

Didactiek bij Programmeren

Probleemsituatie

Leren programmeren is moeilijk.
Duidelijk is dat een student moet oefenen met de lesstof.

Bij hoorcolleges moeten echter eerst wel nieuwe constructies en concepten eerst worden overgedragen. Vaak gaat dit niet in één keer goed, bij practica lopen sommigen zelfs meteen vast bij het toepassen van de nieuwe stof; hoe komt dat?

Code is voor een beginnening véél complexer dan voor ervaren programmeurs. Er is al snel **cognitieve overload**.^[ii] Zelfs code die nog helemaal niets doet is in het begin tóch heel complex!

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

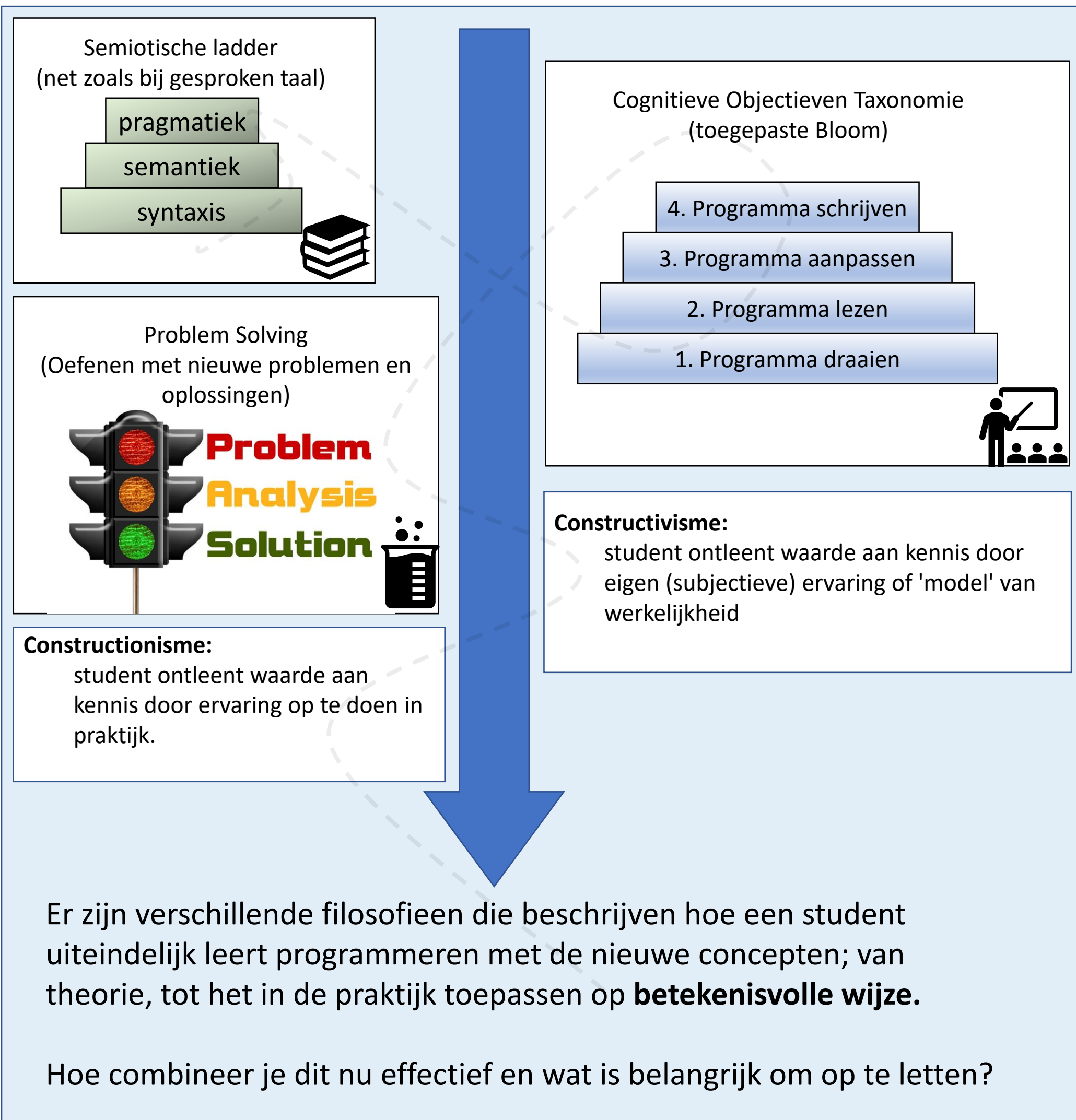
Docenten pakken dit meestal op door stap-voor-stap de betekenis en bedoeling van een programma uit te leggen en zo geleidelijk, volgens **constructionisme**^[i] langzaam aan iedere student zelf een begrip te laten opbouwen. Puur toelichten; dat werkt niet; 'zelf' uitvinden is nodig.

Verbetermogelijkheden

Bij het leren van programmeren is de cognitieve belasting meestal hoog. Daarom is het belangrijk dat gekozen voorbeelden^[iii] en opdrachten goed aansluiten bij elkaar én bij het leerdoel. Dit is zeker niet altijd het geval bij de programmeerlessen die we nu geven.

- Er is sprake van **redundantie** in voorbeelden; dit stoort met het leereffect en verhoogt onnodig de cognitieve belasting.
- Er is sprake van een **split-attention** effect; men moet tegelijkertijd meerdere informatiebronnen combineren; het voorbeeld én het concept én de nog niet zo bekende programmeertaal. Al deze informatie is tekstueel. (welke info kan via een andere **modaliteit**?; beeld? toelichting met spraak of plaatjes?)
- Betere afstemming van voorbeelden op voorkennis is mogelijk voor ofwel lagere cognitieve belasting, of effectievere cognitieve belasting.
- **Workshop**; een workshop vorm zou beter zijn dan hoorcollege+practicum, maar is budgetair lastig te realiseren.
- De meeste practicumopdrachten vergen dat studenten vanaf de grond af aan alles opbouwen, maar er is een goed beschreven '**completion effect**'^[iii]; Een grotendeels opgelost probleem met wat ontbrekende elementen, zodat de oplossing niet te ver weg is.

Didactische modellen bij programmeren



Hypothese

Voor het effectieve aanleren van programmeren moet er een integrale aanpak en koppeling bestaan tussen de opdrachten in het practicum en de specifieke constructies en concepten die aan bod zijn gekomen bij het hoorcollege.

Hierbij kunnen verschillende didactische inzichten worden gecombineerd om tot een goed rendement te komen voor studenten met verschillende behoeften, terwijl de cognitieve belasting zo laag en zo effectief mogelijk wordt gehouden.

Interventie

Verminder de cognitieve belasting en versoepel de overstap tussen kennis en toepassing.

Maak gebruik van een **Worked Example**^[iv]; een voorbeeld dat in het hoorcollege is behandeld en uitgelegd, en laat dit één van de eerste opdrachten zijn in het practicum.

Probeer een 'overload' te voorkomen door hier geen 'te ingewikkelde' of te ver uit-ontwikkelde voorbeelden te geven.

Het ideaal is dat het te bereiken doel van de practicumopdracht zo voor de hand ligt dat het nauwelijks bewuste aandacht nodig heeft. De focus ligt zo veel mogelijk ligt op het toepassen van de nieuwe programmeerconcepten. Na deze afgedekte basis kan worden overgegaan naar betekenisvolle toepassing.

Verwachte resultaten

- Een beter begrip van de basis bij de studenten
- Minder verwarring bij de opdrachten.
- Soepelere transitie van hoorcollege naar practicumopdrachten
Kennis → toepassing

Bronnen:

- Auteur poster: Joep Lijnen, Academie voor Engineering & ICT
- Plaatjes: pixabay.com
- in deze poster gebruikte literatuur:

- Jens J. Kaasbøll, D. o. (1998). Opgehaald van <https://pdfs.semanticscholar.org/a074/b719b2457ce67a95a0dac92ad6c8da0d15d7.pdf>
- Tuovinen, J. (2000). *Optimising student cognitive load in computer education*. Opgehaald van https://www.researchgate.net/publication/221222883_Optimising_student_cognitive_load_in_computer_education
- Xiaoqing Li, U. o. (2016). *Application of Cognitive Load Theory in Programming Teaching*. Opgehaald van Journal of Higher Education Theory and Practice Vol 16.: http://www.na-businesspress.com/JHETP/LIX_Web16_6_.pdf
- Sweller, j. (1985). *The use of worked examples as a substitute for problem solving in learning algebra*. Opgehaald van <https://psycnet.apa.org/record/1988-05885-001>, https://www.tandfonline.com/doi/abs/10.1207/s1532690xci0201_3

