Didactische aspecten van algoritmiek

Erik Barendsen (Radboud Universiteit & Open Universiteit)

Nascholing Algoritmisch Denken, TUD & RU, 2016-2017, versie 23 II

Didactische aspecten

leerdoelen (wat en waarom)

begrip van leerlingen (mogelijkheden en beperkingen)

onderwijsaanpak

toetsing

(Magnusson, Krajcik & Borko, 1999; Van Driel & Henze, 2012)

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

Het nieuwe examenprogramma

Kernprogramma		*Keuzethema's*	
Domein A	**Vaardigheden**	Domein G	**Algoritmiek, berekenbaarheid en logica**
Domein B	**Grondslagen**		
Domein C	**Informatie**	Domein H	**Databases**
Domein D	**Programmeren**	Domein I	**Cognitive computing**
Domein E	**Architectuur**	Domein J	**Programmeerparadigma's**
Domein F	**Interactie**	Domein K	**Computerarchitectuur**
		Domein L	**Netwerken**
		Domein M	**Physical computing**
		Domein N	**Security**
		Domein O	**Usability**
		Domein P	**User experience**
		Domein Q	**Maatschappelijke en individuele invloed van informatica**
		Domein R	**Computational science**

(Barendsen & Tolboom, 2016; Barendsen, Grgurina & Tolboom, 2016)

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

Domein A: Vaardigheden

- **Ontwerpen en ontwikkelen**
- **Informatica hanteren als perspectief**
- **Samenwerken en interdisciplinariteit**

- Ethisch handelen
- Informatica-instrumentarium hanteren
- Werken in contexten

- *Algemene vaardigheden*
- *Wetenschappelijke vaardigheden*

Ontwerpen en ontwikkelen

De kandidaat kan

- in een context mogelijkheden zien voor het inzetten van digitale artefacten;
- deze mogelijkheden vertalen tot een doelstelling voor ontwerp en ontwikkeling, daarbij technische factoren, omgevingsfactoren en menselijke factoren betrekken;
- wensen en eisen specificeren en deze op haalbaarheid toetsen;
- een ditaal artefact ontwerpen;
- bij het ontwerp van een digitaal artefact keuzes afwegen via onderzoeken en experimenteren;
- een digitaal artefact implementeren;
- de kwaliteit van digitale artefacten evalueren,

en deze vaardigheden in samenhang inzetten voor het ontwikkelen van digitale artefacten.

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

Domein B: Grondslagen

Algoritmen

De kandidaat kan

- een oplossingsrichting voor een probleem uitwerken tot een algoritme;
- daarbij standaardalgoritmen herkennen en gebruiken;
- de correctheid en efficiëntie van digitale artefacten onderzoeken via de achterliggende algoritmen.

Datastructuren

De kandidaat kan verschillende abstracte datastructuren met elkaar vergelijken op elegantie en efficiëntie.

Automaten

De kandidaat kan eindige automaten gebruiken voor de karakterisering van bepaalde algoritmen.

Grammatica's

De kandidaat kan grammatica's hanteren als hulpmiddel bij de beschrijving van talen.

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

Domein D: Programmeren

Ontwikkelen

De kandidaat kan, voor een gegeven doelstelling,

- programmacomponenten ontwikkelen in een imperatieve programmeertaal;
- daarbij programmeertaalconstructies gebruiken die abstractie ondersteunen;
- programmacomponenten zodanig structureren dat ze door anderen gemakkelijk te begrijpen en te evalueren zijn.

Inspecteren en aanpassen

De kandidaat kan

- structuur en werking van gegeven programmacomponenten uitleggen;
- zulke programmacomponenten aanpassen op basis van evaluatie of veranderde eisen.

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

Domein G: Keuzethema Algoritmiek, berekenbaarheid en logica

Complexiteit van algoritmen

De kandidaat kan

(havo:) van gegeven algoritmen de complexiteit vergelijken, en kan klassieke *moeilijke* problemen herkennen en benoemen.

(vwo:) het verschil tussen exponentiële en polynomiale complexiteit uitleggen, kan algoritmen op basis hiervan onderscheiden, en kan klassieke *moeilijke* problemen herkennen en benoemen.

Berekenbaarheid

De kandidaat kan berekeningen op verschillende abstractieniveaus karakteriseren en relateren, en kan klassieke *onberekenbare* problemen herkennen en benoemen.

Logica

De kandidaat kan eigenschappen van digitale artefacten uitdrukken in logische formules.

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

Computational thinking

- algoritmisch denken
- decompositie
- generalisatie
- abstractie
- evaluatie

(Wing, 2006; CSTA, 2011; Selby & Woollard, 2013)

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

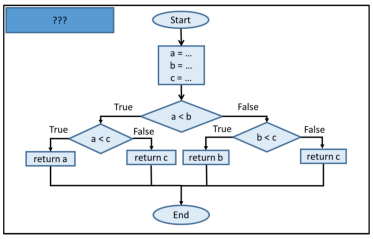
Representatie van algoritmen

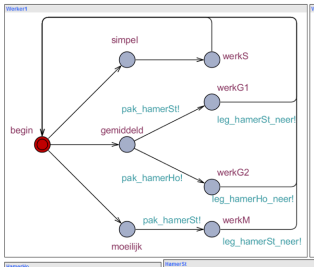
- natuurlijke taal
- pseudocode
- stroomdiagrammen
- eindige automaten

- programmacode?

- {Input: Gesorteerde array A[0 … *N-1*] , element *x*}
- {Output: index *i* met A[*i*] = *x*, als zo'n *i* bestaat;
  0, als x < A[*0*];
  de grootste *i* met A[*i-1*] < *x*, anders}

1. order = 0; boven = *N-1;*
2. **while** (order < boven) **do**
   mid = (order + boven) / 2;
   if (A[mid] < *x*)
   then order = mid+1
   else boven = mid
3. **return** order





(Smetsers-Weeda, 2016)

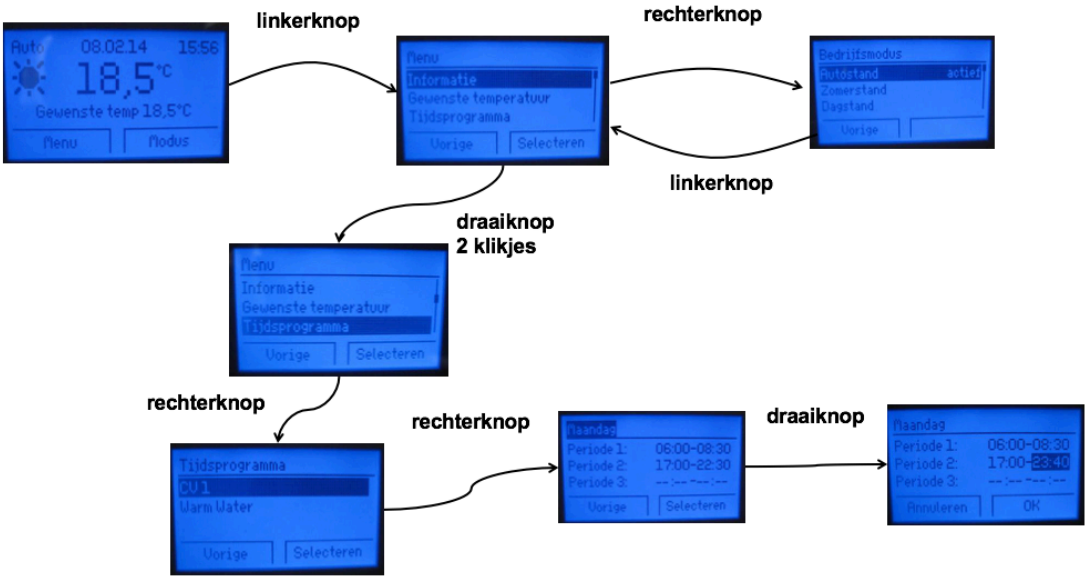
Automaten

- met dank aan Erik Poll, Sjaak Smetsers, Simon Peyton Jones

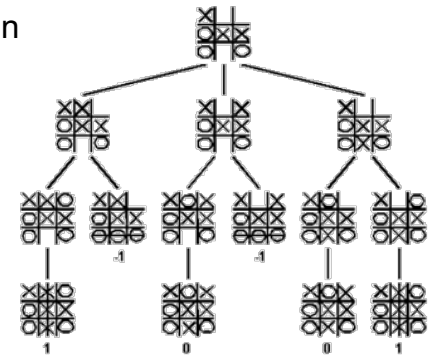
leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

netwerkverbinding

Radboud Universiteit



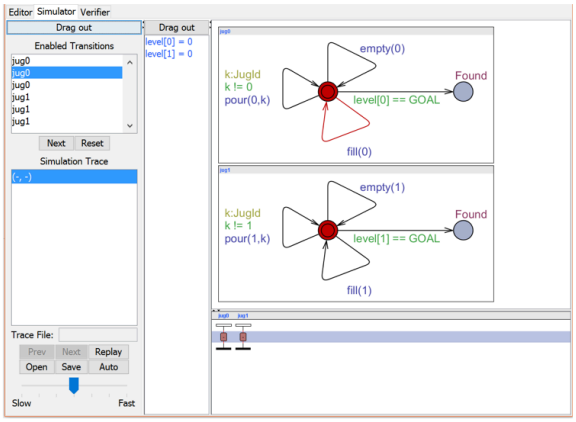
- Eindige automaat: toestanden en toestandsovergangen (transities)
- Elke computer/systeem/programma heeft gedrag dat in termen van toestanden en transities te beschrijven is
  (maar in het ene geval is dat handiger/bruikbaarder dan in het andere)
- Veel puzzels en spelletjes zijn ook zo te begrijpen



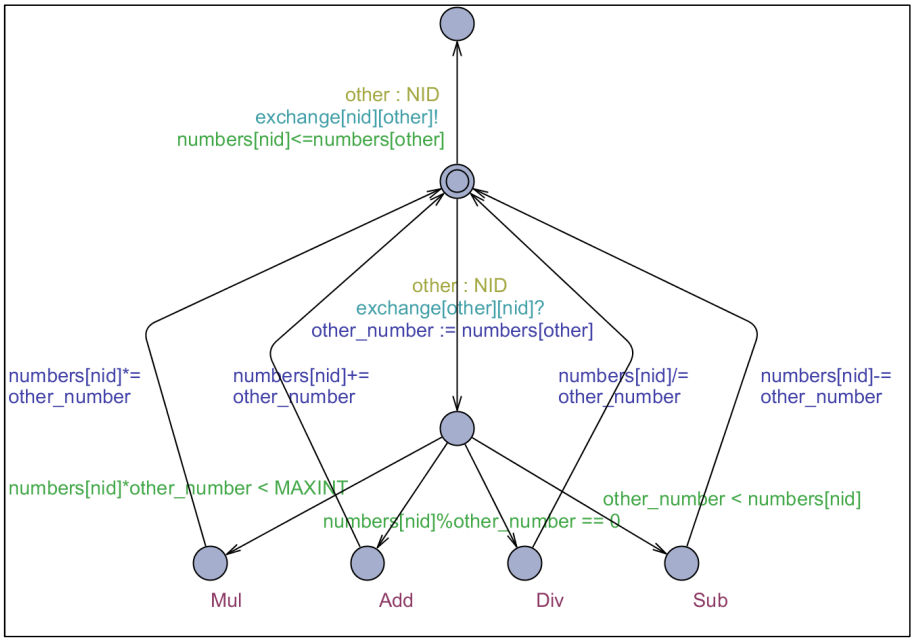
Example: Jug Filling

- movie: Die hard with vengeance
- Given:
  - a five gallon jug
  - a three gallon jug
  - a way to fill up the jugs
  - a way to pour out water
- End up with:
  - exactly 4 gallon of water in one of the jugs.

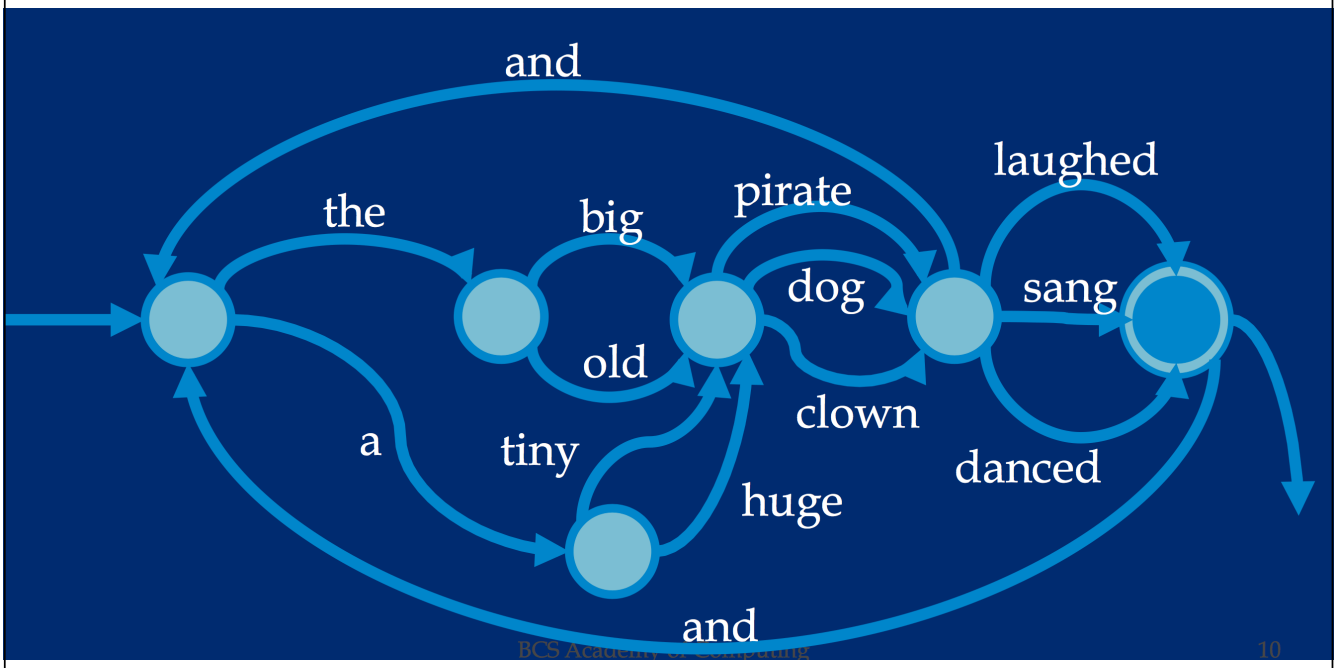
Jug filling – Simulator



leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing



Radboud Universiteit



Ontwikkeling van leerlingen: leren programmeren

- enkele dia's overgenomen uit: keynote Raymond Lister, WiPSCE 2016

(Lister, 2016)

Traceren


```
int x = 7;
int y = 5;
int z = 3;

x = y;
z = x;
y = z;
```

- welke waarde heeft x na afloop?
- en y? en z?

Uitleggen

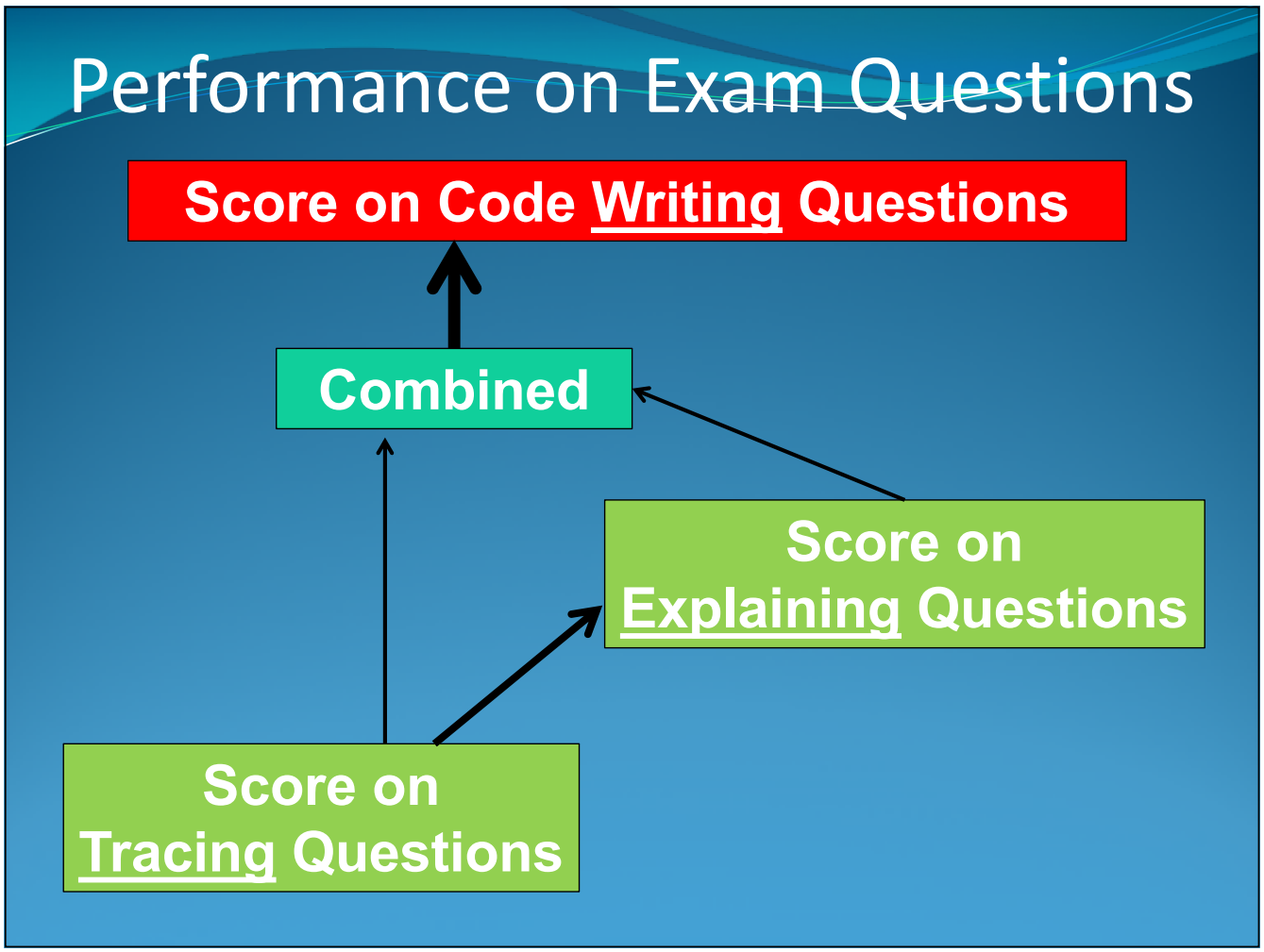

```
if (y1 < y2):
    Code to swap the values in
    y1 and y2 goes here.

if (y2 < y3):
    Code to swap the values in
    y2 and y3 goes here.

if (y1 < y2):
    Code to swap the values in
    y1 and y2 goes here.
```

- wat is de bedoeling van dit programma?

Performance on Exam Questions



leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

Neo-Piagetiaans model: vier ontwikkelingsfasen

- sensorimotor
  kan niet traceren
  incoherent begrip van programma-executie
- pre-operationeel
  kan regels code traceren (“met de hand executeren”)
  raadt naar werking van code door voorbeelden te traceren
- concreet-operationeel
  kan redeneren over programmacode
  door de code zelf te lezen
- formeel-operationeel
  redeneert op expertniveau, incl. reflectie

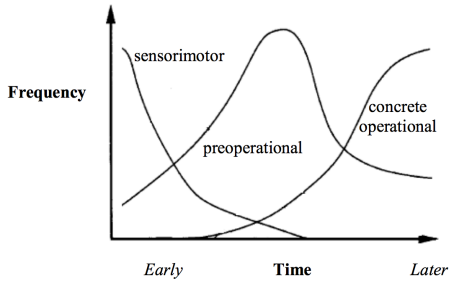
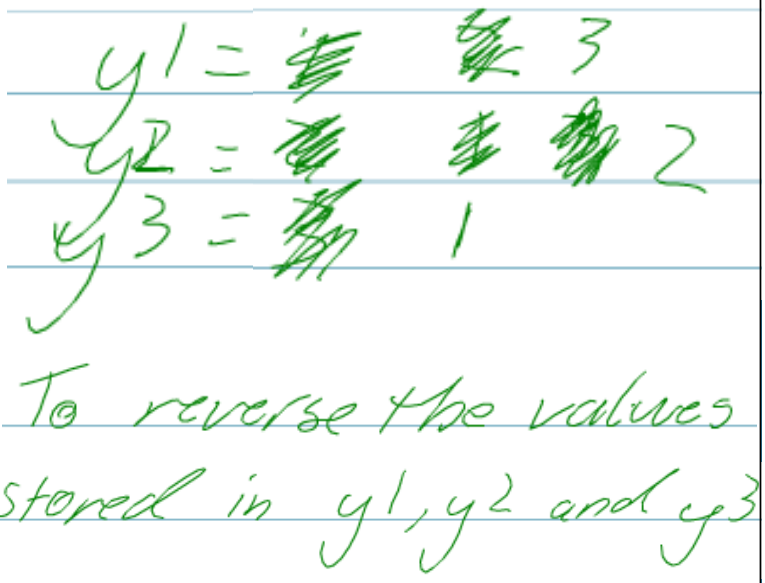


Figure 1: The Overlapping Waves Model

Preoperational "Donald"

(2 - 3 minutes)


```
if ( y1 < y2 ) {
    swap y1 and y2
}
if ( y2 < y3 ) {
    swap y2 and y3
}
if ( y1 < y2 ) {
    swap y1 and y2
}
```



See BOOKMARKED Donald_230911_pp. 67~68

Alternatieve concepties / misconcepties

leerdoelen (wat en waarom)
begrip van leerlingen (mogelijkheden en beperkingen)
onderwijsaanpak
toetsing

- Misconcepties over controlstructuren, variabelen
- *notional machine*

- Misconcepties over efficiëntie
  - Een korter programma is efficiënter
  - Twee programma's met dezelfde instructies zijn even efficiënt
  - Hoe minder variabelen, des te efficiënter is het programma

```
void A( ){
   int n = 100;
   output("This program prints n lines.");
   for ( int i = 0; i < n; i = i + 1 ){
      output ("Line No." + i);
   }
   n = n + 500;
}
void B( ){
   int n = 100;
   n = n + 500;
   output("This program prints n lines.");
   for ( int i = 0; i < n; i = i + 1 ){
      output ("Line No." + i);
   }
}
```

(Du Boulay, 1986; Sorva, 2013; Gal-Ezer & Zur, 2004; Shah, Capovilla & Hubwieser, 2015)

leerdoelen (wat en waarom)

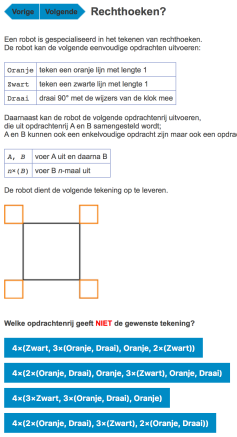
begrip van leerlingen (mogelijkheden en beperkingen)

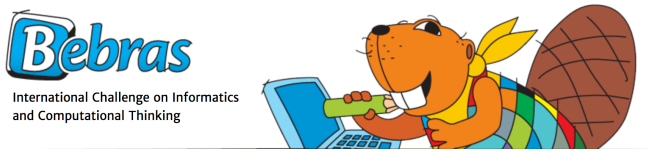
onderwijsaanpak

toetsing

Bever-opgaven als onderwijs- en toetsvorm

- classificatie: abstractie, algoritmisch denken, decompositie, evaluatie, generalisatie





(Barendsen et al., 2015; Dagienė & Sentance, 2016)

Literatuur

- Barendsen, E., Grgurina, N., & Tolboom, J. (2016). A new informatics curriculum for secondary education in The Netherlands. In A. Brodnik & F. Tort (Eds.), *Informatics in schools: Improvement of informatics knowledge and perception* (pp. 105–117). Springer.
- Barendsen, E., Mannila, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., . . . Stupurienė, G. (2015). Concepts in K–9 computer science education. In *Proceedings of the 2015 ITiCSE on Working Group Reports* (pp. 85–116). ACM.
- Barendsen, E., & Tolboom, J. (2016). *Advies examenprogramma informatica vwo-havo: inhoud en invoering*. Enschede: SLO.
- CSTA. (2011). *Computational thinking teacher resources, second edition.* (Retrieved from http://csta.acm.org/Curriculum/sub/CompThinking.html, July 2013.)
- Dagienė, V., & Sentance, S. (2016). It's Computational Thinking! Bebras tasks in the curriculum. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 28–39).
- Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57–73.

Radboud Universiteit

- Gal-Ezer, J., & Zur, E. (2004). The efficiency of algorithms—misconceptions. *Computers & Education*, *42*(3), 215–226.
- Lister, R. (2016). Toward a developmental epistemology of computer programming. In *Proceedings of the 11th workshop in primary and secondary computing education* (pp. 5–16).
- Magnusson, S., Krajcik, J., & Borko, H. (1999). Nature, sources, and development of pedagogical content knowledge for science teaching. In J. Gess-Newsome & N. G. Lederman (Eds.), *Examining pedagogical content knowledge* (pp. 95–132). Dordrecht: Kluwer.
- Selby, C., & Woollard, J. (2013). *Computational thinking: the developing definition.* Retrieved November 2016, from http://eprints.soton.ac.uk/356481
- Shah, P., Capovilla, D., & Hubwieser, P. (2015). Searching for barriers to learning iteration and runtime in computer science. In *Proceedings of the workshop in primary and secondary computing education* (pp. 73–75).
- Smetsers-Weeda, R. (2016). *Think. . . then act(): Flowcharts as a tool for novice programmers to enhance their problem solving skills*. Unpublished master's thesis, Delft University of Technology.
- Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education*, *13*(2), 8.

- Van Driel, J. H., & Henze, I. (2012). *Extended paper for PCK Summit, Colorado 2012.* Retrieved from http://pcksummit.bscs.org.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Radboud Universiteit