Toetsing
Domein B: Algoritmiek
Domein D: Programmeren

7 November, 2018

Renske Smetsers-Weeda

Kennis maken

- Renske Smetsers-Weeda
- Docent informatica (Montessoricollege Nijmegen)
  - 1e graads bevoegd voor informatica & wiskunde
- Promovendus aan de Radboud Universiteit Nijmegen
  - Toetsing voor het nieuw examenprogramma
  - Specifiek: Algoritmen, Programmeren, Vaardigheden (Computational Thinking)

Doel van deze presentanite

Hoe bepaal je doeltreffend wat een leerling weet/kan?

- Doeltreffend: o.a. eerlijke toets, correcte beeld (kloppend bewijs), juiste beoordeling (cijfer), onderscheidend, motiverend.. En vooral: makkelijk&snel uit te voeren door een docent

- Kennis uitwisselen
- Onderzoek vs. praktijk

Wat willen we dat leerlingen leren? (een selectie)

- Domein A: Vaardigheden
  - Computational Thinking (algoritmisch denken, abstractie, generalisatie, decompositie, evaluatie)
  - Reflecteren, onderzoeken, modelleren
  - Werken in contexten
- Domein B: Grondslagen
  - Algoritmen: oplossingsrichting modelleren/uitwerken en (standaardalgoritmen) analyseren op correctheid en efficiëntie
- Domein D: Programmeren
  - Algoritme → goed gestructureerde code (debuggen, aanpassen, creëren, evalueren)

Evidence Centered Assessment Design (ECD)

- Methode om gestructureerd van leerdoelen tot toetsen te komen
- Concrete leerdoelen → potentiële observaties → situaties waarin leerlingen de mogelijkheid geboden worden om hun kennis/vaardigheden te laten zien
- Variabelen mbt opdrachten:
  - Moeilijkheid: leerlingen laten excelleren
  - Contexten: variëren (profielen, j/m)
- Nakijkmodellen/rubrics
- Pilot uitvoeren, evalueren en bijstellen

ECD Layer	Role	Key Entities & Examples
1. Domain Analysis	Gather substantive information about the computational thinking domain of interest that has implications for assessment; how knowledge is constructed, acquired, used, and communicated.	Computational thinking domain concepts (e.g., abstraction, automation); terminology (debugging); tools (programming languages); representations (storyboards); situations of use (modeling predator-prey).
2. Domain Modeling	Express assessment argument in narrative form based on information from Domain Analysis	Specification of knowledge, skills, and other attributes to be assessed (e.g., describe result of running a program on given data); features of situations that can evoke evidence (find errors in programs); kinds of performances that convey evidence (use of recursion).
3. Conceptual Assessment Framework	Express assessment argument in structures and specifications for tasks and tests, evaluation procedures, measurement models.	Student, evidence, and task models; student, observable, and task variables; rubrics; measurement models; test assembly specifications; task templates and task specifications.
4. Assessment Implementation	Implement assessment, including presentation-ready tasks and calibrated measurement models	Task materials (including all materials, tools, affordances); pilot test data to hone evaluation procedures and fit measurement models.
5. Assessment Delivery	Coordinate interactions of students and tasks: task-and test-level scoring; reporting.	Tasks as presented; work products as created; scores as evaluated.

CS concepten en CT vaardigheden:

- Wat is CT?
  - "*solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science*" (Wing, 2006)
  - Algoritmisch denken, abstractie, generalisatie, decompositie, evaluatie
- 3D model van Brennan and Resnick (2012) propose three dimensions of CT:
  1. Computational concepts: variables, loops, methods, …
  2. Computational practices: debugging, abstracting, being incremental, …
  3. Computational perspectives: questioning about the technology world
- Computational Thinking is perceived to be one of the more challenging concepts to teach and evaluate (Brennan & Resnick, 2012).
- CT is multi-faceted (veelzijdig), toetsing moet dat dus ook zijn! (Grover & Pea, 2007)

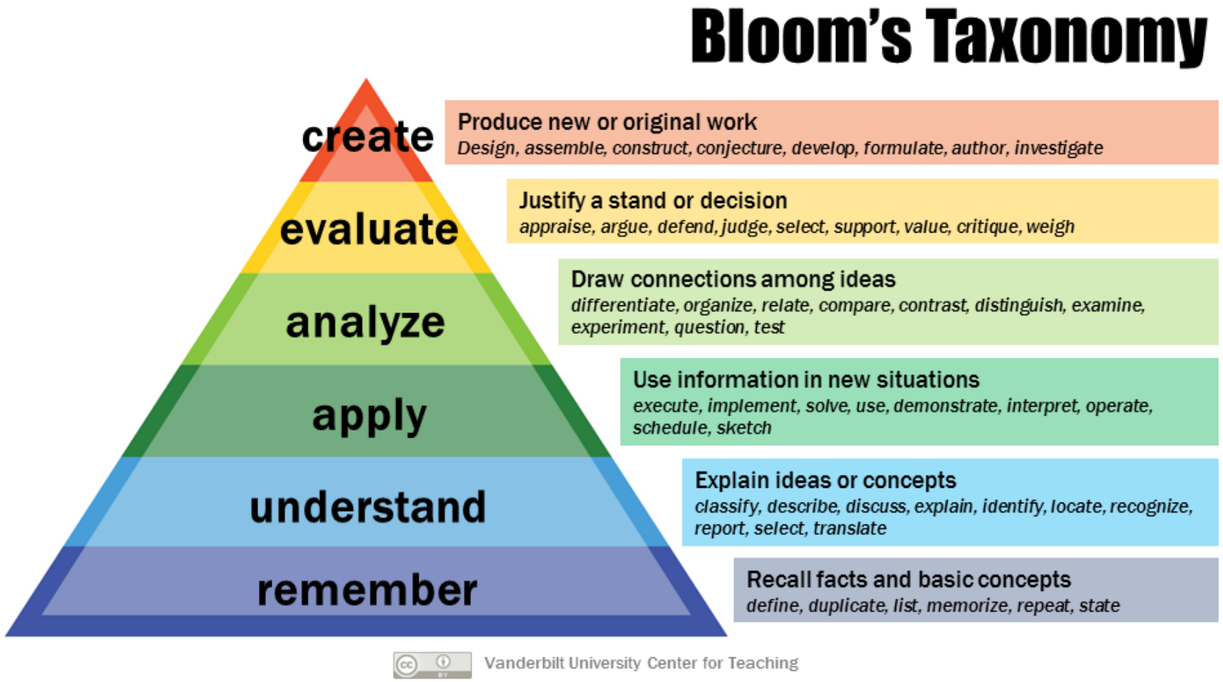
Taxonomieën

- Bloom
- Solo
- Combined Bloom-Solo

Wat voor soort kennis willen we toetsen:

1. Factual knowledge (knowing what)
2. Procedural knowledge (knowing how)
3. Conceptual knowledge (knowing why)
4. Meta-cognitive knowledge (knowing about knowing)

(Streun, 2001)



Niveau van kennis:

- SOLO: Structure of the Observed Learning Outcome (Biggs, 1996)
- In termen van complexiteit

Prestructural	incapabel
Unistructural	code regel-voor-regel kunnen lezen (tracen)
Multistructural	kunnen uitleggen wat de bedoeling van een stuk code is (kunnen samenvatten)
Relational	samenhang tussen verschillende stukken code begrijpen

Voorbeeld vraag: Wat doet deze code eigenlijk?

```
public int mm1 ( int m, int n ) {
    int i = 0;
    int j = 1;
    while ( i < n ) {
        j = j * m;
        i = i + 1;
    }
    return j;
}
```

```
public int mm2 ( int m ) {
    int j = 1;
    j = j + mm1( m, 3 );

    return j;
}
```

a) Maak een trace tabel voor de aanroep mm1(2, 3).

APPLY (Bloom)-UNISTRUCTURAL(SOLO)

b) Verzin een betere naam voor de functie mm1 / Leg in je eigen woorden uit wat mm1 uitrekent.

ANALYZE/EVALUATE(Bloom) – MULTISTRUCTURAL(SOLO)

→ Complexiteit als combinatie van **Bloom-SOLO** (Meerbaum-Salant, 2013)

Combined Bloom-Solo

- **Understanding**: The ability to summarize, explain, exemplify, classify, and compare CS concepts, including programming constructs.
- **Applying**: The ability to execute programs or algorithms, to track them, and to recognize their goals.
- **Creating**: The ability to plan and produce programs or algorithms (constructing, analysing, evaluating and formulating).

Each of these are further subdivided into three levels:

- **Unistructural**: The ability to create very small scripts doing one thing, or adding an instruction with a local effect to an existing script.
- **Multistructural**: The ability to track a program and grasp various parts but not the entity they form as a whole.
- **Relational**: The ability to fully understand a complex concept (such as concurrency) and to coherently explain its various facets.

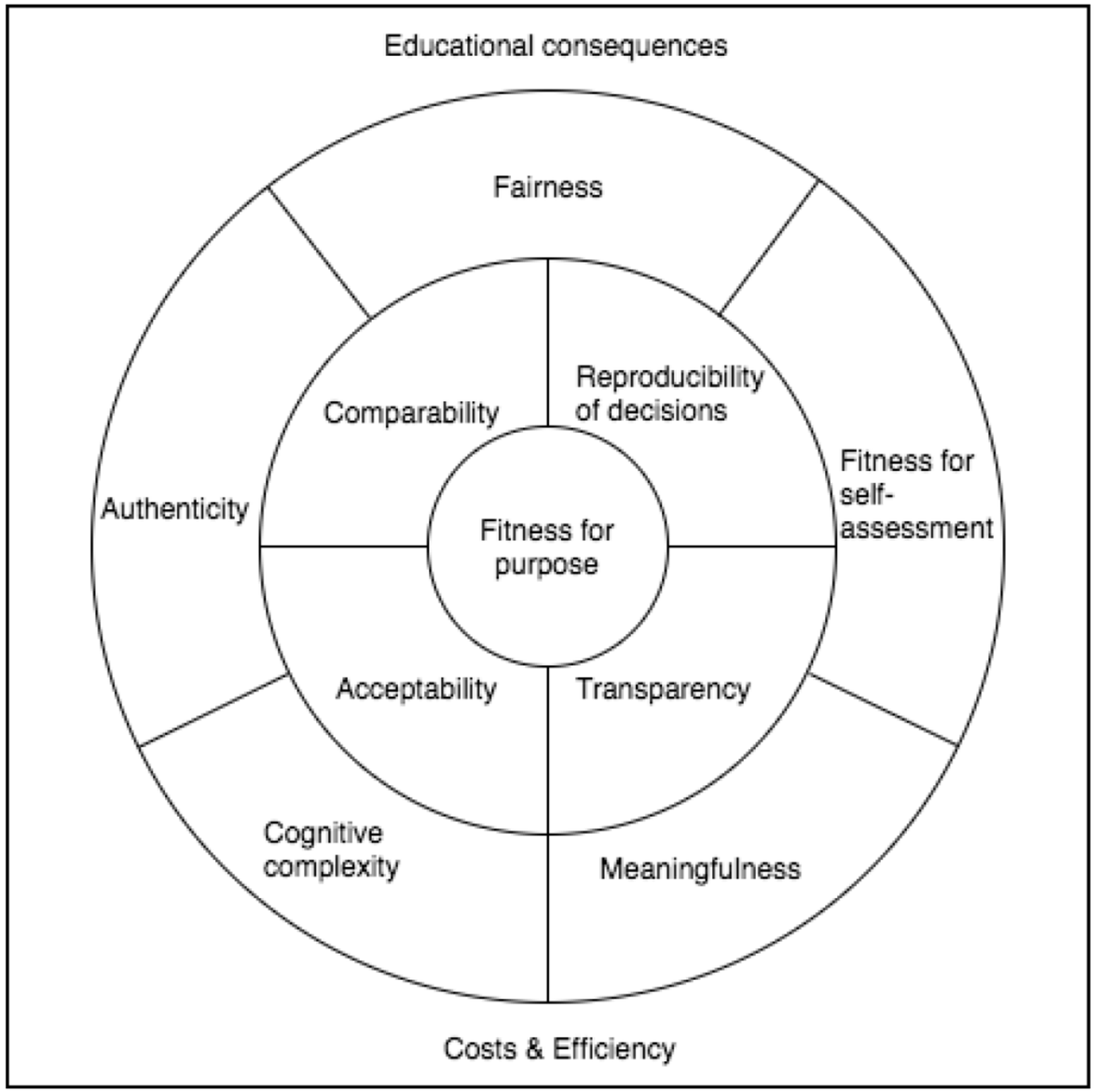
(Meerbaum-Salant, 2013)

Wat voor soort taken zijn waarvoor geschikt?

- Programmeren betreft zowel **kennis** als **vaardigheden**
  - **Kennis** en **vaardigheden** toesten
  - Voor de hand liggend is een schriftelijke theorie toets (kennis) gecombineerd met een PO (vaardigheden).
- **Kennis**: (schriftelijke) theorie toetsen voor de hand liggend
  - Individueel, minder kans op plagiaat
  - Taxonomie: lager niveau (onthouden/begrijpen/toepassen)
- **Vaardigheden**: PO incl. ontwerp + resultaat + evaluatie/reflectie is hier voor de hand liggend
  - Uitvoering in team (samenwerkingen)
  - Individueel reflectieverslag
  - Ontwerp + product maken (mogelijkheden analyseren/afwegen, maken, oplossing/aanpak evalueren)

Kwaliteitscriteria toetsen

- CS is competentie gericht
  - Kennis, skills en attitudes (Merrienboer, 2002)
- Wheel of Competency Assessment (Baartman, 2006)
- Hoe meet je deze criteria?
  - Docenten & leerlingen
  - Betrekken & bekijken
- Te veel werk voor elke docent om zelf doen!
- Daarom belang van goede voorbeelden
- Docenten passen bestaande toetsen aan



Kennis toetsen dmv multiple choice

e) Select the method that would make the most students happy with their club assignment.

☐ Method 1
☐ Method 2

Explain why one method would make more students happy than the other.

- Multiple choice:
  - snel en objectief te beoordelen
  - moeilijker om goede vragen te maken met juiste afleiders
- Combineren met een open-vraag: “leg uit waarom”
  - Leerlingen kunnen niet meer gokken
  - Open vraag van hoger niveau – CT kennis!
  - Docent hoeft alleen na te kijken als mult. choice antwoord goed is

Source: Exploring Computer Science – Unit 2 Assessment: Problem Solving, SRI (2017)

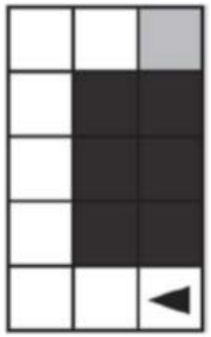
Kun je CT vaardigheden ook met Multiple-Choice toetsen?

Learning Objectives:
Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity.

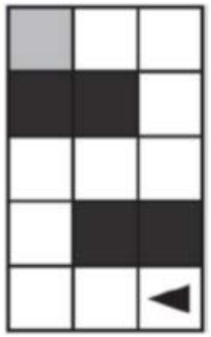
```
REPEAT UNTIL (GoalReached ())
{
   IF (CAN_MOVE (forward))
   {
      MOVE_FORWARD ()
   }
   IF (CAN_MOVE (right))
   {
      ROTATE_RIGHT ()
   }
   IF (CAN_MOVE (left))
   {
      ROTATE_LEFT ()
   }
}
```

For which of the following grids does the code segment NOT correctly move the robot to the gray square?

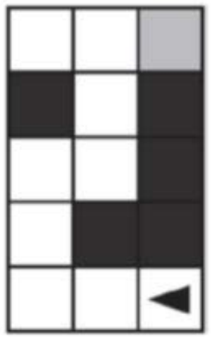
(A)



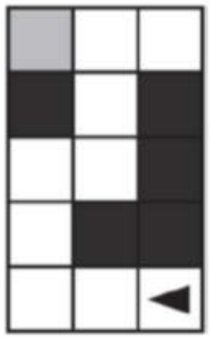
(B)



(C)



(D)

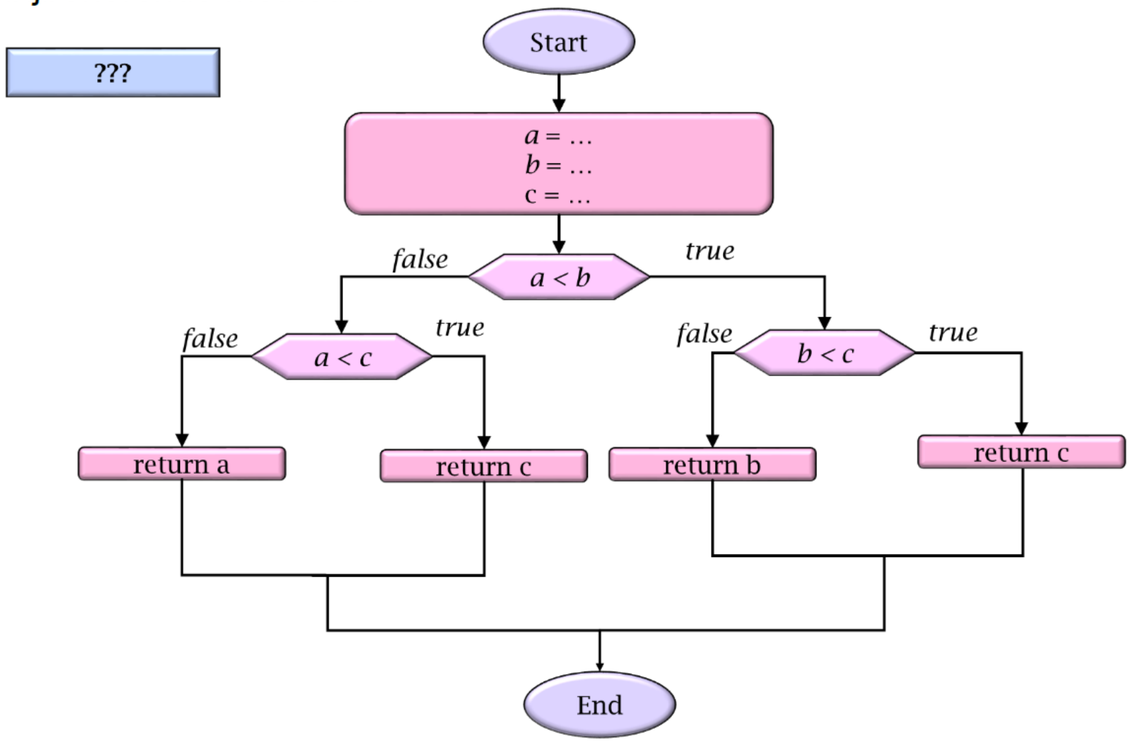


Source: AP Computer Science Principles course and exam description (2017)

Answer: D

Open vragen

Ga na wat het doel is van het volgende algoritme. Geef een geschikte naam voor de bijbehorende methode.



bv. korte programeer vragen

Schrijf een programma dat de maximale waarde uit een lijst afdrukt.

Source: Algorithmic Thinking and Structured Programming in Java, Smetsers and Smetsers (2016)

Parsons Puzzels

We willen een programma schrijven dat de gebruiker om getallen vraagt en deze bij elkaar optelt. Het programma stopt als de gebruiker een 0 ingeeft.

Geef aan welke van de volgende instructies nodig zijn, en in welke volgorde. Je hoeft niet alle regels te gebruiken, en als het nodig is mag je regels vaker gebruiken.

```
A. totaal = totaal + invoer
B. invoer = int( input ("Voer een getal in: ") )
C. invoer = str( input ("Voer een getal in: ") )
D. invoer = input ("Voer een getal in: ")
E. while invoer==0:
F. while not invoer==0:
G. totaal == 0
H. totaal = 0
I. print( totaal )
J. print = int( input ("Voer een getal in: ") )
```

ANTWOORD: H B F A B I

Parsons puzzles

- Voor leerling:
  - Hoog cognitief-niveau: Zelfde niveau als het schrijven van writing code (Ericson et. al. 2017)
  - Laag cognitieve eis: kost minder tijd + inspanning
- Voor docent:
  - Objectief
  - Snel te beoordelen

```
A. totaal = totaal + invoer
B. invoer = int( input ("Voer een getal in: ") )
C. invoer = str( input ("Voer een getal in: ") )
D. invoer = input ("Voer een getal in: ")
E. while invoer==0:
F. while not invoer==0:
G. totaal == 0
H. totaal = 0
I. print( totaal )
J. print = int( input ("Voer een getal in: ") )
```

Charettes (korte programeer opdrachtjes, evt. achter de PC)

Schrijf een programma dat met loop een lijst van getallen doorloopt en de kleinste waarde afdrukt.

*Je mag hierbij de standaard functie min **niet** gebruiken.*

Je kunt deze lijst gebruiken: getallen = [3,8,12,-4].

ANTWOORD:

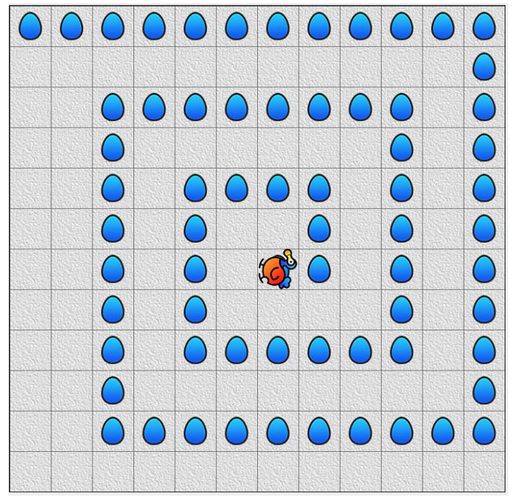
```
min = getallen[0]
for getal in getallen:
        if getal < min:
                min = getal
print( "min is:", min )
```

Tip voor inspiratie:

Informatica Olympiade

Charette

- Specificeer een generiek algoritme waarmee het spiraal-patroon hieronder gemaakt wordt. Generiek betekent dat het voor elke wereldgrootte moet werken. Na afloop staat Dodo op een ei.
- Je moet **zowel** een **stroomdiagram** als **Java** code maken.



Source: Algorithmic Thinking and Structured Programming in Java, Smetsers and Smetsers (2016)

Vaardigheden mbv PO

- **Voordelen**: Samenwerking, meer tijd, meer mogelijkheid om kennis/vaardigheden aan te tonen, problemen op te lossen, debuggen, onderzoeken etc.
- **Nadelen**: mogelijkheid tot plagiaat.
  - Zorg voor een opdracht waarbij standaard knip-plak werk van internet niet mogelijk is, en bij samenwerking dat niemand meelift. Deels af te vangen door reflectie.
- Verschillende combinaties (multi-faceted) liggen voor de hand:
  - Verslag (schriftelijk)
  - Presentatie
  - Digitaal Artefact (code + commentaar in code)
  - Logboek + aantekeningen (code-sniplets, foto's, filmpjes)
  - Interview / mondeling / observaties

CT vaardigheden bij PO: Abstractie

2d. Capture and paste a program code segment that contains an abstraction you developed individually on your own (marked with a **rectangle**). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program.
(Must not exceed 200 words)

```
19 drawFish(randomNumber(30),0,randomNumber(255),250);
20 moveTo(200,200);
21 drawFish(randomNumber(50),randomNumber(255),0,randomNumber(200));
22
23 function drawFish(size, red, green, blue){
24   penRGB(red,green,blue);
25   penWidth(size);
26   penDown();
27
28   dot(size);
29   turnTo(90);
30   moveForward(size);
31
32   turnLeft(30);
33   moveForward(size);
34   turnRight(120);
35   moveForward(size);
36   turnRight(120);
37   moveForward(size);
38   turnRight(120);
39
40   penUp();
41 }
42
```

Look-fors in a student response.

There are several ways a student could talk about managing complexity.

- **code reuse** -- Describe how the parameterized function compartmentalizes (or encapsulates) the drawing of a single fish in a way that lets that code be reused to draw many fish.
- **use of parameters** -- Describes how the parameters allow you to call the function with a single random number as a value and then re-use that value several times throughout the function code. There is no way with a parameter variable, for example, to pick a single random number you want to use as a *size*, and then use that number in multiple places.

A student might write something like:

*"The parameter for *size* in drawFish makes me able to draw a fish of any size I want with some consistency. Without the parameter I'd have to either pick a single size for every fish, or make each of the lines that make up the fish have a random size."**

Source: AP CS Principles Course and Exam Description (2017)

CT vaardigheden bij PO: Algoritmisch Denken

2c. Capture and paste a program code segment that implements an algorithm (marked with an **oval** in **section 3** below) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, and must integrate mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. *(Must not exceed 200 words)*

Source: AP CS Principles Course and Exam Description (2017)

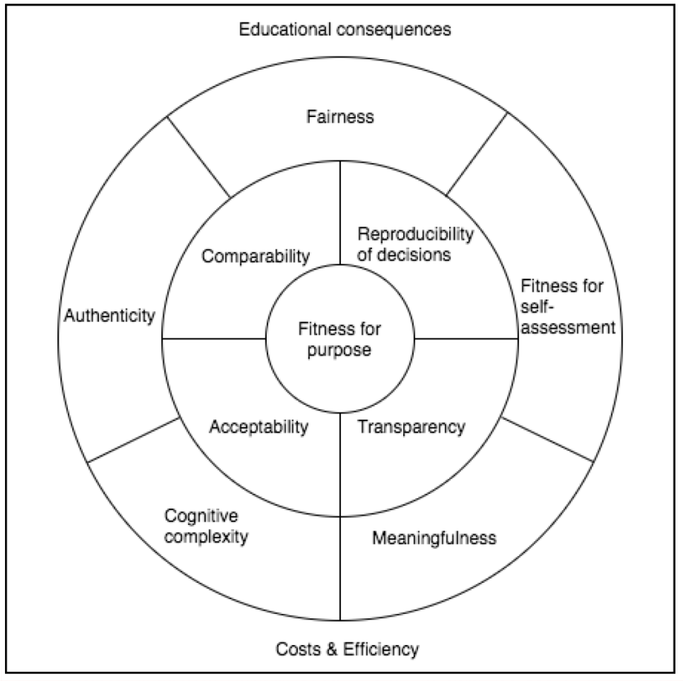
Rubrics

Concept	Limited / No Evidence (0)	Inconsistent Evidence (1)	Strong Evidence (2)	Score
Comments	The program includes no / extremely limited commenting.	The program includes comments but not in all sections. Comments may not effectively clarify the purpose or functionality of the program.	Comments are used consistently. Comments help clarify the purpose or functionality of the program.	
Function and Parameter Names	Function and parameter names do not clearly indicate their purpose. Consistent naming conventions are not used.	Function and parameter names sometimes indicate their purpose. Consistent naming conventions may be used.	Function and parameter names indicate their purpose. Consistent naming conventions are used.	
Functions and Abstraction (Top Down Design)	The program makes limited use of functions. The program does not make use of layers of functions.	The program uses functions but the program may not feature high level and low level functions. There may be missed opportunities to simplify program expression through the use of functions.	Top Down Design clearly used to divide the program into layers of functions. Lower level functions have been further divided into layers when necessary.	
Functions with Parameters	The program does not feature a function with a parameter or the parameters are not used in the function.	A function with a parameter is present, but the parameter is not used in a meaningful way - OR - the function is not called with different values supplied to the parameter (called with the same values every time)	A function with a parameter is present. The parameter controls a meaningful component of the function's behavior. The function is called with different values given to the parameter.	
Loops	The program does not use loops.	The program uses loops inconsistently. There are sections of repeated code that should be placed within a loop.	Loops are used consistently when there is a need to repeatedly run the same block of code.	
Collaboration	Group planning document may be incomplete. In-class communication was limited. Final project may not include code from each member of the team. Comments may not be used to indicate who wrote different sections of the final program.	There is some evidence of effective collaboration. For example the group planning document is complete but in-class communication was weak, leading to program components that do not mesh well.	Group planning guide, classroom participation, and final program code reflect consistent effective collaboration. All team members are assigned significant portions of program. Team members communicated effectively during in-class programming time. Final program includes comments reflecting who completed which sections of the program.	

Source: Code.org CS AP Principles (2018)

Afsluitend

- Informatica is multi-faceted (veelzijdig): concepten, vaardigheden, attitudes
- Veel soorten leerlingen (profielen, gender, toetsvoorkeuren)
- Verschillende aspecten allemaal op verschillende manieren aan bod laten komen:
  - Inhoudelijk (kennis, vaardigheden)
  - Vorm (schriftelijk vs. presentatie, individueel vs. samenwerking)
  - Cognitief (uni/mulitstructural vs begrijp/evalueer/creëer)
- Eigenschappen: transparant, eerlijk, …
  - Welk bewijs wil je zien? In welk situatie kan een leerling dat tonen?
  - Rubrics (bij PO: evt zelf-evaluatie)
  - Toetsmatrijs (van te voren duidelijk hoeveel punten voor elk opgaven)
  - Context (interessant, relevant)
  - …



Vragen, opmerkingen, tips, bedenkingen…

renske.smetsers@science.ru.nl