Hoofdstuk 1 Probleemoplossend denken

Het leren programmeren bestaat in de eerste plaats uit het leren een probleem te ontleden en er stapsgewijze een oplossingsmethode of algoritme op te bouwen.
Het volstaat immers niet een programmeertaal te kennen om goed te kunnen programmeren.

Programma's worden niet geschreven voor eenmalig gebruik, maar moeten een hele tijd operationeel zijn. Gedurende die tijd moeten, ten gevolge van gewijzigde omstandigheden of een vraag naar meer informatie, soms kleine of grote aanpassingen aangebracht worden. Om dat "onderhoud" van programma's, ook door iemand anders dan de programmeur die het programma oorspronkelijk maakte, gemakkelijk en vlot te kunnen laten uitvoeren, moeten de programma's goed gedocumenteerd en duidelijk gestructureerd zijn.

1 Gestructureerd denken

Met gestructureerd denken bedoelen we de systematische aanpak om via een stapsgewijze verfijning vanuit de probleemstelling een goede oplossing te ontwikkelen die onder vorm van één of meerdere programma's kan uitgevoerd worden. Deze aanpak vraagt het doorlopen van verschillende fasen.

fase 1 Probleemanalyse

In deze fase staat het analyseren van het eigenlijke probleem centraal:
Wat moet er gebeuren? Hoe pak ik dit aan? Wat is het probleem?
Over welke gegevens beschik ik?
Welk resultaat wordt gevraagd?

fase 2 Oplossingsstrategie

Hier proberen we een gestructureerde oplossing te formuleren van het probleem.
De basisstructuren voor het programma worden hier gemaakt.
Hoe los ik het probleem op?
Welke verfijnde technieken kan ik hiervoor gebruiken?

fase 3 Algoritme

Een algoritme is een voorschrift, opgebouwd uit één of meer stappen, dat precies aangeeft hoe men vanuit, een gegeven beginsituatie een vooraf beschreven resultaat kan bereiken.

Een algoritme moet volgende ***eigenschappen*** bezitten:

1. De opdrachten van een algoritme moeten duidelijk, ondubbelzinnig en volledig gedefinieerd zijn. Geen enkele stap uit het algoritme mag voor verschillende interpretaties vatbaar zijn.
2. Elke opdracht van een algoritme moet vrij zijn van redundantie, dwz ze mag niet te gedetailleerd zijn, ze moet vrij zijn van overbodige details.
3. Een algoritme moet vermelden welke gegevens nodig zijn en welke het vooropgestelde doel is. Men noemt het uitschrijven van deze gegevens het opstellen van de invoer- en uitvoerspecificaties van het algoritme.
4. Een goed algoritme moet een structuur hebben. Een gestructureerd algoritme is nl. goed leesbaar, eventueel gemakkelijk te wijzigen en handig bij het opsporen van fouten.
5. Een algoritme moet eindigen.

Er bestaan verschillende schematische voorstellingen voor dit programmeeralgoritme.
Vb. voorstellingswijze: **pseudo-code**

fase 4 : Programma

Na de schematische voorstelling wordt het eigenlijke programma geschreven in C#, Visual Basic, Visual Basic for Applications, C++, Java, ...

fase 5 : Testen van het programma

Door middel van testgegevens wordt het eigenlijke programma getest of het gewenste resultaat bereikt wordt.

fase 5 : Documentatie

Dit is vooral van nut om het programma later te kunnen aanpassen. In de documentatiemap zitten o.a. de documenten m.b.t. de probleemanalyse en de schema's die werden getekend en uitgewerkt. Meestal moet ook een gebruikershandleiding geschreven worden. Je programmeercode moet ook veelvuldig voorzien zijn van commentaarregels!

Voorbeeld

Fase 1 Probleemanalyse

Gegeven: /

Gevraagd:
- Wijzig tekst in "Welkom programmeurs"
- Berichtenvenster "Hallo netwerkbeheer"
- Wijzig tekst in "Muis zweeft over label"

Fase 2 Oplossingsstrategie

- Klik op 'Wijzigen': druk tekst "Welkom programmeurs" in label
- Klik op 'Bericht': berichtenvenster "Hallo netwerkbeheer"
- Muis over label: druk tekst "Muis zweeft over label" in label

Fase 3 Pseudocode

```
Begin   BtnWijzigen_
    druk  "Welkom programmeurs" (LblHallo)
Einde

Begin   BtnBericht_Click
    druk "Hallo netwerkbeheer" (berichtenvenster)
Einde

Begin   LblTekst_MouseEnter
    druk  "Muis zweeft over label" (LblHallo)
Einde
```

Fase 4 Programmeren

4.1 Ontwerpfase/ Visual programming

4.2 Programmeerfase / Code programming

```
private void ButtonWijzigen_Click(object sender, RoutedEventArgs e)
{
    LabelHallo.Content = "Welkom programmeurs!";
}
```

```
private void LabelHallo_MouseEnter(object sender, MouseEventArgs e)
{
    LabelHallo.Content= "Muis zweeft over label.";
}
```

```
private void ButtonBericht_Click(object sender, RoutedEventArgs e)
{
    // Berichtenvenster oproepen.
    MessageBox.Show("Hallo Netwerkbeheer", "Berichtenvenster");
}
```

Fase 5 Testen

Gebruik testgegevens om fouten op te sporen.

Fase 6 Documentatie

Ontwerp, probleemanalyse, algoritme, helpfuncties,… alle informatie over de toepassing.

2 Het gestructureerd schema

Het leren programmeren bestaat in de eerste plaats uit het leren een probleem te ontleden en stapsgewijze een oplossingsmethode of algoritme hier rond op te bouwen.
Het volstaat immers niet een programmeertaal te kennen om goed te kunnen programmeren.

Programma's worden niet geschreven voor eenmalig gebruik, maar moeten een hele tijd operationeel zijn. Gedurende die tijd moeten, ten gevolge van gewijzigde omstandigheden of een vraag naar meer informatie, soms kleine of grote aanpassingen aangebracht worden. Om dat "onderhoud" van programma's, ook door iemand anders dan de programmeur die het programma oorspronkelijk maakte, gemakkelijk en vlot te kunnen laten uitvoeren, moeten de programma's goed gedocumenteerd en duidelijk gestructureerd zijn.

2.1 Bepaling

Een gestructureerd schema is een gedetailleerde voorstelling van de opeenvolging van operaties die naar de oplossing van een gesteld probleem leiden, gebruik makend van structuurelementen die slechts één ingang en één uitgang hebben.

2.2 Doel

Gestructureerde schema's worden opgesteld om het verloop van de redenering die moet gevolgd worden ten einde een bepaald probleem op te lossen, op een duidelijke en ondubbelzinnige manier weer te geven. Zij leggen een band tussen de probleemdefinitie en het uiteindelijke programma.

Tot de belangrijkste kenmerken van systematische oplossingsmethodes kunnen we rekenen:

- Het op te lossen probleem wordt in kleine delen gesplitst, waardoor het mogelijk wordt in teamverband een oplossing uit te werken. Dit vraagt duidelijke omschrijvingen van de gebruikte structuren en modules.
- Men maakt gebruik van een reeks structuren die op een goed afgebakende manier gedefinieerd worden.
- Een probleem wordt losgekoppeld van een programmeertaal zelf.

3 De pseudo-code

De "pseudo-code" is een *fictieve* programmeertaal, die toelaat de uit te voeren opdrachten in eigen taal te formuleren. Dit biedt volgende voordelen:

- Men kan zich beter concentreren op de oplossing van het eigenlijke probleem zelf. Er moet immers geen rekening worden gehouden met de syntaxregels van een bepaalde programmeertaal.
- Het programma is beter leesbaar, waardoor de communicatie met niet-informatici zoals bij voorbeeld de directie van een onderneming gemakkelijker wordt.

4 Basisprogrammastructuren

4.1 Sequentie

S1
S2
S3
S4
……

```
Begin
   lees  F1
   lees F2
   lees F3
   SOM = F1 + F2 + F3
   druk SOM
Einde
```

4.2 Selectie

als-dan-anders

```
als V
      dan S1
      anders S2
eindals
```

```
Begin
 lees L
 als L < 16
    dan  lees CODE
         als CODE = 1
            dan druk "Toestemming met ouders"
            anders druk "Geen toestemming"
         eindals
   eindals
Einde
```

ingeval

ingeval I is
    1: S1
    2: S2
      …
    n: Sn
eindgeval

```
ingeval  getal is
    a: HG=10
    b: HG=11
    c: HG=12
eindgeval
```

4.3 Iteratie

Zolang-doe

zolang V
    doe S
eindzolang

```
zolang (BK<= EK)
    doe  BK = BK * (1 + R/100)
         JR = JR +1
         druk BK
eindzolang
```

Voor-doe

voor I van BW tot EW [stap]
    doe S
einddoe

```
Begin
    voor tel van 1 tot 10
        doe  lees F1, F2, F3
             druk F1+F2+F3
    einddoe
Einde
```