Algorithmen, Objekte und Datenstrukturen

2 Grundlagen der Programmierung

2.1 Einführung in das Programmieren

Dieser Abschnitt vermittelt die allgemeinen Schritte des Programmierens und die Bedeutung der abstrakten Beschreibung eines Algorithmus vor jeder Umsetzung mittels Programmiersprache.

Es ist seit jeher spannend, Apparate, Maschinen und vor allem Computer so zu programmieren, dass sie eigenständig Arbeit verrichten können. Man merkt relativ schnell, dass es gar nicht so einfach ist, Abläufe in eine maschinenverständliche Form zu bringen. Probleme, die ein Mensch intuitiv oder durch Schlussfolgerungen lösen kann, müssen für eine Maschine immer in einer mit endlich vielen, genau definierten Einzelschritten festgelegten Handlungsvorschrift angegeben werden. Diese Handlungsvorschrift nennt man **Algorithmus**.

*Ein **Algorithmus** ist eine genau definierte Abfolge von endlich vielen, wohlformulierten Einzelschritten, die bei denselben Voraussetzungen immer das gleiche Ergebnis liefert. Der nächste Schritt muss zu jeder Zeit eindeutig definiert sein.*

Wie bereits in Band 1 (→ Geschichte der Informatik) erwähnt wurde, erfand Ada Lovelace 1842 für die *Analytical Engine* von Charles Babagge den ersten „Computer"-Algorithmus. Sie gilt seither als die erste Programmiererin. Da die *Analytical Engine* nie fertiggestellt wurde, konnte auch der dafür entwickelte Algorithmus nie implementiert werden. Besser gesagt: Der Algorithmus wurde nie in maschinenverständlicher Form durchgeführt.

Wie man dieser Geschichte bereits entnehmen kann, setzt sich die Aufgabe, Maschinenprozesse automatisch ablaufen zu lassen, aus zwei Schritten zusammen: Zuerst aus einer abstrakten Beschreibung der Handlung (Algorithmus) und erst, nachdem dieser Schritt abgeschlossen wurde, aus der Übersetzung des Algorithmus in eine maschinenlesbare Form (Programm). Vereinfacht nennt man den zweiten Teilschritt auch **Codierung**.

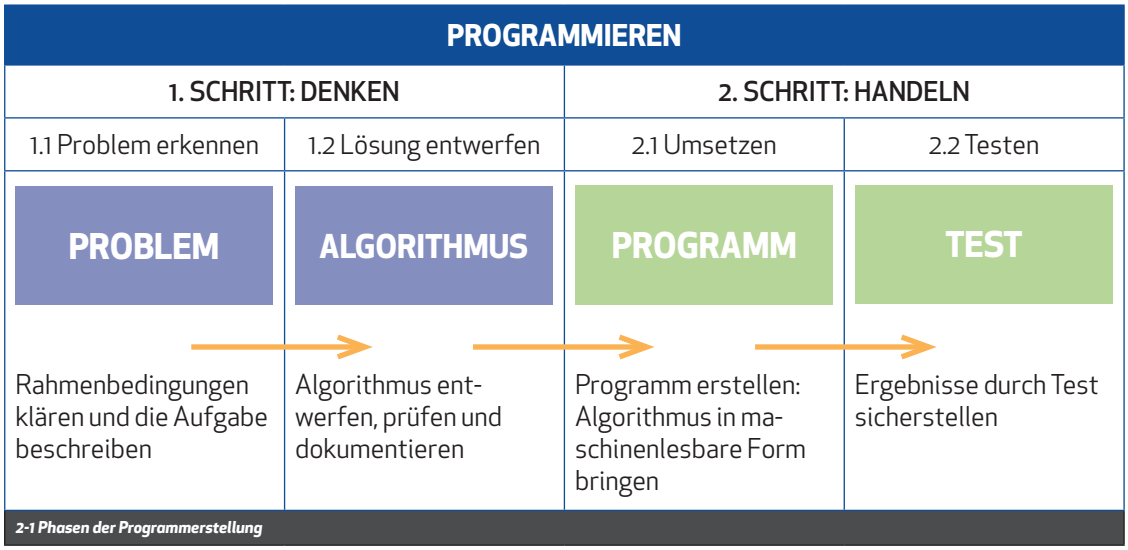
*Den Vorgang, ein auf einem Computer lauffähiges Programm zu erstellen, nennt man **Programmieren**. Es setzt sich grundsätzlich aus zwei voneinander abhängigen Schritten zusammen:*

- ***Schritt 1**: Entwurf des Algorithmus*
- ***Schritt 2**: Implementierung des Algorithmus mittels Programmiersprache*

Schritt 1: In Schritt 1 überlegt man sich zu einem bestimmten Anwendungsproblem einen Algorithmus, also eine eindeutige Handlungsvorschrift, um ein konkretes Anwendungsproblem zu lösen. Dieser Algorithmus ist eine abstrakte Beschreibung dessen, was gemacht werden soll. Die technische Realisierung wird noch nicht miteinbezogen. Leider gibt es keine zwingende Darstellungsform für Algorithmen, daher obliegt es der Erstellerin bzw. dem Ersteller, wie diese Handlungsvorschrift dokumentiert wird. Möglichkeiten sind Struktogramme, Ablaufdiagramme und Pseudocode.

Eine alte Programmierer-Weisheit besagt, dass eine klar formulierte Aufgabenstellung bereits die halbe Lösung darstellt. Daher gebührt gerade der Problemerkennungsphase große Sorgfalt, sonst werden Fehler der Phase 1.1 oft erst in einer Spätphase der Umsetzung erkannt und müssen dann mühsam korrigiert werden.

Neben der eigentlichen Erstellung eines Algorithmus als greifbares Ergebnis dieses Schrittes können in der Phase 1.2 auch schon Plausibilitätstests durchgeführt werden. Hier gilt es nämlich, Fehler tunlichst zu vermeiden, die man zu einem späteren Zeitpunkt nur noch mit großem Aufwand kompensieren kann.



2-1 Phasen der Programmerstellung

Schritt 2: Erst nach Abschluss des ersten Schrittes erfolgt eine Umsetzung des Algorithmus in einer spezifischen technisch-organisatorischen Lösung. Neben der eigentlichen Umsetzung mittels einer konkreten Programmiersprache (Codierung) müssen an dieser Stelle auch Plattform-Entscheidungen (HW-, SW-, Normen-Plattformen) für die Realisierung und das Betreiben getroffen werden. Außerdem müssen organisatorische Anforderungen berücksichtigt werden.

Erwartungsgemäß wird vor allem der Umsetzungsphase 2.1 die größte Aufmerksamkeit geschenkt. Besonders Neulingen erscheint dieses vermeintliche Programmieren als bezeichnende und alles entscheidende Tätigkeit. Die Realität deckt sich jedoch kaum mit der durch Film und Fernsehen vorgegaukelten Fiktion des Programmierprozesses. Der Reiz liegt mit Sicherheit darin, Algorithmen auf vorgegebenen Technologien umzusetzen und auftretende Fehler und Herausforderungen spontan zu lösen.

Die Testphase 2.2 dient neben der Überprüfung der geforderten Funktionen und Ergebnisse dazu, festgelegte Qualitätsansprüche sicherzustellen. Die Testergebnisse haben wiederum Auswirkungen auf die Umsetzungsphase. Ein Programm ist erst dann abgeschlossen, wenn die Testphase erfolgreich beendet werden kann.

So einfach diese Phasen des Programmierens erscheinen, so schwierig sind sie in der Umsetzung. Was im Unterricht noch als Schikane empfunden wird – man möchte doch gleich eine Programmiersprache anwenden! – wird in der beruflichen Praxis oft durch umfangreiche Management-Werkzeuge zur Unterstützung der Anwendungsentwicklungsprozesse massiv eingefordert. Nur so ist eine professionelle Softwarequalität sicherzustellen.

2.2 Methoden zur Erstellung von Algorithmen

In diesem Kapitel werden Methoden zur Erstellung von Algorithmen vorgestellt. Nach der Bearbeitung dieses Kapitels kennt man unterschiedliche Methoden zur Darstellung und kann die Vor- und Nachteile einzelner Methoden benennen und abwägen.

Um möglichst schnell ein wartungsfreundliches Programm zu erstellen, ist es notwendig, die gewünschte Programmarchitektur vor der Codierung mit einer Programmiersprache zu konstruieren. Dies ist ein Vorgang, der in anderen Ingenieurwissenschaften durchaus üblich ist. Von Informatikerinnen und Informatikern wird dieses Prozedere oft vernachlässigt. Folgen dieser nachlässigen Vorgehensweise sind meist zu lange Programmierzeiten und komplizierte Strukturen, die im Zuge von *Trial & Error* zustande kommen.

Das Geheimnis der erfolgreichen Anwendung von Methoden zur Erstellung von Algorithmen liegt darin, dass Programmiererinnen und Programmierer sich schrittweise der Problemlösung nähern. Darüber hinaus übt man so, den richtigen Detaillierungsgrad für die Beschreibung zu finden.

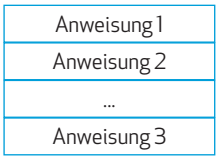
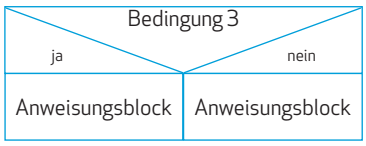
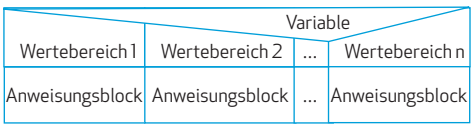
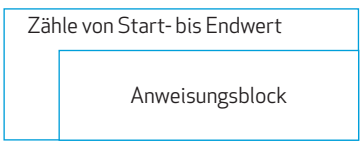
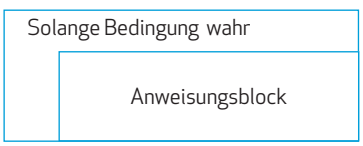
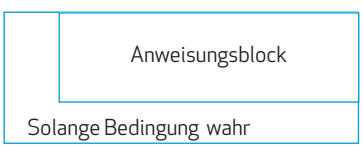
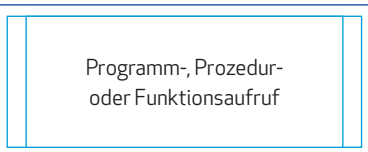
Die hier zur Sprache kommenden Nassi-Shneiderman-Diagramme (Struktogramme), Programmablaufpläne (PAP) und Verbalbeschreibungen (auch Pseudocode) verstehen sich als einfach zu handhabende Werkzeuge, die in der Annäherung an das Programmieren ausgezeichnete Dienste leisten und mit Papier und Bleistift durchgeführt werden können. Sie bestehen aus unterschiedlichen Darstellungselementen: einerseits aus datenmanipulierenden Anweisungen bzw. Anweisungssequenzen und andererseits aus programmablaufstrukturierenden Elementen.

*Um einen **Algorithmus** eindeutig darzustellen, bedarf es unterschiedlicher Beschreibungselemente, welche aus datenmanipulierenden **Anweisungen** und **Kontrollstrukturen** bestehen. Die Kontrollstrukturen stellen sicher, dass Anweisungen in richtiger Abhängigkeit durchgeführt werden.*

Darüber hinaus gibt es teure Software-Modellierungsprodukte, deren Handhabung in den Anfängen der Programmierung jedoch mehr hinderlich als nützlich ist. Ein für unsere Anforderungen nicht zutreffender Kritikpunkt an der Auswahl ist, dass diese Beschreibungsmethoden zur Zeit der imperativen Sprachen (Befehlsorientierte Sprachen → Kapitel 2.4.1) entstanden sind und diese unterstützen. Die heute üblichen objektorientierten Sprachen mit ihren durchaus komplexen Möglichkeiten können aber nicht in vollem Funktionsumfang davon profitieren.

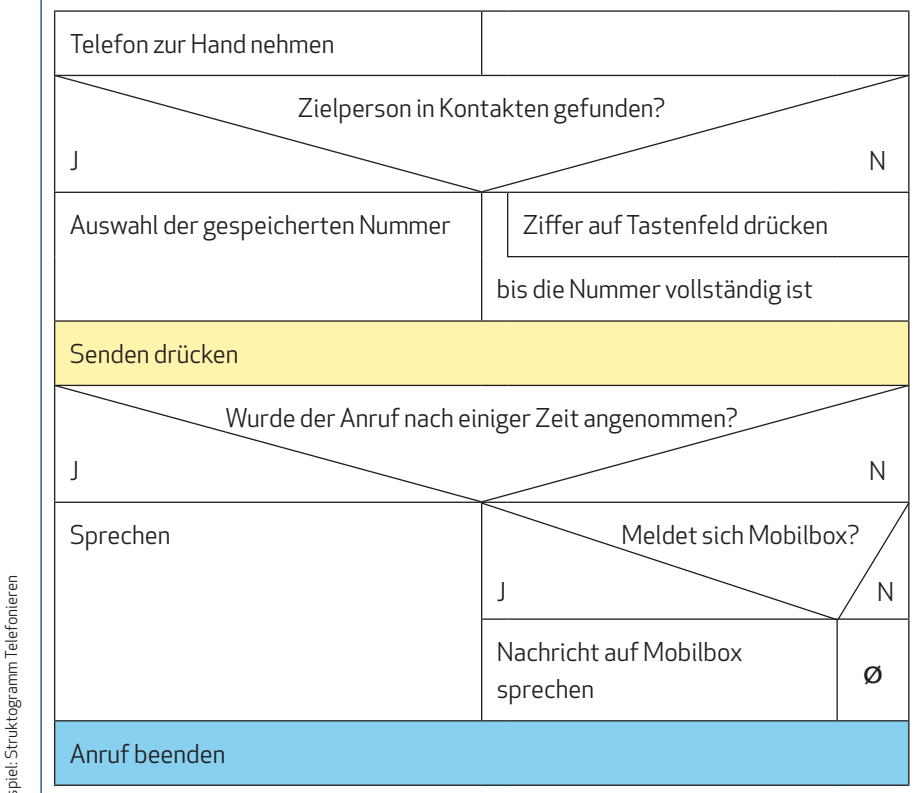
2.2.1 Nassi-Shneiderman-Diagramm

Eine strukturierte und einfache Methode stellen die Nassi-Shneiderman-Diagramme (DIN 66261) dar, die auch Struktogramme genannt werden. Mit dieser grafischen Methode kann man im ersten Schritt einfach eine geeignete Programmstruktur beschreiben

Diagrammart	Name und Beschreibung
	***Sequenz:*** Die Sequenz ist eine Abfolge von Anweisungen, die sequenziell (hintereinander) durchgeführt werden. Sie sind die eigentlich datenmanipulierenden Anweisungen.
	***Auswahl (if-then-else):*** Die Auswahl erfolgt mittels Bedingung (**if**). Bei Zustimmung (**then**) wird der eine Anweisungsblock verarbeitet, bei Abweisung (**else**) der andere.
	***Fallauswahl*** ist eine inhaltliche Auswahl (**select**) mit mehreren Möglichkeiten (**case**). Bei Übereinstimmung wird der jeweilige Block durchgeführt. Wenn keine Übereinstimmung (**else**) eintritt, kann ein eigener Anweisungsblock durchgeführt werden.
	***Zählergesteuerte Schleife:*** Ein Anweisungsblock wird so lange durchgeführt, bis eine Zählvariable (ausgehend vom Startwert) den Endwert erreicht. Jeder Durchlauf erhöht die Zählvariable – die Erhöhung muss nicht in Einserschritten erfolgen, es kann jede beliebige sinnvolle Zahl sein.
	***Abweisende (vorprüfende, kopfgesteuerte) Schleife:*** Ein Anweisungsblock wird durchgeführt, solange eine Bedingung erfüllt ist. Die Prüfung erfolgt zu Beginn des Blocks.
	***Nicht abweisende (nachprüfende, fußgesteuerte) Schleife:*** Ein Anweisungsblock wird durchgeführt, solange eine Bedingung erfüllt ist. Die Prüfung erfolgt am Ende des Blocks.
	***Aufruf:*** Dieses Element stellt den Aufruf eines Unterprogramms, einer Prozedur, Funktion oder Methode dar.

Diese kompakte Diagrammform kann beliebig zusammengesetzt und verschachtelt werden. Da es sich um eine grafische Methode handelt, erkennt man schnell, ob ein allfälliger Programmaufbau zu kompliziert ist oder nicht. Darüber hinaus kann man im Sinne einer strukturierten Programmierung gänzlich auf unbedingte Sprünge verzichten.

Strukturprogramm für den Vorgang "Telefonieren"



Beispiel: Struktogramm Telefonieren

Anhand des folgenden Beispiels wird nun die Funktionsweise eines Struktogramms dargestellt. Es zeigt den einfachen Vorgang des Telefonierens. Wie bereits erwähnt ist es nicht immer so einfach, selbst so simple Abläufe wie das Telefonieren in eine maschinenverständliche Form zu bringen. Die Kunst ist es, eine Beschreibungstiefe zu finden, die gerade dazu ausreicht, den Vorgang eindeutig zu beschreiben.

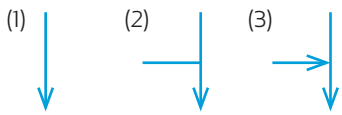
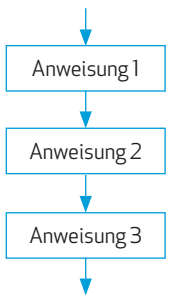
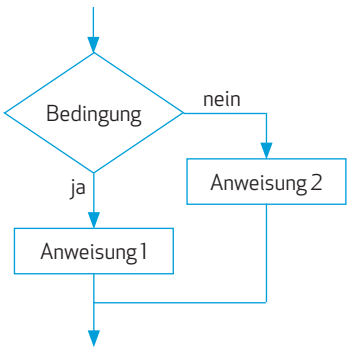
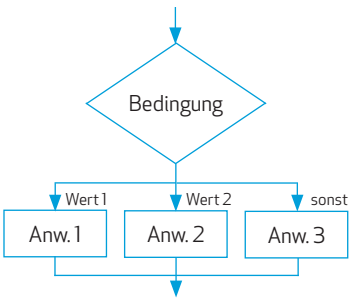
In unserem Fall beginnen wir mit einer einfachen *Anweisung*: Telefon zur Hand zu nehmen. Mittels *Auswahl* überprüfen wir das Vorhandensein der benötigten Telefonnummer im Nummernspeicher des Telefons. Ist sie vorhanden (J), dann kann sie ausgewählt werden. Ist sie nicht gespeichert (N), muss die Nummer mit der Hand eingegeben werden, Ziffer für Ziffer, in *fußgesteuerter Schleife*. Danach erfolgt die *Anweisung* „Senden drücken". Eine weitere *Auswahl* überprüft den Erfolg des Anrufs. Wenn er angenommen wird (J), dann kann man sprechen. Wenn nicht (N), meldet sich vielleicht eine Mobilbox – das wird mit einer verschachtelten *Auswahl* überprüft. Wird der Anruf angenommen, dann setzt das Gespräch ein. Wird der Anruf nicht angenommen, wird nichts gemacht. Alle drei Varianten schließen mit dem Beenden des Anrufs.

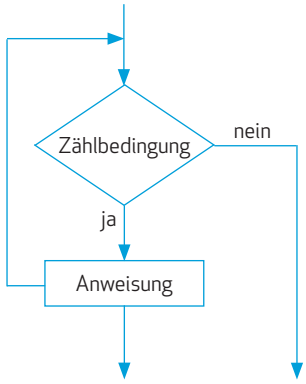
Zusammenfassend kann man feststellen, dass Struktogramme eine programmier- oder besser gesagt codiernahe Methode sind. Die Darstellungsvorschriften verhindern unnötige (wenn auch bequeme) Sprünge durch ein Programm und zeigen unerbittlich zu große Verschachtelungstiefen auf (dann, wenn der Graph unübersichtlich wird).

2.2.2 Programmablaufplan (PAP)

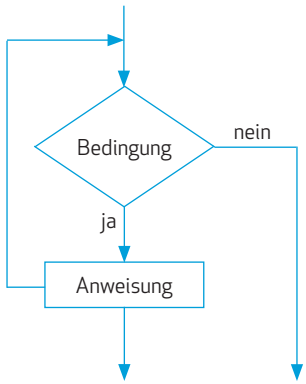
Eine andere Form der Darstellung ist der langgediente Programmablaufplan, kurz PAP. Er dient nicht nur zur Darstellung von Programmalgorithmen, sondern kann auch für beliebige Handlungsvorschriften (z. B. Bedienungsanleitungen, Prozessdarstellung usw.) als Flussdiagramm verwendet werden. Die Symbole sind seit 1977 in der DIN 66001 genormt. Im Lauf der Zeit haben sich einige Elemente aufgrund des Technologiewandels geändert.

Die folgende Tabelle stellt die wichtigste Auswahl von Elementen eines PAPs dar. Die Elemente werden in Analogie zu den Nassi-Shneiderman-Elementen erklärt:

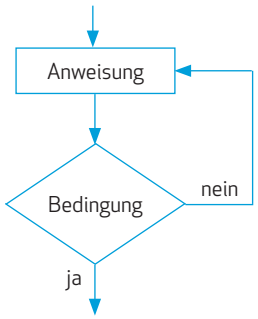
Diagrammart	Name – Beschreibung
	***Grenzstelle:*** Die Grenzstelle symbolisiert den Beginn oder das Ende eines PAPs. Bei Unterprogrammen kann in den Startknoten der jeweilige Name des Unterprogramms eingetragen werden.
	***Ablauflinien:*** Ablauflinien legen die Ablaufrichtung fest (1). Mehrere Linien können zusammengeführt werden (2); das auch mit Richtungsvorgabe (3).
	***Anweisung/Sequenz:*** Die Sequenz ist eine Abfolge von Anweisungen, die sequenziell (hintereinander) durchgeführt werden. Es handelt sich um datenmanipulierende Anweisungen.
	***Auswahl (if-then-else):*** Die Raute symbolisiert die Auswahl mittels Bedingung (**if**). Bei Zustimmung (**then**) wird der eine Anweisungsblock verarbeitet, bei Abweisung (**else**) der andere.
	***Fallauswahl:*** Die Raute stellt eine inhaltliche Auswahl (**select**) mit mehreren Möglichkeiten (**case**) dar. Die jeweilige Auswahl wird durch Wert 1, ..., Wert n und sonst dargestellt. Bei Übereinstimmung wird der jeweilige Block durchgeführt. Wenn keine Übereinstimmung (**else**) eintritt, dann kann ein eigener Anweisungsblock durchgeführt werden.



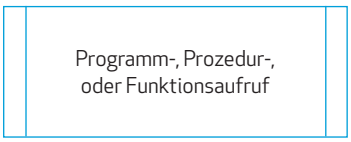
Zählergesteuerte Schleife:
Ein Anweisungsblock wird so lange durchgeführt, bis eine Zählvariable (ausgehend vom Startwert) den Endwert erreicht. Jeder Loopback erhöht die Zählvariable – die Erhöhung muss nicht in Einserschritten erfolgen, es kann jede beliebige, sinnvolle Zahl sein.



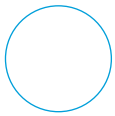
Abweisende (vorprüfende, kopfgesteuerte) Schleife:
Ein Anweisungsblock wird durchgeführt, solange (**while**) eine Bedingung erfüllt ist. Die Prüfung erfolgt zu Beginn des Blocks.



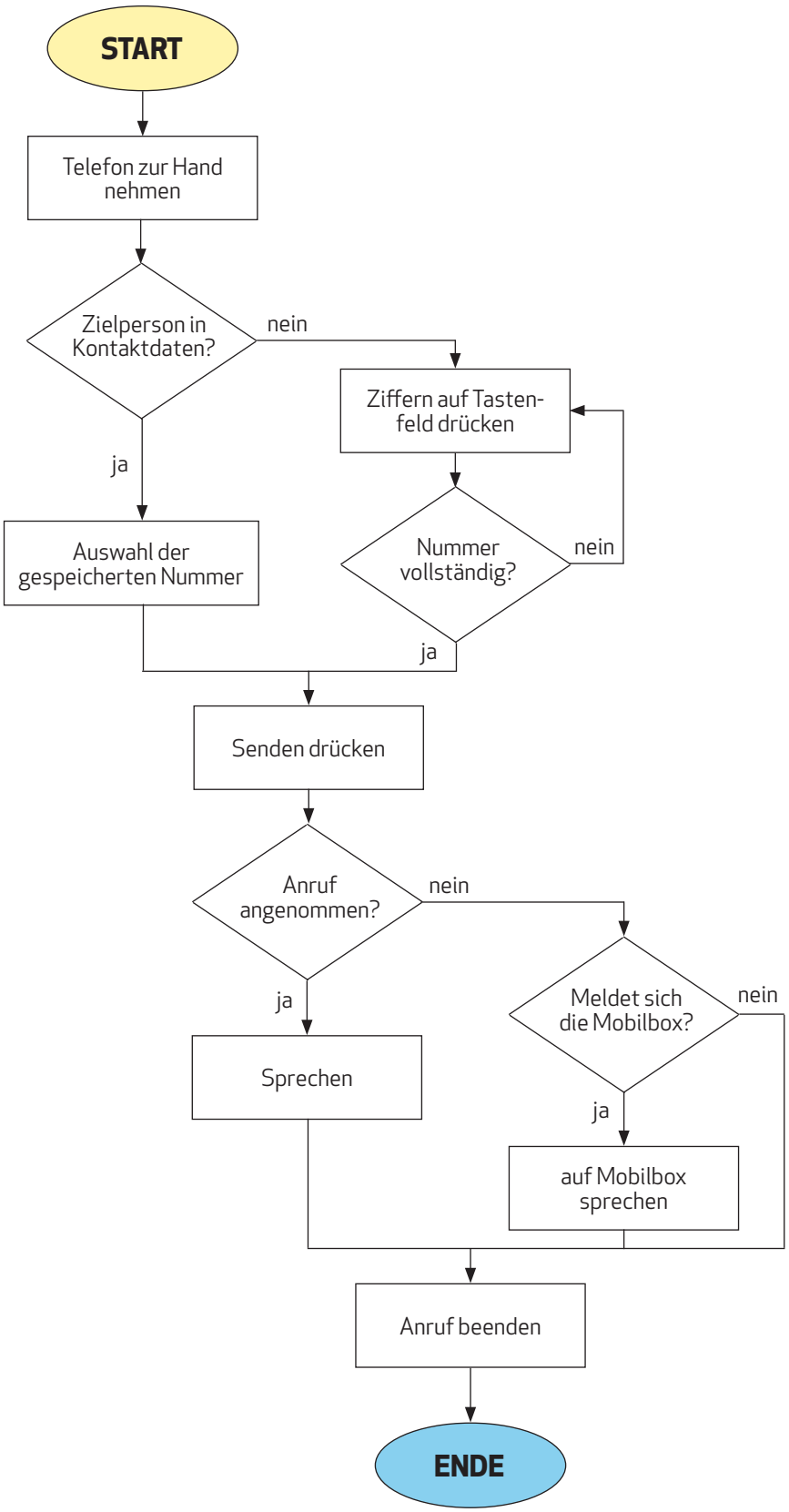
Nicht abweisende (nachprüfende, fußgesteuerte) Schleife:
Ein Anweisungsblock wird durchgeführt (**do-while**), solange eine Bedingung erfüllt ist. Die Prüfung erfolgt am Ende des Blocks.



Aufruf:
Dieses Element stellt einen Aufruf eines Unterprogramms, einer Prozedur, Funktion oder Methode dar.



Übergangsstelle:
Sie dient der Übersichtlichkeit und ermöglicht es, einen Prozess an einer Stelle zu unterbrechen, um ihn an einer anderen Stelle, z. B. auf der nächsten Seite, fortzuführen.



Beispiel: PAP – Telefonieren

Wie schon bei den Nassi-Shneiderman-Diagrammen können hier alle Elemente beliebig kombiniert und verschachtelt werden. Aufgrund der Darstellungsgröße und den für die Übersichtlichkeit notwendigen (Pfeil-) Abständen kann eine derartige Darstellung relativ umfangreich und unübersichtlich ausfallen. Darüber hinaus verleiten die einfach zu zeichnenden Pfeilverbindungen dazu, Komponenten beliebig miteinander zu verbinden. Das kann zur Zerstörung einer Programmstruktur führen.

Während die Nassi-Shneiderman-Diagramme gerade diesen Umstand und ein Wachstum in die Breite (Verschachtelung) zu verhindern wissen, muss man diesbezüglich bei einem PAP selbstständig diszipliniert arbeiten.

Bleiben wir nun beim Beispiel des Telefonierens. Am Algorithmus hat sich zum vorherigen Beispiel nichts geändert, es wurde nur die Darstellungsform gewechselt. Wie man unschwer erkennen kann, treten bereits die oben geäußerten Nachteile dieses Flussdiagramms ein. Selbst so wenige sequentielle Arbeitsschritte verursachen schon einen recht langgezogenen Graphen.

Trotz aller Nachteile kann man mit einem PAP auch komplexe Abläufe anschaulich darstellen. Die universell anwendbare Flussdiagrammtechnik erlaubt darüber hinaus, dass ein PAP für beliebige Anwendungen zum Einsatz kommen kann; auch im Sinne eines einfachen Projektplans. Beispiele sind unter anderem Prozessabläufe, die in diesem Format auch Zeitabschätzungen durch Gewichtungen der Pfeile ermöglichen.

2.2.3 Verbalbeschreibung bis Pseudocode

Eine beliebte und einfache Art der Ablaufdarstellung ist die Verbalbeschreibung. Statt mit Symbolen wird hier mit einfachem Freitext die Handlungsvorschrift beschrieben. Nachdem diese Form nicht genormt ist, muss man jedoch auf eindeutige und wiederkehrende Ausdrücke achten.

Gleichrangige Befehle werden auf gleicher Höhe, untergeordnete Befehle eingerückt geschrieben. Die folgende Aufstellung von möglichen Schreibvarianten wird mittels der Nassi-Shneiderman-Elemente dargestellt:

Diagrammart	Name – Beschreibung
Anweisung 1 Anweisung 2 Anweisung ...	***Anweisung/Sequenz:*** Die Sequenz ist eine Abfolge von Anweisungen, die sequenziell (hintereinander) durchgeführt werden.
wenn Bedingung dann Anweisung 1 Anweisung 2 *wenn* Bedingung dann Anweisung 1 sonst Anweisung 2	***Auswahl (if-then-else):*** Die Auswahl ***wenn*** erfolgt mittels Bedingung (***if***). Bei Zustimmung wird ***dann*** (***then***) der eine Anweisungsblock verarbeitet, bei Abweisung ***sonst*** (***else***) der andere.
für Variable Wert 1: Anweisung 1 Wert 2: Anweisung 2 Wert n: Anweisung n *sonst*: Anweisung n + 1	***Fallauswahl:*** Die Fallauswahl ist eine inhaltliche Auswahl (***select***) für eine Variable mit mehreren Möglichkeiten (***case***). Bei Übereinstimmung mit den einzelnen Werten wird der jeweilige Block durchgeführt. Wenn keine Übereinstimmung, also sonst (***else***), eintritt, dann kann ein eigener Anweisungsblock durchgeführt werden.

Diagrammart	Name – Beschreibung
von Zähler = Startwert *bis* Endwert Anweisung 1 Anweisung 2	***Zählergesteuerte Schleife:*** Ein Anweisungsblock wird so lange durchgeführt, bis eine Zählvariable (ausgehend vom Startwert) den Endwert erreicht.
solange Bedingung Anweisung 1 Anweisung 2	***Abweisende (vorprüfende, kopfgesteuerte) Schleife:*** Ein Anweisungsblock wird durchgeführt, solange eine Bedingung erfüllt ist. Die Prüfung erfolgt zu Beginn des Blocks, daher muss der Anweisungsblock nicht zwingend durchlaufen werden.
wiederhole Anweisung 1 Anweisung 2 *bis* Bedingung	***Nicht abweisende (nachprüfende, fußgesteuerte) Schleife:*** Ein Anweisungsblock wird wiederholt, bis die Bedingung erfüllt ist. Die Prüfung erfolgt am Ende des Blocks, daher wird der Anweisungsblock mindestens einmal durchgeführt.
Prozedur_A: Anweisung 1 Anweisung 2	***Aufruf:*** Dieses Element stellt ein gesondertes Unterprogramm, eine Prozedur, Funktion oder Methode dar. Der Aufruf erfolgt im Hauptprogramm in einem Anweisungsblock durch Anführung des Namens (z. B. Prozedur_A).

Die Verbalbeschreibung ist eine beliebte Darstellungsform, da keinerlei Formalvorschriften erforderlich sind. Ähnlich den Struktogrammen kann man einen Sachverhalt kompakt beschreiben, jedoch führt allzu „freies" Formulieren zu Ungereimtheiten. Daher kommt die Verbalbeschreibung eher als individuelle Dokumentation zum Einsatz. Bei teamorientierten Arbeiten empfiehlt es sich, auf eine der zuvor beschriebenen Darstellungstechniken zurückzugreifen.

Beispiel Verbalbeschreibung Telefonieren

```
Telefon zur Hand nehmen
wenn „Zielperson in Kontaktdaten gefunden"?
        dann Auswahl der gespeicherten Nummer
        sonst wiederhole
                Ziffer auf Tastenfeld drücken
                bis die Nummer vollständig ist
Senden drücken
wenn „wird Anruf nach einiger Zeit angenommen"?
        dann Gespräch führen
        sonst wenn „meldet sich Mobilbox"
                dann auf Mobilbox Nachricht sprechen
                sonst keine Aktion
```

Nichtdestotrotz ist diese beliebte Technik als Grundlage jeglicher Überlegungen zu sehen und unbedingt **vor** jedem Programmier- bzw. Codieransatz zu empfehlen. Um die Lesbarkeit zu fördern, ist es angebracht, einheitliche Phrasen zu verwenden (vor allem bei den strukturierenden Elementen).

In Anwendung auf unser Beispiel „Telefonieren" ergibt sich eine kompakte Beschreibung, die bei Einhaltung der Einrück-Regel einen Überblick über die daraus resultierende Programmstruktur geben kann. Je stärker die Beschreibungssprache vereinheitlicht wird, desto leichter fällt es, komplexe Sachverhalte darzustellen.