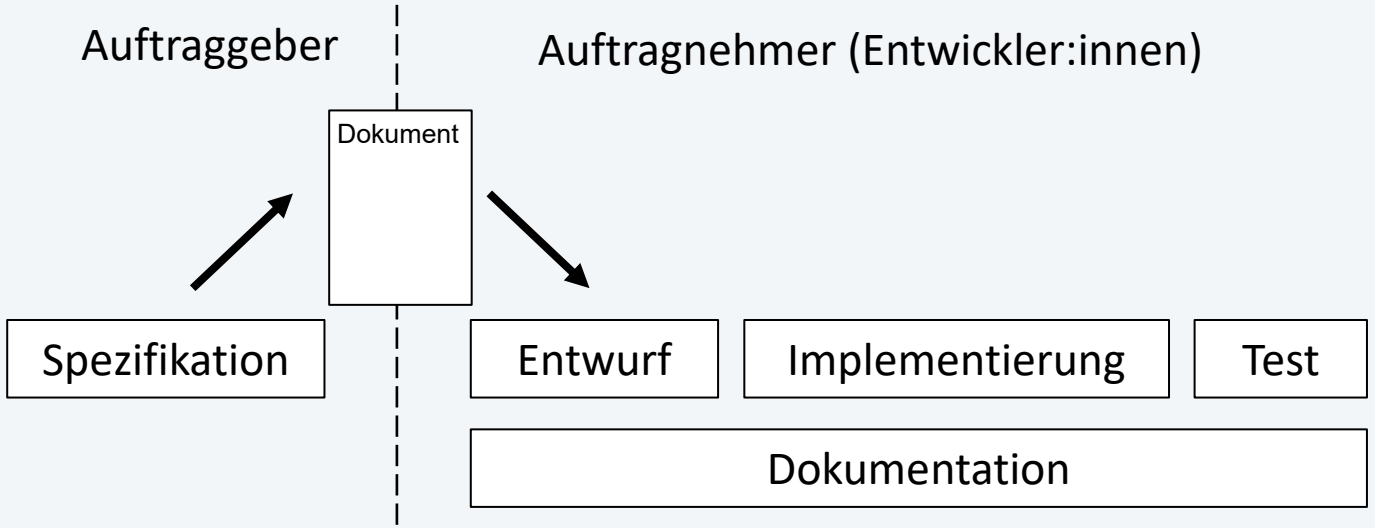
2 Spezifikation

Spezifikation im Entwicklungsprozess

Zweck der Spezifikation

- Was soll entwickelt werden?
- Worauf ist zu achten?
- Was kann vorausgesetzt werden?
- Was gilt als richtig, was als falsch?



2.1 Begriff

- Spezifikation beschreibt, was der Algorithmus leistet aber nicht wie er es leistet.
- Unter der Spezifikation eines Algorithmus verstehen wir die Beschreibung der Aufgabe(n), die durch den Algorithmus gelöst werden soll(en) und der Schnittstelle zu seiner Umgebung. (d.h. Woher kommen die Daten - mit read() oder durchs Netz?)
- Unterschiede zwischen Spezifikation und Algorithmus

Spezifikation	Algorithmus
Beschreibt das Problem und die Aufgabe, insbesondere die Schnittstelle	Ist die Lösung, also die Implementierung
Definiert das **Was**	Definiert das **Wie**
Außensicht (*black box view*)	Innensicht (*white box view*)

Anforderungen an eine gute Spezifikation (Beispiel: Übungszettel)

- Verständlich: soll so formuliert sein, dass beide Vertragspartner verstehen, worum es geht
- Eindeutig und widerspruchsfrei: keine Widersprüche, keine Mehrdeutigkeiten, konsistent, in sich geschlossen
- Vollständig und korrekt: keine Lücken, kein Interpretationsspielraum in wesentlichen Punkten, keine Fehler
- Minimal: so knapp wie möglich; kein „Geschwafel“, keine irrelevanten Details oder Vorwegnahmen der Implementierung

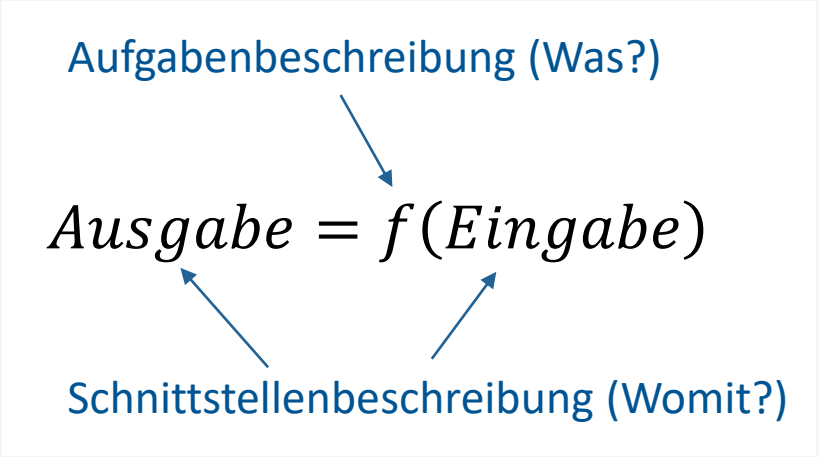
- Forderungen widersprechen einander teilweise
- Es ist oft schwierig, für eine Aufgabe eine angemessene Spezifikation zu formulieren, die alle oben angegebenen Anforderungen erfüllt

Anforderungskategorien

Unterscheidung zwischen funktionalen und nicht-funktionalen Anforderungen

Funktionale Anforderungen

- Beschreiben die geforderten Funktionen, also was zu leisten ist (Aufgabenbeschreibung)
- Gegebene und gesuchte Datenobjekte (Schnittstellenbeschreibung)



Nicht-funktionale Anforderungen

- Enthalten Aussagen über die gewünschte Qualität (z.B. Geschwindigkeit, Zuverlässigkeit, Energie, ...)
- Vorgaben für den Entwicklungsprozess und die Systemdokumentation

2.2 Schnittstellenbeschreibung

Die Schnittstelle zwischen einem Algorithmus und seiner „Umwelt“ wird durch die Menge aller Datenobjekte, die den Algorithmus mit seiner Umgebung verbinden und seinen Namen gebildet

- Datenobjekte der Schnittstelle, Parameter genannt, gliedern sich in
  - Eingangsparameter,
  - Ausgangsparameter und
  - Übergangsparameter und ggf. globale Datenobjekte
- Algorithmusname (erforderlich für die Identifikation)

Hinweis: Von der Verwendung globaler Datenobjekte wird strikt abgeraten

Schema zur Spezifikation von Algorithmen

Aufgabe	stichwortartige Beschreibung
Aufruf	Name, Parameter, Funktionswert
Eingangsobjekte	Eingangsparameter mit ihren Datentypen Erklärung der Bedeutung, weitere Bedingungen über Wertebereich formulieren
Ausgangsobjekte/ Ergebnisobjekte	Ausgangsparameter (und ggf. Funktionswert) mit Datentypen und Bedeutung
Funktionale Anforderungen	Verbale Beschreibung der zu erledigenden Aufgabe
Fehlerverhalten	Anforderung an das Verhalten eines Algorithmus/Systems im Fehlerfall

Beispiel

Aufgabe	Suche in *Integer*-Feldern
Aufruf	`FindValue(↓a ↓n ↓x ↑i)`

Eingangsobjekte	`a: array [1:n] of int  -- Feld`
	`n: int -- n > 0`
	`x: int -- Suchelement`

Ausgangsobjekte/ Ergebnisobjekte	`i: int -- Index des gef. Elements`
Funktionale Anforderungen	Wenn der Wert von x im Feld a enthalten ist, soll als Ergebnis `i` der kleinste Index geliefert werden, für den `a[i] = x` gilt. Wenn der Wert von x nicht im Feld a enthalten ist, soll als Ergebnis `i = 0` geliefert werden.
Fehlerverhalten Aufruf mit unerlaubten Datenobjekten	siehe folgende Seiten

z.B. Eingabe eines chars, wo ein int hingehört

Anfangs- und Endzustand (Zustand: Werte der Variablen)

Beschreibung von Anfangszustand, Funktion und Endzustand:

$$Z_A \rightarrow Algorithmus \rightarrow Z_E$$

Vertragscharakter der Spezifikation

- Verwender von Algorithmus muss Z_A einstellen und die Erfüllung der Vorbedingungen sicherstellen, z.B. $n > 0$
- Implementierer muss sich darauf verlassen können
- Wenn Vorbedingungen gegeben sind, dann muss Implementierer Z_E herstellen und Nachbedingungen erfüllen, es muss gelten $Z_E = f(Z_A)$
- Zwischen Z_A und Z_E müssen die spezifizierten Bedingungen gelten

Problem kann mit falschen Werten nicht gelöst werden.

defensive / robuste Programmierung: Ich prüfe die eingehenden Werte und gebe eine Fehlermeldung aus.
design by contract: Anwender bei falscher Eingabe selbst schuld

Folgerungen

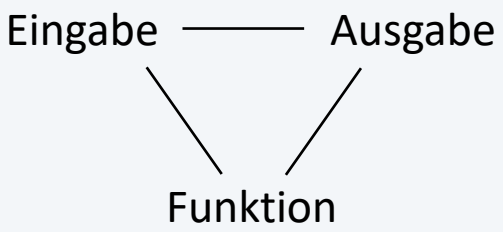
- Z_A muss zur Lösung der Aufgabe reichen; der Implementierer soll und darf darüber hinaus keine Daten benutzen und keine Annahmen treffen
- Algorithmus darf nichts ändern, was nicht zu Z_E gehört
- In Z_A können Bedingungen festgehalten sein, welche Eingangsobjekte erlaubt sind
- Bei verletzten Eingangsbedingungen in Z_A kommt das zu spezifizierende Fehlerverhalten zum Zug

Funktionale Anforderungen	Wenn der Wert von `x` im Feld `a` enthalten ist, soll als Ergebnis `i` der kleinste Index geliefert werden, für den `a[i] = x` gilt. Wenn der Wert von `x` nicht im Feld `a` enthalten ist, soll als Ergebnis `i = 0` geliefert werden.
Fehlerverhalten	Wenn `n ≤ 0` ist, soll ein Fehler gemeldet und das Programm abgebrochen werden.

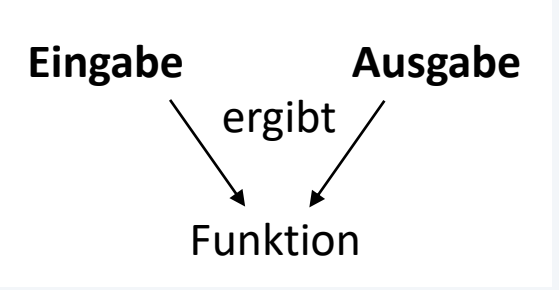
--> defensive / robuste Programmierung

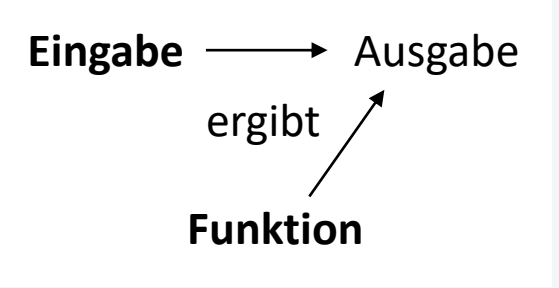
2.3 Aufgabenbeschreibung

- Drei Teile gehören immer zusammen



- Wenn zwei Teile gegeben sind, ergibt sich daraus der dritte
- Ausgangsschema
  - Gegebene Eingabe und gewünschte Ausgabe
  - Funktion wird aus den Daten abgeleitet
- Überführungsschema
  - Gegebene Eingabe und auszuführende Funktion
  - Ausgabe ergibt sich aus der Funktion





Ausgangs- und Überführungsschema

Beschreibung des Anfangs- und des Endzustands sowie der Transformation von Anfangs- in Endzustand

- Ausgangsschema für quadratische Gleichung

  gegeben: $p, q \in R$

  gesucht: $x \in R$, so dass $x^2 + px + q = 0$

  - Kontext ist besser gegeben
  - gut zum Testen -- die x-Werte werden eingesetzt und geprüft ob sie 0 ergeben

- Überführungsschema für quadratische Gleichung

  gegeben: $p, q \in R$

  gesucht: $x \in R = \begin{cases} -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q} & p^2 \geq 4q \\ 0 & p^2 < 4q \end{cases}$

2.4 Darstellungsarten für Aufgabenbeschreibungen

verbal	verbal+formal	formal+verbal	formal
Umgangssprache			math. Ausdrücke
unscharf			präzise
mehrdeutig			eindeutig
verständlich			schwer verständlich
ausschweifend			knapp

Beispiele für Unschärfen und Mehrdeutigkeit:

- Zahlen bis 42 (ist 42 eingeschlossen?)
- nicht negative oder gerade (not n or g, not (n or g)?)
- kleiner Abstand (ungenauer Begriff)
- sortierte Liste (aufsteigend oder absteigend?)

Beispiel: Suche des größten Elements in einem Feld

Verbal	In einem Feld *a* der Länge *n* soll der Index des größten Elements gesucht werden. Wenn es mehrere größte Elemente gibt, soll der Index des ersten Elements geliefert werden.
Verbal mit formalen Teilen	Gegeben ist ein ganzzahliges Feld *a*[1:*n*] mit *n* > 1. Gesucht ist der kleinste Index *i*, so dass *a*[*i*] das größte Element von *a* ist.
Formal mit verbalen Teilen	geg.: `n: int > 1; a: array[1:n] of int` ges.: `i: int,` so dass `a[i] = Max(a)` und `i` ist minimal
Formal	geg.: $n \in N > 1$ und $a_1, a_2, \ldots a_n \in Z$ ges.: $i \in N$, so dass $\forall_{1 \le k \le n, k \neq i} \big((a_i > a_k) \vee (a_i = a_k \wedge i < k) \big)$