

Da Flussdiagramme die Logik des Programms wiedergeben sollen, kann das Auffinden der zu modifizierenden Programmstelle erleichtert werden. Dies wurde in Experiment IV untersucht.

Testpersonen sind Studenten eines Programmierkurses im 2. Semester, die als fortgeschrittene Anfänger eingestuft werden. Es gab wie in Experiment III zwei Gruppen, von denen eine es gewohnt war Flussdiagramme zu jedem selbstverfassten Programm abzugeben, während die andere Gruppe Flussdiagramme zwar aus Vorlesung und Buch kannte, sie aber nicht in den Hausaufgaben verwenden musste. Das Experiment wurde an einem regulären Termin des jeweiligen Kurses durchgeführt.

Das ausgeteilte Material enthielt Anweisungen, einen FORTRAN-Quelltext, Beispielausgaben des Programms, Beschreibungen von drei Modifikationen und einen Fragebogen mit Fragen zu persönlichen Daten. Das Programm in den Unterlagen las Datensätze über Studenten ein und gab für jeden eine Noten-Übersicht aus. Der Quelltext bestand aus insgesamt 75 Zeilen von denen 27 Kommentarzeilen waren, von denen wiederum die meisten (23) in einem grossen Kommentarblock am Anfang des Quelltextes standen.

Die beiden Gruppen wurden, wie im vorhergehenden Experiment, wieder in drei Untergruppen mit jeweils “mikro”-, “makro”–Flussdiagrammen, bzw. ohne Flussdiagramm, aufgeteilt. Den Studenten wurde mitgeteilt, dass sie die drei beschriebenen Modifikationen innerhalb von 45 Minuten durchführen sollten, und dass die Lösungen nach Korrektheit und Lauffähigkeit bewertet werden.

Bezüglich der Erfassung der benötigten Zeit für die Teilaufgaben wurden die beiden Gruppen unterschiedlich behandelt – bei der NFC-Gruppe nahm ein Assistent die Zeit für jeden Studenten, während bei der FC-Gruppe jeder Student selber dafür verantwortlich war, die benötigte Zeit pro Modifikation festzuhalten.

Um die Analyse zu vereinfachen wurden die Daten von 12 Studenten nicht berücksichtigt. Fünf von ihnen hatten den Kurs abgebrochen und sieben hatten einen GPA ≤ 2.5. Auf diese Weise wurden beide Gruppen (NFC/FC) mit jeweils 30 Studenten gleich gross. Da die meisten Teilnehmer die dritte Modifikation zeitlich nicht mehr schafften, wurden auch diese Daten nicht berücksichtigt.

In die Bewertung des Experiments gingen zum Beispiel falsch formatierte Ausgaben, Modifikationen an der falschen Stelle im Programm, nicht korrekt umgesetzte Modifikationen und Syntaxfehler ein. Shneiderman bemerkt hier, das eigentlich nur die Modifikation an der falschen Stelle im Programm durch ein Flussdiagramm vermeidbar gewesen wäre.

Eine statistische Analyse der Ergebnisse lässt die folgenden zwei Aussagen zu: Die Studenten mit Flussdiagrammen haben weniger Fehler gemacht und die zweite Modifikation war schwieriger als die Erste. Wenn man allerdings nur den Fehler berücksichtigt, der am ehesten durch ein Flussdiagramm vermeidbar gewesen wäre, dann gibt es keinen signifikanten Unterschied mehr zwischen den beiden Gruppen.

Das ein detailliertes Flussdiagramm keinen Einfluss auf das Ergebnis hatte, legte nahe, dass im Flussdiagramm und in Quelltext die gleiche Menge an Informationen steckt, d.h. Flussdiagramme keine nützlichen Zusatzinformationen liefern. Um ein Programm modifizieren zu können, sollte man es auf einer abstrakteren Ebene verstanden haben

und intuitiv wäre dazu ein abstrakteres Flussdiagramm hilfreich. Aber auch hier waren keine Unterschiede feststellbar. Scheinbar liefert auch das “makro”-Flussdiagramm keine zusätzlichen Informationen.

Shneiderman selbst hat ein paar Punkte in seinem Experiment ausgemacht, welche den fehlenden Unterschied zwischen den Flussdiagramm-Gruppen erklären könnten. Zum einen könnten die Ergebnisse bei einem wesentlich längerem, komplexerem Programm vor allem bei “makro”-Flussdiagrammen besser ausfallen, da es einen besseren, kompakteren Überblick über das gesamte Programm bietet. Die Studenten der FC-Gruppe waren darüber hinaus besser mit dem Gegenstand des Programms vertraut, da es sich um das Benotungsschema ihrer Universität handelte, während die NFC-Gruppe ein anderes kannte. Und zuletzt war der verhältnismässig umfangreiche Kommentarblock am Anfang des Quelltextes so ausführlich, dass er vielleicht schon genug Informationen enthielt, um die Modifikationen durchzuführen.

2.1.5 Experiment V – Verständnis

Die vorhergehenden Experimente haben nicht gezeigt, welche Informationen sich aus einem Flussdiagramm ablesen lassen. Deshalb soll im letzten Experiment der Informationsgehalt eines detaillierten Flussdiagramms bei einer Verständnisaufgabe untersucht werden.

Testpersonen waren 58 Studenten eines achtwöchigen, einführenden Sommerkurses in die Informatik. Das Experiment wurde im Rahmen des “Quiz” am Anfang der sechsten Woche durchgeführt. Zu diesem Zeitpunkt waren die Studenten mit FORTRAN und Flussdiagrammen vertraut.

Das Quiz bestand aus einem Merge-Algorithmus, der den Studenten vorher nicht bekannt war und fünf Fragen für die es je 20 Punkte gab. Die Zeit war auf 25 Minuten beschränkt. Zwei Fragen erforderten manuelle Simulation des Programmablaufs und drei weitere Fragen betrafen Ablauf und Eigenschaften des Algorithmus. Es gab wieder drei Gruppen, von denen die erste einen 23 Zeilen langen FORTRAN-Quelltext bekam, die zweite ein detailliertes “mikro”-Flussdiagramm auf einer Seite und die dritte Gruppe beides.

Sieben Ergebnisse wurden bei der Auswertung nicht berücksichtigt. Drei Studenten führten den Kurs nicht zuende durch und vier weitere hatten eine Benotung schlechter als “D”[2].

Eine Analyse der Varianz legt nahe, dass alle Gruppen gleich gut waren. Die Ergebnisse zeigen zwar, dass die Gruppe, die nur den Quelltext zur Verfügung hatte, nach Punkten am besten abschnitt, aber die Varianz innerhalb der Gruppen war so gross, dass sich keine statistisch signifikanten Unterschiede ermitteln liessen. Die Ergebnisse scheinen zu belegen, dass die Informationen in einem (detaillierten) Flussdiagramm äquivalent zu denen im Quelltext sind.

Obwohl die Ergebnisse statistisch nicht signifikant sind, könnte das gute Abschneiden der Gruppe, die nur den Quelltext zur Verfügung hatte, bedeuten, dass Quelltext für schrittweise, manuelle Simulation besser geeignet ist, und dass Quelltext Informationen enthält, die nicht im Flussdiagramm darstellbar sind, die aber bei Verständnisfragen hilfreich sind.

Flussdiagramm und Quelltext zusammen scheinen das Verständnis eher zu behindern.

[2] Ein “D” entspricht einer 4, also “*ausreichend*” in unserem Schulnotensystem von 1–6.

2.1.6 Schlussfolgerungen

Shneiderman kommt zu dem Schluss, dass Flussdiagramme im wesentlichen reduntante Darstellungen von Quelltexten sind. Sie enthalten sogar weniger Informationen, z.B. keine Deklarationen, Sprungmarken und Ein-/Ausgabeformatierungen, sind andererseits aber viel umfangreicher als kompakter Quelltext.

Der Trend in Richtung Top-Down-Entwicklung und "strukturierte" Elemente in neuen Programmiersprachen[3] könnten detaillierte Flussdiagramme überflüssig machen. Zukünftige Experimente sollten sich eher auf "makro"-Flussdiagramme konzentrieren.

Ausserdem würde Shneiderman gerne Experimente mit erfahreneren Programmierern und wesentlich grösseren und komplexeren Programmen (ab 1000 Zeilen aufwärts) sehen. Und auch die Nützlichkeit von Flussdiagrammen für "Nicht-Programmierer" und Manager wäre interessant.

2.2 Scanlan

Scanlan hat im Jahr 1981 zufällig in einem Kurs über Datenstrukturen entdeckt, dass überwältigend viele Studenten Flussdiagramme, Pseudocode-Beschreibungen vorziehen, obwohl frühere Untersuchungen wie z.B. die von Shneiderman [ShnMayMcK77] gezeigt haben, dass Flussdiagramme keine Vorteile beim Verständnis von Algorithmen gegenüber Pseudocode aufweisen. Das hat ihn dazu veranlasst selber ein Experiment durchzuführen, um zu zeigen, dass Flussdiagramme sehr wohl einen signifikanten Vorteil gegenüber Pseudocode haben können.

Scanlan erwähnt, genau wie Shneiderman, zunächst einige Studien, die für bzw. gegen seine Behauptung sprechen.

2.2.1 Befürworter

Wright und Reid [WriRei73] haben herausgefunden, dass bei höherer Komplexität von Algorithmen, auch der Bedarf an grafischen Beschreibungen steigt. Sie gaben ihren Testpersonen Algorithmen mit verschiedenen Schwierigkeitsgraden in Form von Fliesstext, Tabellen und Flussdiagrammen. Bei komplexen Algorithmen machten die Testpersonen weniger Fehler wenn sie mit Flussdiagrammen arbeiteten. Bei einfachen Algorithmen konnte kein Unterschied festgestellt werden.

Eine andere Studie hat zwei verschiedene Arten von Flussdiagrammen mit Beschreibungen als Fliesstext-Anweisungen verglichen. Das Verständnis, gemessen an gemachten Fehlern, war besser wenn die Testpersonen mit den Flussdiagrammen arbeiteten.

Des weiteren wurde bemerkt, dass entsprechend eingerückter Quelltext das Verständnis erleichtert. Daraus leitet Scanlan die Vermutung ab, dass eine "räumliche" Darstellung dem Verständnis zuträglich ist, auch wenn es sich in diesem Fall nicht um eine "richtige" grafische Notation handelte, sondern eben nur um Einrückung von Quelltext.

2.2.2 Kritiker

Scanlan stellt fest, dass die meisten vorhergehenden Experimente, die Flussdiagramme und Pseudocode verglichen hatten, keinen Unterschied zwischen beiden Darstellungsformen bezüglich des Verständnisses bei den Testpersonen aufwiesen. Er behauptet,

[3] Hier sind Kontrollstrukturen gemeint, die kein "unkontrolliertes" Springen im Program à la `GOTO` erlauben.

dass diese Experimente entweder fehlerhaft waren und/oder keine *strukturierten* Flussdiagramme verwendet haben.

Bei den Kritikern von Flussdiagrammen hat Scanlan den in Abschnitt 2.1 auf Seite 9 vorgestellten Artikel von Shneiderman als den am meisten verbreiteten und zitierten ausgemacht.

Shneiderman hat fünf Experimente durchgeführt, die keine signifikante Unterstützung zum Programmverständnis durch Flussdiagramme, im Gegensatz zu alternativen Präsentationstechniken wie zum Beispiel FORTRAN Quelltext, feststellen konnten. Er ist zu dem Schluss gelangt, dass Flussdiagramme nicht mehr oder weniger Informationen transportieren, als Quelltext in einer Programmiersprache.

An dieser Stelle geht Scanlan etwas detaillierter auf das zweite Experiment (siehe Zusammenfassung in Abschnitt 2.1.2 auf Seite 11) von Shneiderman ein und führt drei Kritikpunkte an:

1. Die Testpersonen hatten so viel Zeit, wie sie benötigten, aber diese Zeit wurde nicht als Messgrösse erfasst. Hier hätte sich unter Umständen herausstellen können, dass die benötigte Zeit der Flussdiagramm-Gruppe gegenüber der FORTRAN-Gruppe eine deutliche Differenz aufweist.

2. Fünf der Verständnisfragen erforderten die Formulierung der Ein- und Ausgabe als FORTRAN-Anweisungen. Diese sprachspezifischen Informationen waren nur im FORTRAN-Quelltext vorhanden, nicht aber in den Flussdiagrammen, so dass die FORTRAN-Gruppe an dieser Stelle einen Vorteil hatte.

3. Das Experiment benutzte relativ einfache Algorithmen. Vielleicht wären die Vorteile von Flussdiagrammen erst bei komplexeren Algorithmen richtig zum Tragen gekommen.

Eine weitere Studie fand keinen signifikanten Unterschied beim Verständnis zwischen Flussdiagrammen und einer Programmdesign-Sprache. Das liesse sich laut Scanlan wieder damit erklären, dass die Zeit keine Messgrösse war. Ausserdem nahmen an dem Experiment nur 20 Personen teil.

2.2.3 Hypothesen

Scanlan stellt folgende Hypothesen auf: Strukturierte Flussdiagramme...

1. ... sind schneller zu Verstehen.

2. ... ergeben weniger Fehler beim Verstehen.

3. ... geben Studenten mehr Vertrauen in ihr Verständnis eines Algorithmus.

4. ... verringern die Zeit zur Beantwortung von Fragen über den Algorithmus.

5. ... verringern die Anzahl der Blicke der Testpersonen auf den Algorithmus.

Da die Hypothesen an drei Algorithmen unterschiedlicher Komplexität getestet wurden, handelt es sich um insgesamt 15 Hypothesen, die überprüft wurden.

2.2.4 Testpersonen

Der Kreis der Testpersonen bestand aus 82 freiwilligen Studenten der Fachrichtung *Management Information Science*, die alle aus Veranstaltungen "rekrutiert" wurden, welche PASCAL[4], COBOL[5], Datenstrukturen oder Softwaretechnik zum Thema hatten.

Im Schnitt waren die Teilnehmer 27.8 Jahre alt, hatten in Informatik einen GPA von 3.24 und beherrschten 3.35 Programmiersprachen. Die Programmiererfahrung reichte von "fortgeschrittenen Anfängern" bis zu "Profis". In welchem Stadium ihres Studiums sich die Testpersonen befanden, ist in Abbildung 4 aufgeschlüsselt.

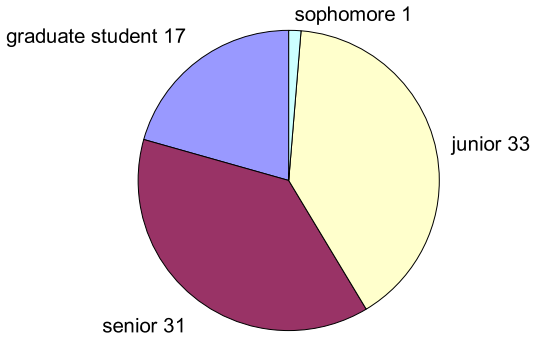

Abbildung 4: Die Grafik zeigt, wieviele Testpersonen sich in welchem Stadium des Studiums befanden. In der Regelstudienzeit von vier Jahren ist jedem Jahr einer der folgenden Begriffe zugeordnet: *freshman*, *sophomore*, *junior* und *senior*. Studenten, die nach dieser Zeit eine "master thesis" (entspricht in etwa einer Diplomarbeit) schreiben, bezeichnet man als *graduate students*.

2.2.5 Algorithmen

Ein Algorithmus war aus `IF ... THEN ... ELSE` Verzweigungen zusammengesetzt, wobei immer beide Zweige vorhanden waren. Es gab drei solcher Algorithmen mit unterschiedlicher Komplexität. Die Abbildungen 5 auf der folgenden Seite, 6 auf der nächsten Seite und 7 auf Seite 19 zeigen die Algorithmen als Flussdiagramme. Die Bedingungen wurden mit Adjektiven und die Anweisungen mit Verben belegt. Jeder Algorithmus lag sowohl als Flussdiagramm, als auch in Pseudocode-Notation vor.

Adjektive und Verben wurden zufällig auf den entsprechenden Positionen im Algorithmus verteilt und ergeben keinen "Sinn". Das Verteilen der Worte in die Struktur der drei Algorithmen wurde zweimal durchgeführt, so dass es einen Satz A und einen Satz B mit je drei Flussdiagrammen und Pseudocodes gab.

Aus diesen 12 Einzelteilen wurden zwei Sequenzen zusammengestellt. Eine bestand aus den drei Flussdiagrammen aus Satz A und den Pseudocodes aus Satz B und die andere aus den Flussdiagrammen aus Satz B und den Pseudocodes aus Satz A.

[4] Imperative Programmiersprache von Nikolaus Wirth. Nach dem Mathematiker Blaise Pascal benannt.

[5] Die **Co**mmon **B**usiness **O**riented **L**anguage ist eine imperative Programmiersprache, die keine Funktionen, Module oder ähnliche Möglichkeiten kennt, um ein Programm zu untergliedern.

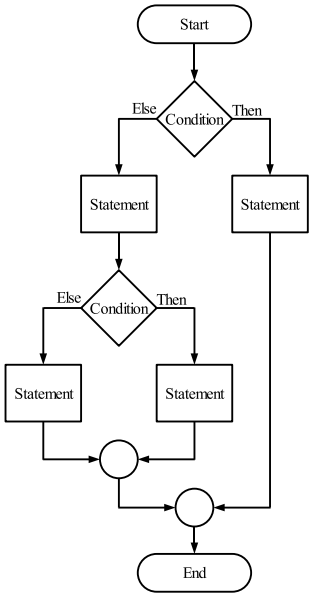

Abbildung 5: Einfacher Algorithmus als Flussdiagramm.

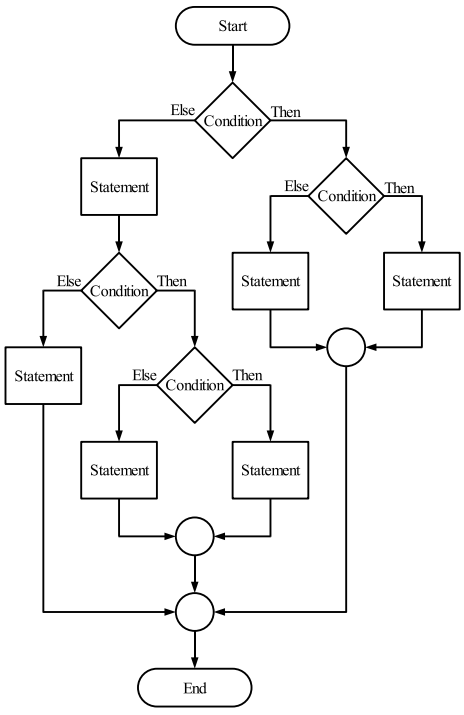

Abbildung 6: Mittlerer Algorithmus als Flussdiagramm.

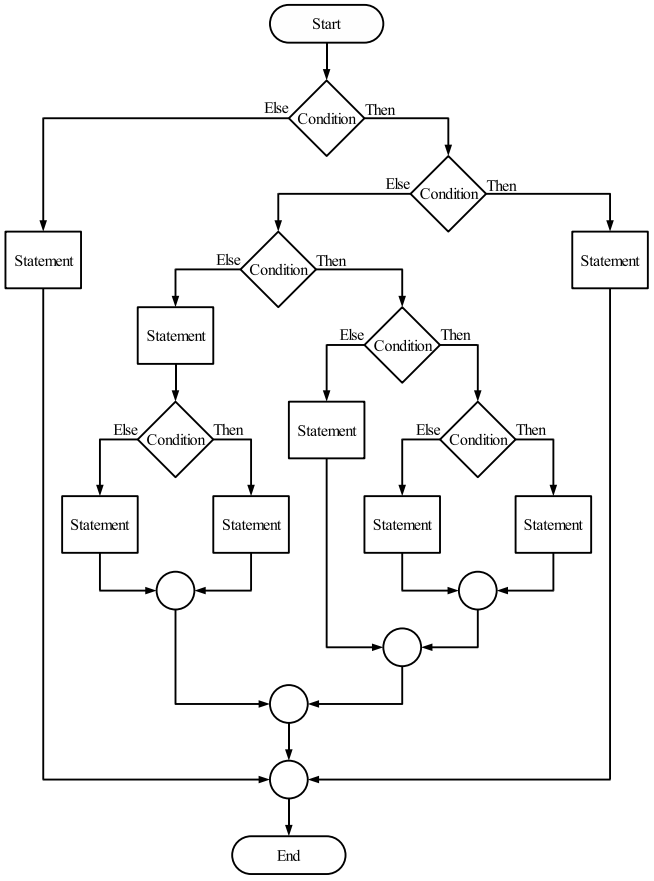

Abbildung 7: Komplexer Algorithmus als Flussdiagramm.

Welche der Sequenzen eine Testperson bekam und in welcher Reihenfolge die Algorithmen bearbeitet werden sollten, wurde zufällig festgelegt, um Lern- und Ermüdungseffekte auszubalancieren.

Sowohl die Flussdiagramme, als auch der Pseudocode wurden in der gleichen Schriftart gesetzt, um die Lesbarkeit als Störfaktor zu eleminieren.

2.2.6 Ausrüstung und Material

Das Experiment wurde in einem Raum des Psychologieinstituts durchgeführt, der gegen äussere Einflüsse (Geräusche, Licht) abgeschirmt war.

Folgendes Material kam zum Einsatz:

- PC mit MS-DOS (inkl. Monitor und Tastatur),
- spezielles Anzeigegerät für die Algorithmen,
- Sprachausgabegerät,
- 12 Karten (8.5×11 inch[6]) mit den Algorithmen,
- Fragebogen für demographische Daten,
- 6-seitiges Papier über den Ablauf des Experiments,
- 82 Disketten für die Ergebnisse jedes Teilnehmers,
- und 1 Assistent.

Der PC diente zur Anzeige der Fragen und Entgegennahme der Antworten, sowie zur Zeiterfassung und führte via Sprachausgabegerät sowohl Testperson, als auch Assistent durch das Experiment. Auf diese Weise wurde versucht, den “Störfaktor” *Mensch* zu minimieren.

Das spezielle Anzeigegerät für die Algorithmen war so konstruiert, dass entweder der Algorithmus sichtbar (beleuchtet) war und dafür der Bildschirm des Rechners nichts anzeigte, oder umgekehrt. Ausserdem hat es die Zeit erfasst, die der Algorithmus illuminiert wurde. Die Testperson konnte nach Belieben zwischen beiden Zuständen umschalten.

Der Assistent hatte die Aufgabe auf Anweisung des Computers die Karten mit den Algorithmen im Anzeigegerät auszutauschen.

2.2.7 Ablauf des Experiments

Einige Tage vor dem Experiment bekamen die Teilnehmer das 6-seitige Papier mit der Beschreibung des Experiments und eines Testlaufs ausgehändigt. Bei Testantritt musste jeder Teilnehmer einen kleinen Test mit 10 Fragen zu diesem Papier bestehen, d.h. 90% richtig beantworten. Falls das nicht gelang, hatte der Prüfling eine Viertelstunde Zeit, das Papier noch einmal durchzulesen und den Test zu wiederholen. Zusammen mit der Durchführung des im Papier beschriebenen Testdurchlaufs, sollte damit sicher gestellt werden, dass alle Teilnehmer den Ablauf des Experiments gut verstanden

[6] Das ist das amerikanische *letter*-Format ($\approx 21.59 \times 27.94$ cm). Es ist etwas breiter und etwas kürzer als DIN A4.

und verinnerlicht haben. Laut Scanlan ist ein schlechtes Verständnis des Ablaufs eines Experiments bei den Testpersonen eine Hauptfehlerquelle, die es auszuschliessen galt.

Testdurchlauf und "Ernstfall" waren identisch aufgebaut und durch eine Pause getrennt, in der die Testperson dem Assistenten Fragen stellen konnte. Auch während der Durchläufe durften dem Assistenten Fragen zum Experiment gestellt werden – von beiden Möglichkeiten wurde allerdings kaum Gebrauch gemacht.

Die Testperson bekam nacheinander jeden Algorithmus, und Fragen dazu präsentiert. Für das Studieren des Algorithmus und die Beantwortung der Fragen durfte sich die Testperson soviel Zeit wie benötigt nehmen und zwischen den Algorithmen Pausen einlegen. Im Schnitt dauerte das gesamte Experiment $1\frac{1}{2}$ Stunden pro Testperson.

Die Fragestellung bestand aus einer Kombination von Verben, zu denen die Testperson den Zustand (*wahr*, *falsch* oder *unbekannt*) jeder einzelnen Bedingung/Verzweigung im Algorithmus angeben sollte. Je grösser der Algorithmus, umso mehr Fragen, mit mehr Verben wurden gestellt. Siehe Tabelle 1.

Algorithmus	Fragen	Verben
einfach	6	2
mittel	9	4
komplex	10	6

Tabelle 1: Anzahl der Fragen und Verben pro Algorithmus.

Ein Beispiel für eine Frage zum einfachen Algorithmus (Flussdiagramm in Abbildung 5 auf Seite 18), der in Abbildung 8 mit einer Belegung von Bedingungen und Anweisungen versehen ist, wäre: Geben sie die Belegung aller Prädikate an, wenn das Nahrungsmittel gekocht[7] wird. Die korrekte Antwort lautet in diesem Fall, das `GREEN` mit *wahr* belegt sein muss und `CRISPY` mit *unbekannt*, d.h. man kann keine Aussage treffen.

```
PROC
  IF GREEN
    THEN
      BAKE
    ELSE
      BOILE
      IF CRISPY
        THEN
          STEAM
        ELSE
          FRY
      END IF
  END IF
END PROC
```

Abbildung 8: Der einfache Algorithmus als Pseudocode. Die Bedingungen und Anweisungen sind mit Adjektiven und Verben belegt.

Die gemessenen Grössen für jeden Algorithmus beim Experiment waren

- die Zeit (Sekunden), die der Algorithmus angeschaut wurde,
- Prozent der korrekt beantworteten Fragen,

[7] kochen – to boile

- der Grad der Selbstsicherheit der Testperson bei jeder Antwort,
- die Zeit (Sekunden), die zum Lesen und Beantworten der Fragen benötigt wurde,
- und die Anzahl der Blicke auf den Algorithmus.

Die Selbstsicherheit bei den Antworten mussten die Testpersonen selbst auf einer Skala von 1–4 für jede Antwort angeben.

Scanlans Hauptaugenmerk lag auf der benötigten Zeit, da die Fehleranzahl nicht besonders aussagekräftig ist, wenn die Testperson soviel Zeit bekommt, wie sie benötigt.

2.2.8 Ergebnisse

Die Ergebnisse bringen Scanlan zu dem Schluss, dass Algorithmen mittels Flussdiagrammen schneller zu verstehen sind, als durch Pseudocode. Und dieser Vorteil ist umso grösser, je komplexer der betrachtete Algorithmus ist.

Bis auf eine Ausnahme schnitten Flussdiagramme bei den Messergebnissen besser ab als Pseudocode – zum Beantworten der Fragen benötigten die Testpersonen beim einfachen Algorithmus bei beiden Darstellungsformen die gleiche Zeit.

Die Messergebnisse zeigen, dass die Testpersonen bei Flussdiagrammen deutlich weniger Fehler machten, mehr Vertrauen in ihre Antworten hatten, weniger Zeit für die Antworten benötigten, und weniger oft das Flussdiagramm anschauten.

Scanlan sieht seine Hypothesen durch die Ergebnisse gestützt. Verwundert hat ihn der deutliche Unterschied zwischen den Zeiten beim einfachen Algorithmus, den die Testpersonen damit verbrachten, die Algorithmen zu betrachten.

Er zieht den Schluss, dass grafische Notationen Vorteile bringen können und schlägt vor, weitere Experimente in dieser Richtung durchzuführen. Nicht nur mit Flussdiagrammen, sondern auch mit anderen grafischen Darstellungsformen von Programmlogik.

3 Beurteilung

Ab diesem Abschnitt folgt meine Meinung zu den beiden Experimenten und den Schlussfolgerungen der Autoren. Ausserdem Gedanken zu eventuell interessanten Folgeexperimenten bzw. Themen, die eine nähere Betrachtung wert sein könnten.

3.1 Shneiderman

Das erste Experiment (2.1.1 auf Seite 10) sagt meiner Meinung nach sehr wenig darüber aus, ob Flussdiagramme einen Mehrwehrt haben oder nicht. Wenn man zeigen will, dass ein Problem mit Flussdiagrammen besser gelöst werden kann, dann müsste man das Problem so wählen, dass ohne Flussdiagramm auch wirklich Verständnisschwierigkeiten auftreten können. Das sehr gute Abschneiden *aller* Teilnehmer lässt vermuten, dass das Problem zu trivial war.

Dieser Eindruck wird verstärkt wenn im zweiten Experiment (2.1.2 auf Seite 11) ein komplexeres Programm verwendet wird – mit 27 Zeilen Quelltext. Wie die kurze Übersicht über FORTRAN in Abschnitt 1.4 auf Seite 6 zeigt, kann man mit dieser Sprache richtig komplizierten "Spaghetti-Code" schreiben, wenn man es darauf anlegt. Aber Programme, die nur knapp über einen Bildschirm[8] reichen, sind normalerweise nicht besonders komplex. Zumindest nicht, wenn eine Zeile höchstens 72 Zeichen lang sein darf und normalerweise eine Anweisung pro Zeile angegeben wird.

Im dritten Experiment (2.1.3 auf Seite 11) durften die Teilnehmer, die ein Flussdiagramm bei ihrem Material hatten, wenn sie wollten auch nur dieses studieren. Die Aufgabe war dann aber, die Korrektur anhand der Zeilennummer im Quelltext und die korrigierte Zeile anzugeben. Damit hatten sie den Nachteil sich erstmal im Quelltext zurechtfinden zu müssen, den sie sich nicht angeschaut hatten. Jedenfalls sofern das Flussdiagramm keine Zeilennummern enthielt – wovon ich nicht ausgehe. Ausserdem waren zwei der drei Fehler gar nicht ohne den Quelltext zu finden.

Ferner habe ich den Sinn nicht ganz verstanden, dass die FC-Gruppe 10 Minuten mehr Zeit bekam. Sinnvoll wäre mir höchstens erschienen, wenn die Gruppen, die tatsächlich im Test ein Flussdiagramm zur Verfügung hatten, mehr Zeit bekommen hätten, weil sie mehr Material zum lesen hatten. Aber ein Zeitunterschied aufgrund der "Vorbildung", obgleich sowohl FC- als auch NFC-Gruppe auch Untergruppen mit Aufgaben hatten, welche die bisherige Arbeitsweise der Studenten bevorzugten, erscheint mir wenig plausibel.

Den Ausschluss von Teilnehmern aus der Auswertung in Experiment Nummer vier (2.1.4 auf Seite 13) und fünf (2.1.4 auf Seite 13) kann ich nur bedingt nachvollziehen. Wenn die Absicht war, Extremwerte an den "Rändern" auszuschliessen, dann machen Studenten mit einem schlechten Notendurchschnitt noch Sinn, aber jemand, der den Kurs abbricht, muss nicht zwangsläufig schlecht sein. Und ein schlechter GPA kann auch bedeuten, das ein Student den Kurs wiederholt hat, und eine mittelmässige Note bekam.

Was mich am Shneiderman-Papier am meisten irritierte, war die Tatsache, dass er an vielen Stellen gesagt hat "statistisch gesehen kann man keine Aussage treffen" – es dann aber doch getan hat. Zwar immer im Konjuntiv, aber es hat bei mir trotzdem einen leicht unseriösen Eindruck hinterlassen.

[8] 80×24 Zeichen Terminal.

Die von Shneiderman aufgestellte Behauptung, dass Flussdiagramme nur redundante Informationen enthalten, die im Quelltext vorhanden sind, trifft auf die untersuchten, kleinen FORTRAN-Programme wahrscheinlich zu. Dass zumindest aus kodierungstheoretischer Sicht im Flussdiagramm nicht mehr Information steckt als im Quelltext, ist leicht einzusehen, wenn man sich überlegt, dass ein Flussdiagramm per Programm aus gegebenem Quelltext generiert werden kann.

Zu untersuchen wäre, wie es sich verhält, wenn die notwendigen Informationen sich nicht alle im Quelltext wiederfinden lassen. Dafür gäbe es mehrere Gründe. Algorithmen, die sich über mehrere Systeme erstrecken, vielleicht sogar mit "Kontrollfüssen" ausserhalb von Computern. Oder Algorithmen, die über eine Vielzahl von Objekten verteilt sind und die erst zur Laufzeit *dynamisch* zusammengesetzt werden und deshalb nicht ohne weiteres aus *statischem* Quelltext ersichtlich sind.

3.2 Scanlan

Das die Schriftart eine "Störgrösse" sein kann, hat Scanlan selbst bemerkt und bei den Diagrammen und beim Pseudocode die gleiche benutzt. Das der Text durchgehend in Grossbuchstaben gesetzt wurde, ist dann aber eine typographische Katastrophe, da sich derartiger Text nur schwer lesen lässt. Unterschiedliche Worte lassen sich schwieriger auseinanderhalten. Vor allem wenn sie alle ungefähr gleich lang sind. Somit wird die Orientierung im Pseudocode unnötig erschwert. Auf das Flussdiagramm trifft das nicht so stark zu, weil man sich dort auch räumlich orientieren kann.

Ebenso ist die Einrückung von nur zwei Leerzeichen pro Ebene nicht gerade förderlich um die Ebenen im Pseudocode auseinanderzuhalten, oder das Ende einer Ebene zu finden. Insgesamt ist der Pseudocode meiner Meinung nach sehr leserunfreundlich formatiert.

Die verwendeten Algorithmen erscheinen mir sehr praxisfremd. Einen Entscheidungsbaum linear durchzugehen ist kein besonders anspruchsvoller Algorithmus. Ich vermute, dass die Vorteile, die durch die Flussdiagramme entstanden, an Gewicht verlieren würden, wenn die einfachen Verzweigungsstrukturen mit *sinnvollen* Bedingungen und Aktionen bezeichnet wären. Das Experiment hat in meinen Augen nicht das *Verständnis* gemessen, sondern wie gut/schnell sich ein gegebener Algorithmus auswendig lernen, oder "runterbeten" lässt. Das entspricht nicht meiner Definition davon, dass jemand einen Algorithmus *versteht*. Scanlan fragt nur ab *was* passiert, aber nicht *warum* – was bei der Sinnfreiheit seiner Algorithmen natürlich auch nicht möglich ist.

Angenommen seine Ergebnisse wären auch auf andere Arten von Algorithmen übertragbar, dann kann ich seiner Folgerung, dass Flussdiagramme umso nützlicher sind, je komplexer der Algorithmus ist, nur insoweit folgen, dass es eine Obergrenze gibt, ab der ein Flussdiagramm sehr unübersichtlich wird. Ab wievielen Seiten das im allgemeinen der Fall ist, müsste man experimentell überprüfen.

Wenn man das Experiment heutzutage wiederholen würde, dann könnte man die Störgrösse "Assistent" weiter reduzieren, indem man kein spezielles Anzeigegerät für die Algorithmen verwendet, bei dem ein Assistent die Anzeige manuell austauschen muss, sondern einen zweiten Monitor oder zwei Sichten auf einem Monitor zum Umschalten zwischen Algorithmus und Fragen. Das "Sprachausgabegerät" wäre eine normale Soundkarte, dafür wäre das Steuerprogramm auch grösser als die gigantischen 109 KB, die Scanlan erwähnt. Die Beschreibung der Gerätschaften und Programmgrössen war fast so amüsant wie Anzeigen in alten Computerzeitschriften. :-)

Literaturverzeichnis

[CorLeiRiv90] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest: Introduction to Algorithms. Cambridge: The MIT Press, 1998

[InfDuden93] Duden »Informatik«: Ein Sachlexikon für Studium und Praxis. 2. Auflage, Mannheim; Leipzig; Wien; Zürich: Dudenverlag, 1993

[PatHen98] David A. Patterson, John L. Hennessy: Computer Organization & Design: the hardware/software interface. 2nd edition. San Francisco: Morgan Kaufman Publishers, Inc., 1998

[Scanlan89] David A. Scanlan: Structured Flowcharts Outperform Pseudocode: An Experimental Comparison Software, IEEE Publication Date: Sep 1989 On page(s): 28-36 Volume: 6, Issue: 5

[ShnMayMcK77] Ben Shneiderman, Richard Mayer, Don McKay and Peter Heller: Experimental investigations of the utility of detailed flowcharts in programming, Commun. ACM, volume 20, number 6, 1977, pages 373–381

[WriRei73] P. Wright, F. Reid: Written Information: Some Alternatives for Prose for Expressing the Outcomes of Complex Contingencies, Applied Psycology, Feb. 1973, pages 160–166

Kolophon

Die vorliegende Ausarbeitung wurde mit *XEmacs* als LaTeX2e Quelltext erfasst und mit *pdfLaTeX* übersetzt. Die Zeichnungen und Diagramme sind mit *OpenOffice 1.1.0* erstellt und von dort als EPS exportiert worden.

Der Text ist in der Schriftart *Bookman* gesetzt. Als serifenlose Schriftart für Überschriften, Kopfzeilen u.ä. kommt *AvantGarde* zum Einsatz. Für Maschinenschrift wurde *Courier* gewählt.