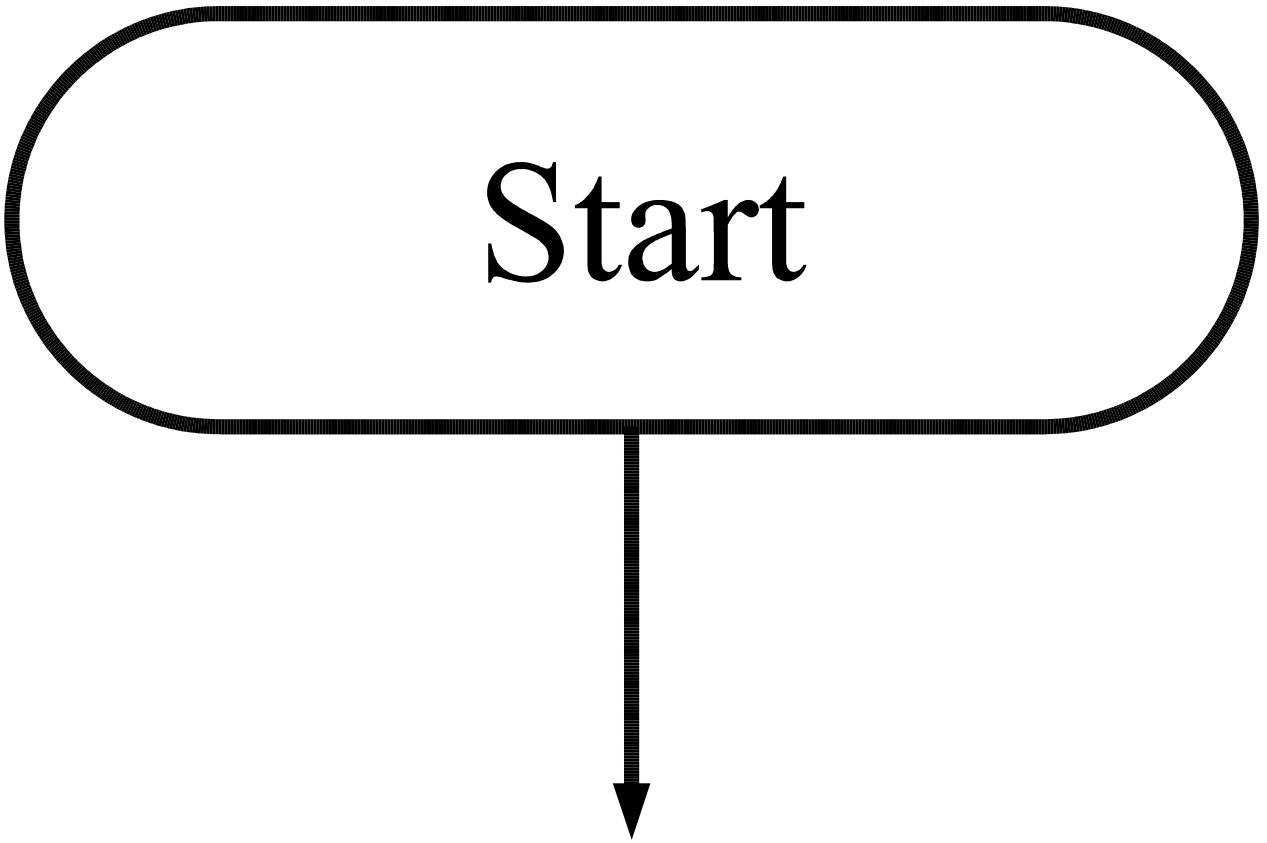
Flussdiagramme vs. Pseudocode

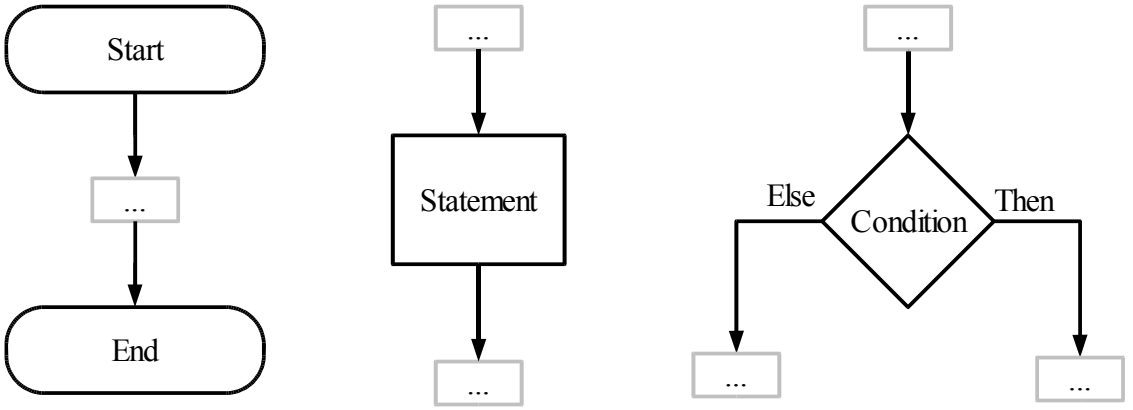
Seminarvortrag von Marc 'BlackJack' Rintsch



Worum geht's?

- Bringen Flussdiagramme Vorteile gegenüber
  - Pseudocode
  - echtem Quelltext
- Zwei Experimente
  - Scanlan (pro)
  - Shneiderman (contra)

Flussdiagramme (FD)



- Synonyme: *Flowchart*, *Programmablaufplan*
- grafische Notation zur Darstellung von Programmlogik

Flussdiagramme (FD)

- Anwendungen
  - Abläufe in ganz kleinen Schritten darstellen
  - gröberer Überblick über Abläufe
- nicht auf Programmabläufe beschränkt
- konkrete Anwendungen z.B.:
  - UML Aktivitätsdiagramme (activity diagram)
  - Lego Mindstorms Programmierung
  - „Patterson/Hennessy“

Pseudocode

- textuelle Notation zur Darstellung von Programmlogik
- nicht standardisiert
- Mischung aus „Pseudo-Sprache“ und Prosa
- manchmal auch mathematische Schreibweisen
- „Cormen“

Fortran 77

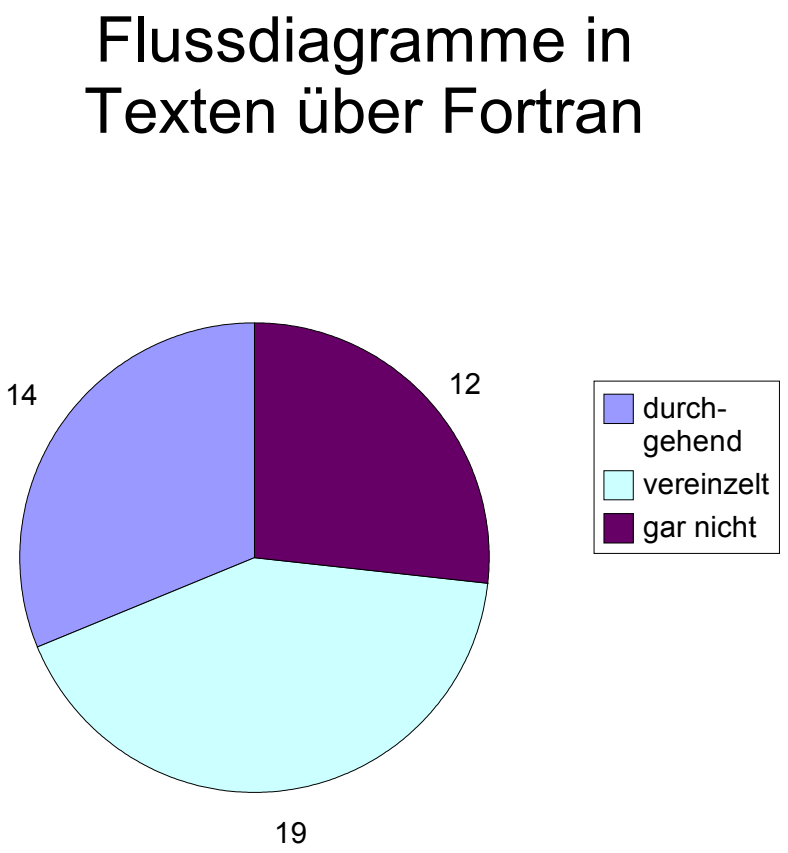
- Formular Translator
- imperative Programmiersprache
- ausgiebiger Gebrauch von Sprungmarken
  - damit unübersichlicher „Spaghetti“-Code möglich
- historisch bedingte Formatierungsanforderungen
  - die ersten 6 Zeichen jeder Zeile sind für (Sprung)Marken reserviert
  - max. Zeilenlänge = 72 Zeichen
  - Grossschreibung von Anweisungen und Bezeichnern

Shneiderman

„Experimental Investigations of the Utility of Detailed Flowcharts in Programming“

Ben Shneiderman, Richard Mayer, Don McKay und Peter Heller

Flussdiagramme in Texten



- Texte über Fortran
  - über ⅔ verwenden Flussdiagramme
- allgemeine Einführung in Programmierung
  - Flussdiagramme sind sprachunabhängig

FD ausserhalb der Informatik

- Anleitung zum Telefonieren
  - Gründe für Flussdiagramme
    - Hauptentscheidungskriterien werden an den Anfang verschoben
    - Komplexität des Textes wird reduziert
    - wichtige/unwichtige Information wird klar unterschieden
    - Beschränkung der Informationsmenge
- NASA
  - Langzeitlerneffekt bei Prosa höher

Ergebnisse für Informatik nutzbar?

- nicht direkt übertragbar
  - Programme sind in klar definieren Programmiersprachen verfasst
  - nur einfache Aufgaben untersucht
  - FD waren meistens auf 1 Seite beschränkt
  - Testpersonen bekamen entweder FD *oder* Text

Grundaufgaben beim Programmieren

- Grundaufgaben
  - Verständnis
  - Komposition / Entwurf
  - Fehlersuche
  - Modifikation
- Verständnis ist am wichtigsten
  - Voraussetzung für die anderen Aufgaben

Automatisch generierte Flussdiagramme

- Sollen
  - Programmverständnis erhöhen
  - Fehlersuche vereinfachen
  - Modifikation von fremden Programmen erleichtern
- Umstritten
  - zeigen nur Programmlogik
  - aber nicht die Zusammenhänge von Funktionen

Experiment I – „Entwurf“

- Hypothese
  - FD unterstützen den Entwurf von Programmen, da der Programmierer das Problem besser erfasst
- Testpersonen
  - Studenten aus einführendem Fortran-Kurs
  - Dozent und Buch benutzten Flussdiagramme
  - Experiment hat Einfluss auf Note des Kurses

Ablauf / Ergebnis

- zwei Gruppen
  - Gruppe 1: Flussdiagramm + Programm
  - Gruppe 2: nur Programm
- so viel Zeit wie erforderlich
- Problem mit vielen Verzweigungen gewählt
- Ergebnis
  - sehr gute Ergebnisse bei beiden Gruppen
  - Punkteunterschied von 1% statistisch unbedeutend

Experiment II – „Verständnis“

- Hypothese
  - Flussdiagramme erleichtern das Verständnis von vorhandenem (fremden) Quelltext
- Testpersonen
  - 53 Studenten aus Fortran-Einführungskurs
  - Experiment hat Einfluss auf Note des Kurses

Ablauf

- zwei Fortran-Programme (27 / 24 Zeilen)
  - komplexer als im ersten Experiment
- zwei Gruppen
  - Gruppe 1: Flussdiagramm zu Programm 1
  - Gruppe 2: Flussdiagramm zu Programm 2
- beide Gruppen bekamen beide Quelltexte
- Fragen zu Ausgaben bei gegebenen Eingaben und Kontrollfluss
- so viel Zeit wie erforderlich

Ergebnis

- Bewertung war einfacher da Antworten eindeutiger
- laut Varianzanalyse einzige fundierte Aussage: Programm 2 war schwieriger
- wieder trotz vielen Verzweigungen in Programmen
- Beobachtung:
  - Flussdiagramme nicht häufig zu Rate gezogen

Experiment III – „Fehlersuche“

- Hypothese
  - Flussdiagramme erleichtern die Fehlersuche, da sie die Logik des Programms übersichtlich darstellen
- Testpersonen
  - zwei Gruppen
    - 43 Studenten die keine FDs abgeben mussten (NFC-Gruppe)
    - 27 Studenten die zu jedem Programm FD erstellen mussten (FC-Gruppe)
  - Experiment hatte keinen Einfluss auf die Kursnote

Ablauf

- Tic-Tac-Toe in Fortran
  - 81 Zeilen Hauptprogramm (kommentiert)
  - 43 und 23 Zeilen Unterprogramme
  - 3 Fehler
- weitere Unterteilung der Gruppen
  - „mikro“-Flussdiagramm
  - „makro“-Flussdiagramm
  - kein Flussdiagramm

Ablauf

- 3 Schritte
  - Fehler finden / beseitigen
  - Fragen zum Programm
  - Fragen zur Selbstsicherheit bei Schritt 1 und 2

Ablauf – Schritt 1

- Ausgabe des Materials
  - Anweisungen
  - Programm + Ausgabe eines Programmlaufs
  - Flussdiagramm („mikro“, „makro“ oder „keins“)
- Informationen
  - es handelt sich um ein Tic-Tac-Toe Spiel, das nicht verlieren soll
  - es ist *mindestens* ein Fehler enthalten, der sich auch in der Ausgabe niederschlägt

Ablauf – Schritt 1

- Anweisung
  - studieren des Programms, wahlweise
    - Quelltext
    - Quelltext und Flussdiagramm
    - nur Flussdiagramm
  - den/die Fehler finden und beseitigen
    - Angabe der Zeilennummer
    - korrigierte Zeile
- Zeit NFC-Gruppe 40 Min., FC-Gruppe 50 Min.

Ablauf – Schritt 2

- 11 Multiple-Choice Fragen
  - Programmverständnis
  - grundlegende Programmierkenntnisse
  - Simulation des Programmablaufs
- Zeit
  - NFC-Gruppe 20 Minuten
  - FC-Gruppe 30 Minuten

Ablauf – Schritt 3 / Ergebnis

- Schritt 3
  - Beurteilung der Selbstsicherheit
    - Skala von 0-9
  - wie nützlich war das Flussdiagramm?
- Ergebnis
  - unterschiedliche Testbedingungen bei beiden Gruppen
  - deshalb nicht direkt vergleichbar
  - nur Trends innerhalb der Gruppen

Ergebnis

- Abschneiden der Untergruppen
  - NFC-Gruppe
    - ohne Flussdiagramm
    - „micro“
    - „makro“
  - FC-Gruppe
    - „mikro“
    - „makro“
    - ohne Flussdiagramm

Ergebnis

- Trends scheinen Zusammenhang zwischen Hintergrund (NFC/FC) und Nützlichkeit von Flussdiagrammen nahe zu legen
- statistisch nicht belegbar
- Frage nach Nützlichkeit
  - 1 Fehler gefunden – FD war hilfreich
  - 0, 2 oder 3 Fehler gefunden – FD war weniger hilfreich
- mögl. Erklärung
  - nur ein Fehler konnte mit Hilfe des FD gefunden werden

Experiment IV – „Modifikation“

- Hypothese
  - Die relevanten Stellen in einem Programm sind leichter zu finden, wenn man ein Flussdiagramm zur Verfügung hat, da dieses eine Übersicht über das Programm liefert
- Testpersonen
  - Studenten aus Programmierkursen im 2. Semester
  - „fortgeschrittene Anfänger“
  - wieder NFC / FC Gruppen
  - kein Einfluss auf Kursnote

Ablauf

- 75 Zeilen Fortran-Programm
  - 27 Kommentarzeilen
  - davon 23 als grosser Kommentarblock am Anfang
  - liest Studenten-Datensätze und gibt Notenbericht aus
- 3 Untergruppen („mikro“, „makro“, „ohne“)
- 45 Minuten Zeit für drei Modifikationsaufgaben
- Bewertung der Lösung
  - Korrekheit
  - Lauffähigkeit

Ergebnis

- 12 Studenten nicht berücksichtigt
  - 5 brachen Kurs ab
  - 7 hatten GPA ≤ 2.5
  - damit waren NFC- und FC-Gruppe gleich gross (30)
- Daten von dritter Modifikation nicht berücksichtigt, weil die Zeit nicht reichte

Ergebnis

- Bewertungskriterien
  - Formatierung der Ausgaben
  - Modifikationen an der falschen Stelle?
  - Korrektheit der Modifikation
  - Syntaxfehler
- Statistische Analyse stützt folgende Aussagen
  - Studenten mit FD machten weniger Fehler
  - zweite Modifikation war schwieriger als die erste
  - kein Unterschied wenn man nur Kriterium 2 betrachtet!

Selbstkritik

- Mögliche Gründe für fehlenden Unterschied
  - bei wesentlich grösseren Programmen hätten „makro“-FDs wahrscheinlich einen Vorteil, da sie eine Übersicht über das Programm bieten
  - ausführlicher Kommentarblock am Anfang war vielleicht ausreichend um Programm zu verstehen
- Studenten der FC-Gruppe waren mit Problem vertrauter
  - das Benotungsschema entsprach dem an ihrer Universität

Experiment V – „Verständnis“

- Frage
  - welche Informationen lassen sich aus FD herauslesen?
- Testpersonen
  - 58 Studenten aus 8-wöchigem Einführungskurs in die Informatik
  - „Quiz“ in der 6. Woche
  - kannten Fortran und Flussdiagramme

Ablauf

- Merge-Algorithmus in 23 Zeilen Fortran
- „Erstkontakt“
- 3 Untergruppen („mikro“, „makro“, „ohne“)
- 5 Fragen à 20 Punkte
  - 2 Fragen erforderten manuelle Simulation
  - 3 Fragen betrafen Ablauf und Eigenschaften des Algorithmus
- 25 Minuten Zeit

Ergebnis

- 7 Studenten wurden nicht berücksichtigt
  - 3 brachen den Kurs ab
  - 4 hatten eine Note schlechter als „D“
- Analyse der Varianz ergibt, das alle Gruppen gleichgut waren
- Informationsgehalt von FD und Pseudocode scheint äquivalent zu sein

Schlussfolgerung

- FD im wesentlichen redundante Darstellung von Information, die im Quelltext vorhanden ist
- sogar weniger Information
  - keine Deklarationen
  - keine (Sprung)Marken
  - keine Formatierungsinformationen
- dafür viel mehr Platzbedarf als kompakter Quelltext

Ausblick

- detaillierte FD könnten überflüssig werden durch
  - „top-down“ Entwicklungsansatz
  - „strukturierte“ Sprachelemente
- „makro“-FD könnten dagegen interessant sein
  - bei grossen, komplexen Programmen (> 1000 Zeilen)
  - für „Nicht-Programmierer“, z.B. Manager
- weitere Experimente mit erfahrenen Programmierern und grossen Programmen wären wünschenswert

Scanlan

„Structured Flowcharts Outperform Pseudocode: An Experimental Comparison“

David A. Scanlan

Flussdiagramm Befürworter

- mehrere Studien die grafische Darstellungsformen als hilfreich befunden haben
  - je komplexer das Problem umso nützlicher sind Bildchen
  - syntaxabhängiges Einrücken von Quelltext ist auch eine Art „räumlicher“ Darstellung von Programmlogik

Kritik an Shneiderman & Co

- vorhergehende Experimente sind fehlerhaft
- Kritik an Shneiderman Experiment II
  - Testpersonen hatten soviel Zeit wie sie benötigten
  - 5 Verständnisfragen waren Fortran-spezifisch
  - Algorithmen zu einfach
- Zeit muss eine Messgrösse sein

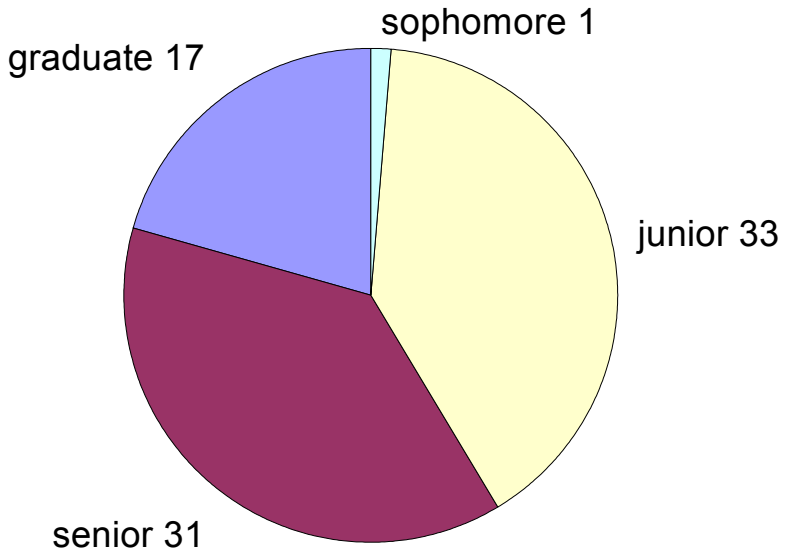
Hypothesen

- Flussdiagramme
  - sind schneller zu Verstehen
  - ergeben weniger Fehler beim Verstehen
  - geben mehr Selbstvertrauen in das Verständnis von einen Algorithmus
  - verringern die Zeit für Antworten auf Fragen
  - verringern die Zahl der Blicke auf einen Algorithmus
- 3 Algorithmen (einfach, mittel, komplex)
- $5 \cdot 3 = 15$ Hypothesen

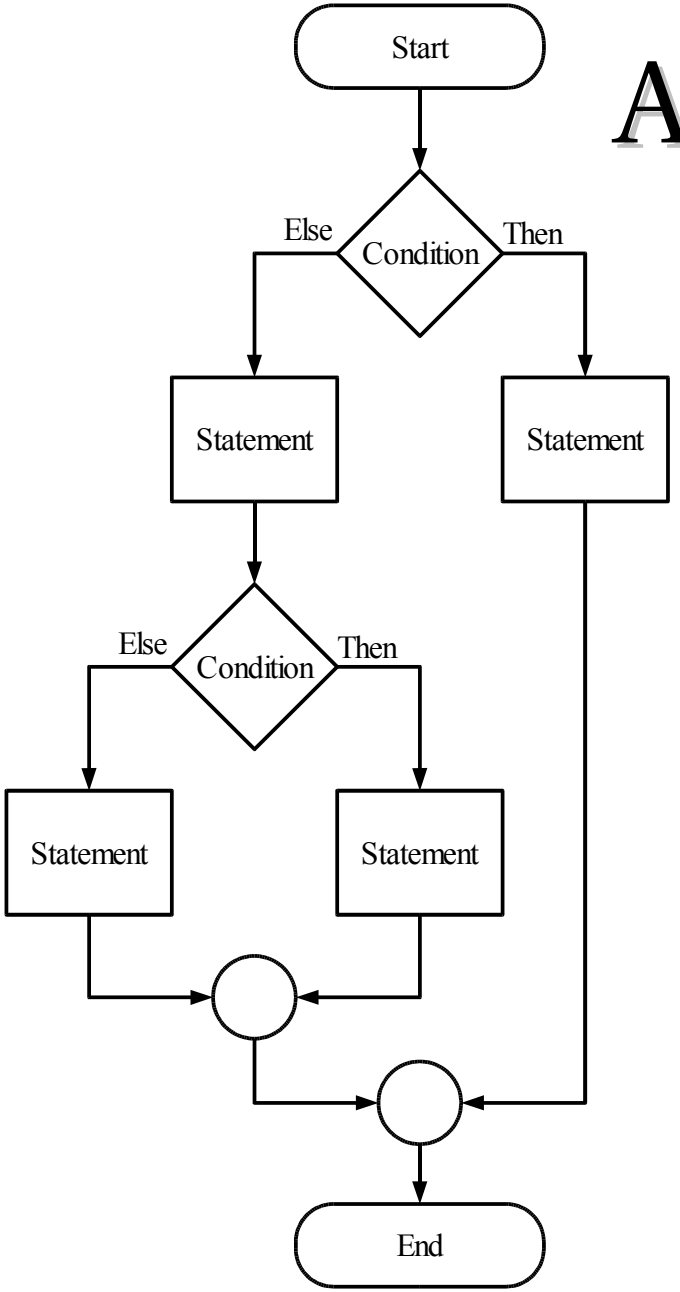
Testpersonen

- 82 *Management Information Science* Studenten
- 48 ♂ / 34 ♀
- Veranstaltungen
  - Pascal
  - Cobol
  - Datenstrukturen
  - Softwaretechnik

Testpersonen - Daten



- Teilnehmer im ∅
  - 27.8 Jahre alt
  - 3.35 Progr.-Sprachen
  - Informatik GPA 3.24



Algorithmen

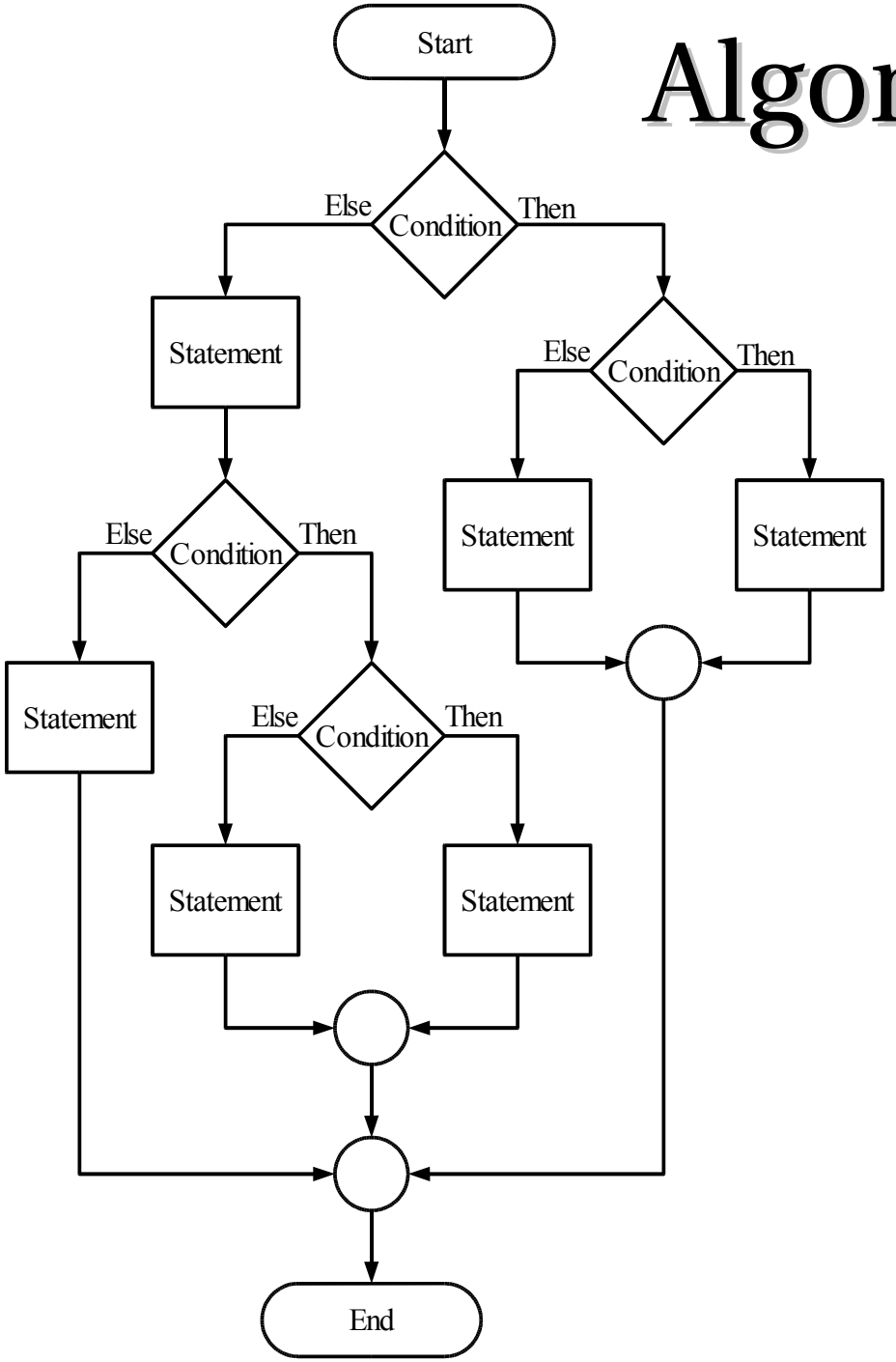
- 3 Algorithmen
  - einfach
  - mittel
  - komplex
- 2 Darstellungen
  - Flussdiagramm
  - Pseudocode

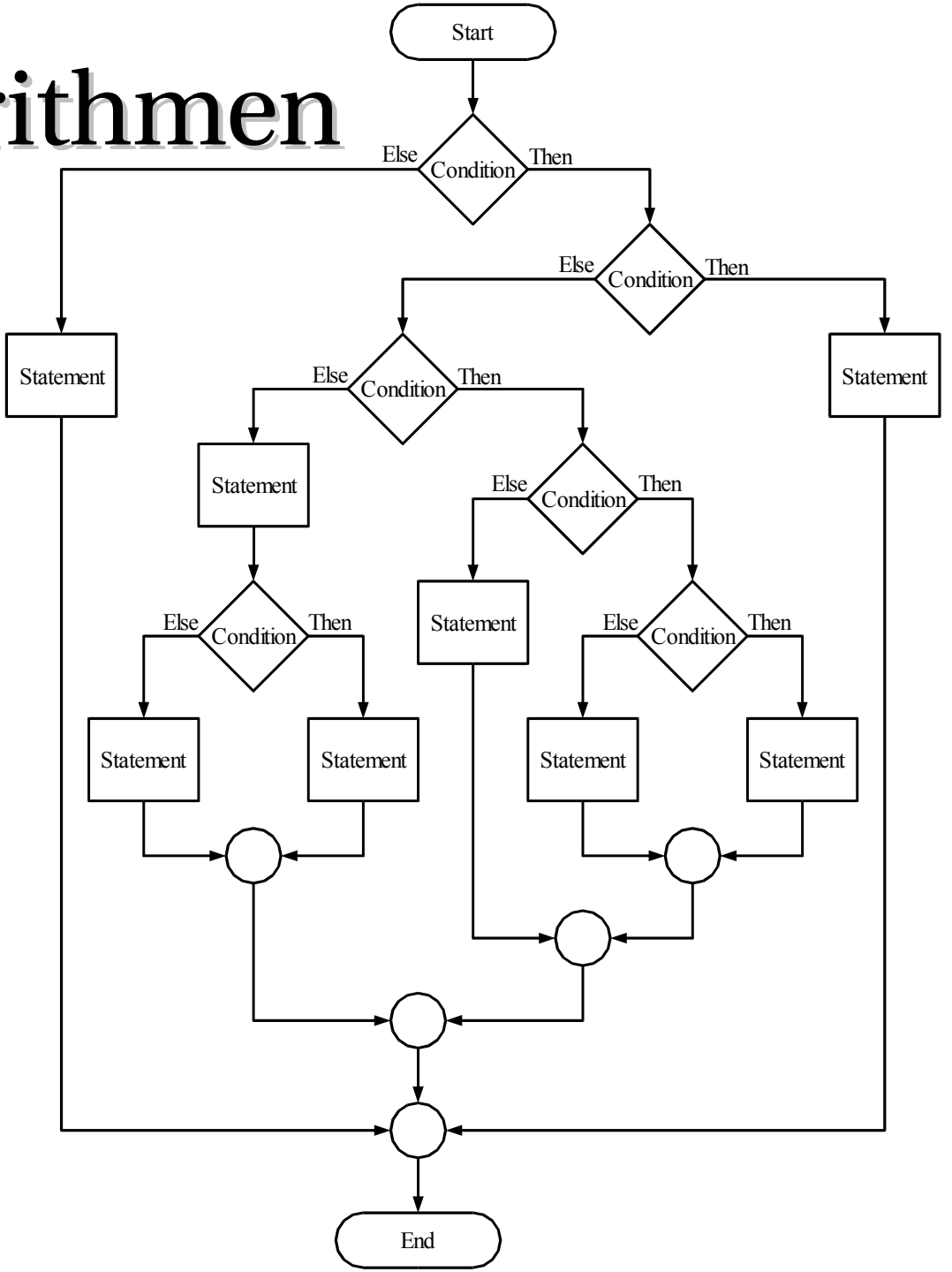
Algorithmen

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

- Bedingungen
  - Adjektive
- Anweisungen
  - Verben
- zufällig verteilt
- Programm ist „sinnlos“
- gleiche Schriftart

Algorithmen





Algorithmen - Zusammenstellung

- Verteilung von Adjektiven/Verben wurde 2× durchgeführt (Satz A und Satz B)
- 2 Sequenzen
  - 3 FD aus Satz A + 3 Pseudocodes aus Satz B
  - 3 FD aus Satz B + 3 Pseudocodes aus Satz A
- pro Student zufällig gewählt
  - Sequenz
  - Reihenfolge der Algorithmen

Material

- PC (inkl. Tastatur/Monitor)
- spezielles Anzeigegerät
- Sprachausgabegerät
- 12 Karten (8.5×11 inch)
- Fragebogen
- 6-seitige Beschreibung des Experiments
- 82 Disketten zum Erfassen der Testdaten
- 1 Assistent

Ablauf

- Informationen ein paar Tage vor Experiment
- am Tag des Experiments
  - Test über Informationspapier
  - Testdurchlauf
  - Pause
  - „echter“ Durchlauf
- während der beiden Durchläufe und der Pause konnten dem Assistenten Fragen gestellt werden

Aufgaben

- Algorithmus als
  - Flussdiagramm
  - Pseudocode
- Fragen
  - Kombination aus Verben
  - Status jeder Verzweigung angeben
    - wahr
    - falsch
    - unbekannt

Messgrössen

- Zeit, die der Algorithmus angeschaut wurde
- Prozent der korrekt beanworteten Fragen
- Selbstsicherheit der Testperson bei jeder Antwort
- Zeit, die zum Lesen und Beantworten der Fragen benötigt wurde
- Anzahl der Blicke auf den Algorithmus

Ergebnisse

- Messergebnisse zeigten bei Flussdiagrammen
  - weniger Fehler
  - höheres Vertrauen in die Antworten
  - weniger Zeit
  - weniger Blicke auf den Algorithmus
- Effekt ist umso deutlicher, je komplexer der Algorithmus ist
- Scanlan sieht seine Hypothesen bestätigt

Beurteilung – Shneiderman

- Experiment I
  - sagt wenig über die Nützlichkeit von FD aus, da Problem viel zu trivial war
    - Shneiderman bemerkt selber, dass die Teilnehmer alle sehr gut abgeschnitten haben
    - im Schnitt 94% bzw. 95% der Punkte
  - zum Vergleich
    - das komplexere Programm aus Experiment II hatte 27 Zeilen

Beurteilung – Shneiderman

- Experiment III
  - Teilnehmer durften auch ausschliesslich das FD studieren (laut Anweisung)
  - die Aufgabe und einige Fragen waren aber ohne den Quelltext nicht lösbar
    - Nachteil für Studenten, die sich nur das FD angeschaut haben

Beurteilung – Shneiderman

- Experiment IV & V
  - Ausschluss von „schlechten“ Studenten nur bedingt nachvollziehbar
    - Abbrecher müssen nicht „schlecht“ gewesen sein
    - „schlechter“ GPA kann auch Kurswiederholung mit mittelmässiger Note bedeuten

Was sollte untersucht werden?

- Äquivalenz von verwendeten FD und Fortran-Programmen einsehbar
- wie sähe es aus wenn Kontrollfluss nicht so einfach aus Quelltext ersichtlich wäre
  - Algorithmen über mehrere Objekte verteilt
  - Kontrollflüsse über mehrere Systeme verteilt
  - Algorithmen zur Laufzeit dynamisch zusammengesetzt

Beurteilung - Scanlan

- Pseudocode leserunfreundlich formatiert
  - Monospace
  - durchgehend Grossbuchstaben
  - in etwa gleichlange Schlüsselworte / Adjektive / Verben
  - Einrücktiefe nur 2 Leerzeichen
- Algorithmen scheinen praxisfremd
  - einfache, lineare Entscheidungsbäume
  - völlig sinnfrei

Beurteilung – Scanlan

- Experiment hat nicht „Verständnis“ gemessen, sondern wie gut jemand den Algoritmus auswendig lernen und „herunterbeten“ kann
  - Scanlan fragt „was“ passiert, nicht „warum“ passiert etwas
  - wegen „Sinnfreiheit“ der Algorithmen auch gar nicht anders möglich
- „Je komplexer der Algorithmus umso nützlicher sind FD“
  - es gibt sicher eine Obergrenze wegen Platzbedarf