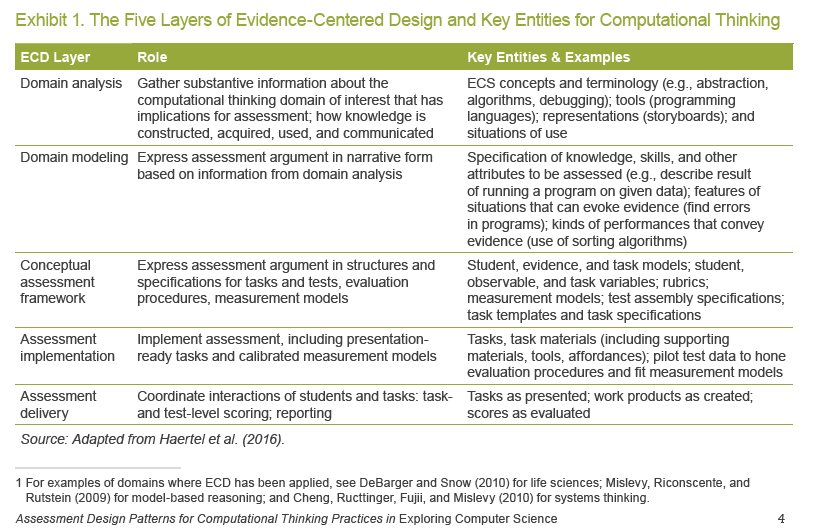
EVIDENCE CENTERED ASSESSMENT DESIGN

Bron van inspiratie 2006Mislevy & Assessment designPatterns for comp thinking (aug 2017)



# Domain Analysis layer

Gather substantive information 
about the computational thinking 
domain of interest that has 
1. Domain 
implications for assessment; how 
Analysis 
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). 

*[2017Snow] 2.3.2 Domain Analysis.* In the domain analysis layer,

assessment developers and domain experts identify and analyze

the domains, constructs, and underlying skills of interest. The

analyses include the ways people acquire and use the knowledge

and skills, the situations under which this knowledge is used,

and indicators of successful application of the knowledge.

Typical sources of information used in domain analysis include

existing domain and construct definitions, content and practice

standards, curriculum documents, relevant research findings,

especially regarding validity, and assessment instruments.

Practitioner judgement is also important. Typical outcomes of

the domain analysis layer include lists of concepts and principles

in the domain(s) being analyzed that are organized as an

assessment framework that lays out constructs, measurement

approaches, and templates that guide task design.

*Primary input is the NL curriculum specification. In addition, relevant topics in the curriculum specification for AP CS/ECS and a mapping to their concrete learning objectives.*

The top layer in the ECD process is a Domain Analysis to clarify learning goals.

This level includes determining relevant concepts and terminology,(e.g. abstraction, algorithms), tools (i.e. programming languages) and representations (flowcharts, pseudocode) as well as understanding how knowledge is constructed, acquired, used and communicated (SRI Assessment Design Patterns for CT, 2017). In the process, the curricular learning objectives are translated into more specific and learning goals specified in operational terms, ready for teacher use.

The domain of interest is programming (domain D), relevant algorithms (domain B) and related skills (domain A), as specified in the Dutch curriculum specification. Together these entail program-relevant concepts and (Computational Thinking) skills.

**Design pattern Algorithms (part of Domain B)**

**Algorithmic Thinking: design and analyze algorithmic solutions**

|  |  |
| --- | --- |
| Design Pattern Attribute | Attribute Content |
| Overview | **This design pattern supports the development of tasks in which students create, analyze, and compare algorithmic solutions. It encompasses steps of problem solving and Computational Thinking. These include understanding the problem, abstracting it from its description, decomposing, recognizing standard algorithms, unambiguously formulating, evaluating and improving a solution prior to implementation. In addition, it encompasses evaluating correctness and efficiency and comparing trade-offs of standard algorithms or algorithms implemented in a digital artefact.** |
| Focal Knowledge, Skills and Attributes | Ability to:  **FKSA 1)** understand an algorithm and its behavior on a set of inputs;  **FKSA 2)** develop a provisional solution for a problem into an algorithm;  **FKSA 3)** recognize and apply standard algorithms;  **FKSA 4)** investigate the correctness and efficiency of a digital artefact via its underlying algorithm. |
| Potential Observations  (basis for rubrics) | **FKSA 1:** **understand an algorithm and its behavior on a set of inputs;**   1. Accurately describe the term algorithm. 2. Accurately hand-trace an algorithm (expressed as a flowchart or pseudocode), use logical reasoning about inputs (preconditions) to predict outputs (postconditions), summarize its behavior and determine if it successfully solves a stated problem.   **FKSA 2: develop a provisional solution for a problem into an algorithm;**   1. Use diagrams (i.e. flowcharts) or pseudocode to express a (digital) solution that meets specified objectives. 2. Design algorithmic solutions using building-blocks: (assignment) statements, selection (one-way, two-way, multi-way), repetition (counter-controlled loop primed sentinel controlled loop), standard algorithms (such as minimum, maximum, count), and apply these in sequence, abutment, and nesting. 3. Use and combine a range of (arithmetic, relational and Boolean logic) operators and expressions, and apply them in the context of program control. 4. Design a solution to a problem that depends on solutions to smaller instances of the same problem (recursion). (vwo level)   **FKSA 3: recognize and apply standard algorithms;**   1. Describe the characteristics of standard algorithms on linear arrays (such as: sequential search, binary search, bubble sort, selection sort) and graphing algorithms (WELKE??: kortste pad, TSP, minimal spanning tree, dominating sets, map-coloring problem) 2. Recognize and apply standard plans (such as determining the minimum, maximum or count) and standard algorithms those for searching, sorting, data-compression and graph algorithms (such as routing). 3. Recognize and apply at least two different standard solutions for solving the same problem, and compare tradeoffs between each algorithm.   **FKSA 4: investigate the correctness and efficiency of own or standard (i.e. search, sort, graph, min, max, count) algorithms and of a digital artefact using its underlying algorithms.**   1. Investigate the correctness of an (own or standard) algorithm. 2. Reason about an algorithm’s (in particular a standard algorithm) efficiency for a given problem (and with different data sets - sizes, sorted, unsorted, inverse), discriminate between running in a reasonable time and not in a reasonable time, and compare its efficiency to an alternative algorithm for the same problem (Big O notation is not required). 3. Examine the behavior of an existing program through the underlying algorithm, and relating problems (e.g. errors, inefficiencies, ambiguity, exceptions/preconditions) to its underlying algorithm. 4. Model a digital artefact using an algorithm. |
| Potential Work Products  (what student produces) | **FKSA 1:** **understand an algorithm and its behavior on a set of inputs;**   1. Definition of the term algorithm 2. Evaluation of the result of a flowchart and/or pseudocode (through hand-tracing). 3. Identification of an error in a flowchart and/or pseudocode, and correct the solution. 4. A high-level goal summarization of an algorithm and evaluation of fit-for-purpose of an algorithm to solve a stated problem, such as for which general inputs it is valid (including boundary cases) and prediction of output in general terms (i.e. not tracing for specific values but rather generalizing for a set of inputs and outputs).   **FKSA 2: develop a provisional solution for a problem into an algorithm;**   1. Construction of a solution using a flowchart and/or pseudocode that addresses a set of specifications. (for vwo level: including recursion)   **FKSA 3: recognize and apply standard algorithms;**   1. Identification of an error in the flowchart and/or pseudocode of standard algorithm (such as determining the minimum, maximum or count) and provision of a corrected solution. 2. Combine two or more standard algorithms to create a solution for a problem.   **FKSA 4: investigate the correctness and efficiency of digital artefacts using the underlying algorithms.**   1. Discuss an algorithm to solve a specific problem. Students are expected to compare trade-offs between different algorithms for solving a stated problem. The algorithms include both standard and novel algorithms. For example, discussing the advantages and disadvantages of using a binary search as opposed to a sequential search. 2. Evaluate the fitness for purpose of algorithms in meeting requirements efficiently using logical reasoning and test data. Reason about the complexity, for example execution-time or number of steps of an algorithm: what is the relation with respect to the size of the input dataset (i.e. what happens when the dataset is doubled?). For example: An algorithm which counts how often a word appears in a text on the fly requires as many steps as the number of words in the text. An alternative algorithm is to first store the words in the text, one by one, and then to perform a search. The new algorithm is still correct, but each word is read twice making the new algorithm less efficient. 3. Determine best-case, worst-case and average number of steps and relating these to the data format. Examples are the effect of doubling the input size or state of the input data (such as sorted, unsorted, or inversed in advance). 4. Examine and identify optimizations of algorithms in terms of efficiency. Understand and explain the difference in efficiency between a single loop, nested loops, a loop that ends when a condition is met or questions of similar complexity. Suggest changes in an algorithm to improve efficiency such as using a flag to stop a search. |
| Characteristic features | **CHARACTERISTIC FEATURES:**  The student must be presented with an activity.  The task must <SEE ASS.DESIGNPATTERSNFORCOMPTHINK…ECS>  *…* |
| Variable features | **FKSA 1: understand an algorithm and its behavior on a set of inputs;**  **FKSA 2: develop a provisional solution for a problem into an algorithm;**  **FKSA 3: recognize and apply standard algorithms;**  **FKSA 4: investigate the correctness and efficiency of a digital artefact via its underlying algorithm.**  Variable features:   * Varying on level according to Solo’s taxonomy (unistructural, multistructural, relational) the scope/complexity of the solution in terms of sequence/nesting/abutment of constructs, plans and combinations of (arithmetic, relational and Boolean logic) operators and expressions. * Varying the choice of plan required or used to solve the problem determines the required level of conceptual knowledge, in addition to varying in the cognitive load. For example, recognizing and applying a count-plan is less challenging and requires less application of (CT) practices such as abstraction and decomposition, then an average-plan. * Varying the way in which the algorithm is presented, such as in natural language, pseudocode or flowchart. * Vary the (range of) input data (structures), such as including negative integers, decimal values, using an arrays or involving user input values. The later requires the student to realize that a large range of input is possible, which are acceptable, handle unexpected cases (such as providing an error text and allowing the user to re-enter a value), and how to cast types to a format for further processing. * Varying in the (number of) output data, or the type of output data. For example, printing an integer to the screen or returning an array with particular values. * Degree of scaffolding provided in the task (for example, providing steps or hints) determines the level of cognitive load. |

Notes on differences between NL curriculum and other curricula:

In NL but not other curricula:

* Use an algorithm to model a digital artefact.
* Investigate the correctness and efficiency of an digital artifact by means of the underlying algorithm.

In other curricula but not in NL curriculum:

* (in AP CS) Explain the difference between algorithms that run in a reasonable time and those that do not run in a reasonable time (i.e. number of steps)

**Design pattern Programming (domain D)**

**Programming: design and implement creative solutions and digital artifacts**

|  |  |
| --- | --- |
| Design Pattern Attribute | Attribute Content |
| Overview | **This design pattern supports the development of tasks in which students expand their knowledge of algorithms, abstraction and apply it in the creation, inspection, and adaption of program components for a given objective. Students will be able to translate designs into functional components, apply coding conventions and abstraction to structure components, increasing readability, re-usability, testability and adaptability of a program (by others).** |
| Focal Knowledge, Skills and Attributes | Ability to:   1. develop robust program components for a particular goal, in an imperative programming language; 2. use programming language constructs that support abstraction; 3. structure a program component in such a way that they can be easily understood and evaluated by others; 4. explain the structure and functioning of certain program components; 5. adapt such program components based on evaluation or changing requirements. |
| Potential Observations | **FKSA1.** develop program components for a particular goal, in an imperative programming language. The program can constitute compositing a new algorithm, or remixing and/or applying standard algorithms.   1. Convert an algorithm into high-level (imperative language) program code. 2. Design and develop program components for a particular objective in a certain context, taking technical, environmental and human factors into account. 3. Design, code, systematically test (determine correctness of the program in terms of input and output), debug (locate and correct), adapt (extend), and execute a program that corresponds to a set of specifications or requirements. 4. Describe and apply the concept of a data type and use the following appropriately: integer, real/float, Boolean, character, string, date/time, records, one dimensional data structures (arrays or equivalent), and two dimensional data structures. 5. Declare, initialize and assign values to datatypes in a programming language. Such as variable declaration, constant declaration, assignment. 6. Describe, apply and combine the operators and expressions in the context of program control/program components: arithmetic, relational, Boolean logic. 7. Use standard operations on datatypes in a programming language. Examples for string-handling are: casting conversion operations, length, position, substring, concatenation (or similar for collections or arrays). 8. Describe, apply and combine the following by abutment, nesting and merging: statements, iterations (counter-controlled loop and primed sentinel controlled loop), selections(single-branch or multi-branch), subroutines (procedure/function). 9. Design solutions by decomposing a problem and create a sub-solution for each of these parts. (RELATED TO CT-DECOMPOSITION) 10. Be able to use subroutines/functions that return values to the calling program component. Be able to use parameters to pass data.   1.1.28- examenprograamen a8 ROBUUST  **FKSA2.** Use programming language constructs that support abstraction;   1. Define and use user-defined data types based on language-defined (built-in) data types. Data abstraction provides a means of separating behavior from implementation, use of ADT and their operations, API/libraries. 2. Use abstraction to manage complexity in programs. 3. Knows that a procedure can be used to hide the detail with sub-solution. (AL) (DE) (AB) (GE). 4. Appreciates the need for, and writes, custom functions including use of parameters and return values (AL) (AB). Understand scope: Know that subroutines may declare their own local variables, use local variables. 5. Data abstraction provides a means of separating behavior from implementation, use of ADT and their operations, API/libraries. 6. Be able to describe the use of parameters to pass data within programs.   **FKSA3.** Structure a program component in such a way that they can be easily understood and evaluated by others;   1. insightful, verifiable and maintainable 2. Use meaningful identifier names. Explain the advantages of using named constants. 3. Uses modular components which can be tested individually??   **FKSA4.** Explain the structure and functioning of certain program components;   1. Describe how a program works. (Use an algorithm to describe a digital artefact (REVERSE ENGINEERING?)???)   **FKSA5.** Adapt such program components based on evaluation or changing requirements. |
| Potential Work Products | *Applying code conventions and abstraction to structure components, increasing readability, re-usability, and adaptability of a program (by others).*  *1a.*  Explain how programs implement algorithms. (algorithms are implemented using program instructions, simple algorithms can solve a large set of problems, improvements in algorithms can increase the kinds and size of problems solvable by programming)  1ab: Define the terms: variable, constant, operator, object.  1ab: Define the operators =, ≠, <, <=, >, >=, mod, div.  1ab: Employ appropriate mathematical concepts, Boolean algebra, arithmetic operators, compound expression using and, or and not)  1ab: Composition: Know how to build a composition abstraction by combining procedures to form compound procedures.  1ab. Be able to use subroutines/functions that return values to the calling program component.  1ab. Declare and use data objects (variables and constants) and assignment to assign values to variables  *1c. Evaluate the correctness of a digital artifact (describe its functionality at a high level, intended behavior, debugging, program style i.r.t. correctness, inputs, justification can include a written explanation about how a program meets its specifications).*  *1a,b,c. SIMPLE Develop a computer program involves: ● writing code for a program that performs a specified task using a suitable programming language ●setting out the program code clearly ●documenting the program with comments ●testing and debugging the program to ensure that it works on a sample of expected cases.*  1abc. BEGINNER. Construct an advanced computer program for a specified task involves: • implementing a plan for an advanced program in a suitable programming language • setting out the program code clearly and documenting the program with comments • testing and debugging the program to ensure it works on a sample of expected input cases.  VS  Develop a complex computer program for a specified task involves: • designing and implementing a program that includes variables, an indexed data structure, and a modular structure including details of the procedural structures of the modules • including a working graphical user interface with different sources of event generating components and event handling, and using class(es) and objects to encapsulate data and methods • setting out the program code clearly and documenting the program with comments • testing and debugging the program to ensure it works on a sample of expected input cases.  1abc INTERMEDIATE. Skillfully construct an advanced computer program for a specified task involves: • independently implementing a plan for an advanced program in a suitable programming language that uses well-chosen scopes for variables, and well-chosen parameters for modules • documenting the program with variable and module names and comments that accurately describe code function and behavior • testing and debugging the program in an organized way to ensure it works on inputs that include both expected and boundary cases.  VS  Skilfully develop a complex computer program for a specified task involves: • documenting the program with variable and module names and comments that accurately describe code function and behaviour • following a disciplined design and implementation process, with documented cycles of incremental development and comprehensive testing process, to ensure that the program works on inputs that include both expected and boundary cases.  1abc ADVANCED. Efficiently construct an advanced computer program for a specified task involves: • constructing an advanced program where the modules (including their procedural structures) constitute a well-structured logical decomposition of the task • using variables, constants, and derived values effectively so as to increase the flexibility and robustness of the program • setting out the program code concisely and documenting the program with comments that explain and justify decisions • comprehensively testing and debugging the program in an organised and time-effective way to ensure the program is correct on expected, boundary and invalid inputs.  VS  A complex computer program is one that has a modular structure; an indexed data structure (e.g. array or list); input and output; procedural structures that combine sequential, conditional, and iterative structures; a graphical user interface and event handling; and that includes class(es) and objects. Inheritance is not required.  *<See above for variable features>*  *1c. Testing and debugging a program: Identify the inputs and outputs (using concrete examples) required in a solution and relevant boundary cases.*  *1d Also test using unexpected input (related to human factors)*  *1c. Evaluate programs analytically, for efficiency, correctness, and clarity.*  *1e. Variable features: compound expressions using AND, OR, NOT*  *2.* *A program with a modular structure contains a collection of named modules (procedures, functions, methods, or subroutines) where each module implements a procedural structure for a sub-task. At least the top-level module (and possibly others) must contain calls to other modules. The modules should include parameters as needed.*  2a. Analyze the use of variables, constants, and operators in algorithms. (i.e., justify the use of a constant as opposed to a variable)  Knows the difference between, and uses appropriately procedures and functions. (AL) (AB)  1f make appropriate use of data structures  2. Use multiple levels of abstraction to write a program (i.e., constants, expressions, statements, procedures, libraries/modules)  2. design and develop modular programs that use procedures or functions  *Understands and applies parameter passing. (AB) (GE) (DE)*  *2. (evaluation) Explain the role (and advantages) of sub-procedures in solving a problem.procedures (methoden, subroutines, …) voor stukken programma die iets doen; Explain why abstraction is required in the derivation of computational solutions for a specified situation.*  *Appreciates the effect of the scope of a variable e.g. a local variable can’t be accessed from outside its function. (AB) (AL).*  *3. Use and develop an abstraction effectively, to appropriately structure programs into modular parts with clear, well-documented interfaces*  *(includes: generalize concepts from specific examples, parameters)*  *3. Example of ways of making a program flexible and robust include: ● using methods, functions, procedures, actions, conditions and control structures effectively ● checking input data for validity ● correctly handling expected, boundary and invalid values ● using constants, variables and derived values in place of literals.*  *3.*  *Efficiently develop a complex computer program for a specified task involves: • ensuring that the overall modular and procedural design, graphical user interface, and event handling design, are a well-structured, logical decomposition of the task • using variables, constants, and derived values effectively so as to increase the flexibility and robustness of the program • comprehensively testing and debugging the program in an organised and time effective way to ensure the program is correct on expected, boundary and invalid input cases* |
| Characteristic features | **CHARACTERISTIC FEATURES:**  The student must be presented with an activity or task/specifications.  The task must include the creation of a digital artefact.  The task must include a set of specifications that can be used to generate an algorithm.  The task must involve creating a novel algorithm and abstraction.  The task must involve evaluation of the digital artefact in terms of efficiency, correctness, and clarity.  *…* <SEE ASS.DESIGNPATTERSNFORCOMPTHINK…ECS PG39 + P10>  *Vragen:*  *students individually were*  *given a certain robot programming task and were asked to describe*  *aloud the process they would follow to solve it. Simultaneously, the*  *researcher prompted students to reflect on CT concepts relevant to*  *their solution. The assessment of the student’s proposed solution*  *was based also on the same graded criterion instrument (rubric)*  *as before. The main difference is that the*  *‘think-aloud’ method allows students to express their thinking*  *more freely as opposed to the highly structured form of the*  *questionnaire instruments.* [source: 2015Atmatzidou]  CT skills and relevant prompts to trigger students’ self-reflection.  **Abstraction** What is common in robot behaviour in both programs? How would you describe this common behaviour? What is the common  programming structure? Which is the information you actually need? What is irrelevant detail and not necessary in your description?  **Generalisation** Propose a more general solution for the activity above, that can cover a wider variety of cases. Is the proposed solution more general and  why?  **Algorithm** Write step-by-step the operations needed so that the robot can do what the problem asks. What are the steps I will need to do to solve this  problem?  **Modularity** Are there any parts of the code that you have met before? Have you created your own blocks for these? What are they? Do you expect to  need some parts of this particular code in the future or in a different problem?  **Decomposition** Can I break down this complex problem into smaller ones? Can I solve and explain the smaller problems, building up a solution towards the complex problem? |
| Variable features | *…* <SEE ASS.DESIGNPATTERSNFORCOMPTHINK…ECS PG40>  GENERAL RUBRICS AND VARIABLE FEATURES:   |  | | --- | | *Construct a plan for an advanced computer program for a specified task involves: • specifying variables, their scopes and data types • specifying an indexed data structure • specifying a modular structure for the program, including details of the procedural structures of the modules • specifying a set of expected input cases for testing the program.* | | *Skillfully construct a plan for an advanced computer program for a specified task involves: • independently constructing the plan • specifying well-chosen scopes for the variables • specifying well-chosen parameters for the modules • specifying a set of expected and boundary input cases for testing the program* | | *Efficiently construct a plan for an advanced computer program for a specified task involves: • specifying modules (including their procedural structures) that constitute a well-structured logical decomposition of the task • specifying variables, constants, and derived values effectively so as to maximize the flexibility and robustness of the plan • specifying a comprehensive set of expected, boundary and exceptional input cases for testing the program.* | | An advanced computer program must have a modular structure, an indexed data structure (e.g. array or list), input and output, and procedural structures that combine sequential, conditional and iterative structures. | | The scope of a variable may be global (accessible from all modules) or limited to a single module. A well-chosen scope matches the way the variable is used. | | Well-chosen parameters for modules are those where the number and types of parameters are decided in the context of the module’s sub-task. | | Constants should be used as required when a value never changes. Derived values are returned properties or are calculated from other values. Examples include but are not limited to: the length of an array or string; area which is calculated from the width and height of a rectangle; and the mid-point of a graphics object which is calculated from its width and height. | | In a well-structured logical decomposition of the task, each module will have a clear and well-defined purpose within the context of the task. The interaction between modules will be minimized, modules will be reused rather than duplicated, and the procedural structure of each module will be efficient.  PRESENTATION/ORAL REPORT/WRITTEN REPORT (combined with in-class observations??) | |

**EXAMPLE TASK**

The potential observations and related work products call for a mult-facetted examination. In particular, capturing the creative aspects of Computational Thinking and problem solving skills require a different assessment tool to theoretical knowledge (see the analysis on work products). Two types of products are proposed:

1. Task (in and out of class activities over a period of time specified by the teacher, creative design report (written or as presentation- as specified by the teacher), may include coaching (SEE THE RULES FOR CS AP project).
2. Written exam (set-time, in-class, under exam conditions).

**ALGORITHMS**

The majority of the potential work products have been accounted for in the following tasks:

|  |  |  |  |
| --- | --- | --- | --- |
| Assessment objective | Written exam algorithms | Algorithms Task | Overall |
| 1. Demonstrating knowledge and understanding |  |  | 40 |
| 2. Applying and using |  |  | 30 |
| 3. Constructing, analyzing, evaluating and formulating |  |  | 20 |
| 3. Using skills |  |  | 10 |
|  |  |  |  |

**PROGRAMMING**

The majority of the potential work products have been accounted for in the following tasks:

|  |  |  |  |
| --- | --- | --- | --- |
| Assessment objective | Written exam programming | Programming Task | Overall |
| 1. Demonstrating knowledge and understanding |  |  | 40 |
| 2. Applying and using |  |  | 30 |
| 3. Constructing, analyzing, evaluating and formulating |  |  | 20 |
| 3. Using skills |  |  | 10 |
|  |  |  |  |

The overall weighting has been given as an indication. These weights have been taken from IB (50, 20, 20, 10), and CS and NZ.

Note that not all potential work products have been accounted for, but rather XX%, which is generally a good rule-of-thumb for a fair task (see the analysis on characteristics for qualitative assessments).

**Written exam algorithms:** theoretical knowledge of algorithms, problem solving, theory of computation, understanding implication of algorithms and analysis of correctness and efficiency

**Written exam programming:** ability to program, theoretical knowledge of programming, *data structures*, problem solving, *theory of computation*

**Task algorithms:** ability to use the knowledge and skills gained through the course to solve or investigate a practical problem. Designed project (see analysis work products). Students will be expected to follow a systematic approach to problem solving.

***WHY EFFICIENCY IS IMPORTANT FOR COMPARING MULTIPLE ALGORITHMS (AND THUS ADD ONTO THE AP CS Principles REPORT:*** *The efficiency of an algorithm is determined by the complexity measures needed to perform it, the most important of which are memory space and time. In many cases, more than one algorithmic method can solve a specific problem. By analyzing the efficiency of algorithms, different solutions to the same algorithmic problem can be compared and the most efficient algorithm identified. If a problem is not analyzed, and in the absence of planning, the algorithm could be too expensive, and thus useless. That is, it would be impossible to run the algorithm in practice within a reasonable amount of time (the myth concerning the computer’s speed notwithstanding). Teaching algorithm efficiency at a relatively early stage makes it possible to expand the range of algorithmic methods, plan algorithms, evaluate alternative methods and perform analyses [5].[*BRON: *2002 Gal\_ezer Algorithm Efficiency]*

**Task programming:** ability to use the knowledge, and (Computational Thinking and problem solving) skills gained through the course to **create** a digital artifact. Open-ended project (see analysis work products). Students will be expected to use an iterative process. Steps include planning (creating a flowchart), implementing, test, evaluating (correctness, efficiency, elegance) followed by reporting (how and why - describing the used algorithms, techniques, and reflection). Evidence may be generated from discussion, group work, decision making and/or reflection, teacher observation of procedures and the presentation of the final product.

The evidence generated will collectively demonstrate the student’s ability to implement procedures to produce a digital artifact.

SEQUENCING OF ASSESSMENTS:

Evidence from two types of assessments (written exam and task) collectively contribute the student’s demonstration of his or her ability. Though both are intended to be used as summative, the written exam can also be given in advance and be used formatively. Students can receive feedback on their knowledge levels and the learning. This may help pinpoint any gaps in knowledge or process, indicating what a student can work on prior to starting the task. In addition, it can help determine what kind of a task the student should do in order to ensure a success-experience and be able to generate substantial evidence of the student’s abilities (in contrast to a project which is too large or complex and with which the student does not generate any substantial evidence at all).

TEACHER ROLE: See NZ Assessment eample 91073 (OneNote resource example questions)

Description of the intended learning objectives in operational terms

The goal of Domain Analysis is to gather substantive information about the domain of interest that has implications for assessment. This level includes determining relevant concepts and terminology,(e.g. abstraction, algorithms), tools (i.e. programming languages) and representations (flowcharts, pseudocode) as well as understanding how knowledge is constructed, acquired, used and communicated (SRI Assessment Design Patterns for CT, 2017).

The curriculum learning objectives have been specified by the curriculum committee as follows [Barendsen & Grgurina, 2016]:

Domain B: Algorithms

***Algorithms***

The candidate is able

a) to develop a provisional solution for a problem into an algorithm;

b) to recognise and apply standard algorithms;

c) to investigate the correctness and efficiency of digital artefacts using the underlying algorithms.

Domain D: Programming

***Developing***

The candidate is able, for a given objective,

– to develop program components in an imperative programming language;

– to use programming language constructs that support this abstraction;

– to structure a program component in such a way that they can be easily understood and evaluated by others.

***Inspecting and adapting***

The candidate is able

– to explain the structure and functioning of certain program components;

– to adapt such program components based on evaluation or changing requirements.

Domain A: Skills

***Using informatics as a perspective***

In contexts, the candidate is able to

– indicate, interpret and explain phenomena in terms of informatics;

– recognise and interconnect informatics concepts;

– estimate the possibilities and limitations of digital artefacts and reason about these in terms of informatics concepts.

This learning goal contains elements of *computational thinking.* This includes *analytical skills* to formulate problems in such a way that one can use computers and other tools to help solve them, as well as *problem solving skills*, such as finding solutions in terms of algorithms and data.

## Concepts, CT skills and Competencies

The following aspects are relevant for assessment.

* Concepts: learning objectives distilled from domain B/D in the curriculum
* CT skills: algorithmic thinking, abstraction, decomposition, generalization, evaluation (Wing, 2006)
* (21st century) Competencies:
* Domain A also entails competencies. Competencies have been described by Weinert (2001) as the cognitive abilities and skills enabling students them to solve particular problems and the capacity to use the solutions successfully in variable situations.

‘the cognitive abilities and skills possessed by or able to be learned by individuals that enable them to solve particular problems, as well as the motivational, volitional and social readiness and capacity to use the solutions successfully and responsibly in variable situations’ (pp. 27–28)(Weinert, 2001)

The combination of these different elements – cognitive ability and skill, motivation and readiness, and capacity to use –make competencies much more wide-ranging and complex than learning objectives, but give us the potential to describe and assess our subject in a more comprehensive way. (Hubweiser & Sentence, 2018)

## Operational definitions

### Operational definition of Computational Thinking

This level includes determining relevant concepts and terminology,(e.g. abstraction, algorithms), tools (i.e. programming languages) and representations (flowcharts, pseudocode) as well as understanding how knowledge is constructed, acquired, used and communicated (SRI Assessment Design Patterns for CT, 2017).

RELEVANT CONCEPTS

RELEVANT TERMINOLOGY (e.g. abstraction, algorithms, standard plan)

Algorithm, instructions, sequence, selection, repetition, recursion, variables

Plans: De Raadt

TOOLS (e.g. programming languages) –

REPRESENTATIONS (flowcharts, pseudocode)

Flowcharts, pseudocode

• Uses diagrams to express solutions. (AB)

• Creates programs that implement algorithms to achieve given goals. (AL)

HOW KNOWLEDGE IS CONSTRUCTED. ACQUIRED, USED AND COMMUNICATED

See background info.. includes neo-piaget, comprehension vs. composition, progression pathways, misconceptions

Domain A in the curriculum specifically mentions Computational Thinking in relation to analytical and problem solving skills needed for problem formulation and finding solutions in terms of algorithms and data. There is, however, no general consensus on the definition of Computational Thinking (Yadav, 2015). Voogt et al (2015) call for a flexible and pragmatic approach and use operationalization of CT. We define an operational definition based on similarities and relationships in the discussions about CT (see theoretical background) and their descriptions in related curricula. Thus in particular, during this stage an operational definition of Computational Thinking was established. The five iterative steps to problems solving described by McCracken et al. (2001) and rooted in the definition and have been linked to CT skills (algorithmic thinking, abstraction, decomposition, generalization, evaluation) described by Wing (2006):

DEFINITION CT:

e definition of computational thinking still remains contested as no dominant discourse reigns ().

See Corerandini et al 2017 Conceptions … CT

(1) Abstract the problem from its description - Identify and extract the **relevant aspects** from the problem description (Lee et al., 2011), then **model** the elements in an abstract framework (such as a flowchart). (CT Abstraction)

(2) Generate sub-problems - **Decompose** the design (determine methods and sub-methods). (CT Decomposition)

(3) Transform sub-problems into sub-solutions - Determine an **implementation strategy** for individual classes, methods, appropriate language constructs as well as data structures and programming techniques. **Recognize and apply** standard algorithms. (CT Abstraction and Generalization)

(4) Re-compose the sub-solutions into a working program - Automation: **Combining** (Abutment, nesting and merging) **sub-solutions** from the previous step into a working program / **complete solution.** (CT Algorithmic Thinking)(Lee et al., 2011)

(5) Evaluate, reflect and iterate – Justify the appropriateness of a solution and explain how it works. Analyze **correctness** and **efficiency** of the proposed solution to the problem (Lee et al.,2011) (CT Evaluation), locate and correct errors, and take appropriate action to **improve** the solution and correct any identified design or implementation faults or invalid assumptions (Lee et al., 2011).

The operational definition of Computational Thinking and the teacher-oriented learning objectives were reviewed by the panel. The panel was specifically asked to consider whether the definition correctly adhered to the (intentions of the) curricular specification and were furthermore correct and complete.

*TODO: A panel of experts was* carefully selected. The group comprised of educators, practitioners and experts from multiple institutions (two high schools and two universities), as well as experts that have been involved in conceiving the reformed Dutch curriculum. *(Martin, Erik, Jacqueline, Sjaak)*

*This panel reviewed the operational definition of Computational Thinking and the teacher-oriented learning objectives for correctness and completeness. They were specifically asked to consider whether the definition correctly adhered to the curricular specification.*

# Domain Modeling layer

*[2017Snow] 2.3.3. Domain Modeling.* In the domain modeling layer,

developers organize information from the domain analysis using

a design pattern [34]. Design patterns are structured narratives

that fully lay out an assessment argument. They specify (1) the

knowledge, skills, and other attributes one wants to address, (2)

potential observations and work products that can provide

evidence about acquisition of this knowledge or skill, (3)

potential rubrics to evaluate student performances on the tasks,

and (4) features of task situations that enable the student to

provide this evidence. When completed, a design pattern makes

the relationships among these elements of an assessment

blueprint explicit.

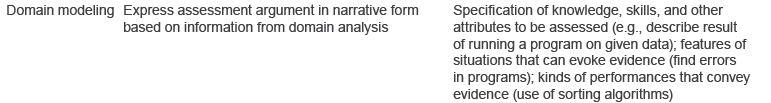
The goal of the domain modeling layer is to express an assessment argument in narrative form based on the information form the domain analysis layer. The assessment argument is a specification of knowledge, skills, and other attributes to be assessed.

In the next layer, Domain Modelling, the assessment argument was designed. The learning goals were translated into the specific focal knowledge, skills, and attributes (FKSAs) to be assessed.

As a starting point, assessment arguments and their correlated FKSAs from the selected relevant international curricula were reviewed for common aspects.

In addition, concepts recognized in relevant publications were considered (i.e. Sheard et al, 2011).

As part of this stage in ECD, the resulting FKSAs and evidence-eliciting-performances were prone to panel review. The main goal was to ensure that the resulting FKSA’s do indeed match the learning objectives, are complete, and adhere to a classroom situation. Feedback was used to fine-tune the FKSAs and were adjustments were reviewed again.



[source: sri 2017 assessment design]

Discuss discrepenties:

Recursion is part of NL curr (VWO) but not others, while it is one of Wing’s list.

Organize according to categories of from Corrandini et al (2017): mental processes, methods, …

## Intended learning goals

### Computational Thinking: intended learning goals

EVIDENCE

• Understands the difference between, and appropriately uses if and if, then and else statements. (AL)

• Uses a variable and relational operators within a loop to govern termination. (AL) (GE)

• Selects the appropriate data types. (AL) (AB)

• Appreciates, writes and can justify the need for, and writes, custom functions including use of parameters. (AL) (AB)

ECS: summarize the behavior or purpose of an algorithm

|  |  |  |
| --- | --- | --- |
|  |  |  |

Explain how a particular program functions.

*TODO: A panel of experts were asked to review the test specification, and an initial draft of sample questions to concretize the testing constructs. The provided feedback can be incorporated into an improved version.*

# Conceptual assessment framework layer

*[2017Snow] 2.3.4. Conceptual assessment framework.* Another layer in ECD

is the conceptual framework, and developing this requires a

designer to specify the constructs of interest (what knowledge

and skills) and the activity to be performed. These two parts are

linked by an evidence model that specifies how student

performances are evaluated to yield observed data, and also by

measurement models that connect the observed data to the

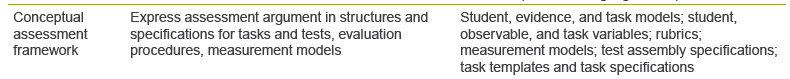
constructs of interest.

Other layers in ECD encompass additional work related to

assessment design and development, including the specification

of the tasks, the development of models to collect and analyze

evidence and assessment delivery.



[source: sri 2017 assessment design]

Evidence is obtained by deliberately putting students in situations, challenges, or tasks that will elicit the desired FKSAs (Grover & Bienkowski, forthcoming). In ECD’s Conceptual Assessment Framework layer, we analyzed which characteristic features of tasks and potential observations could elicit the evidence-behaviors for the specific FKSAs and define suitable model assessment tasks. \todo{ Specifiek: voor elk leerdoel uit elk repository 10 assessments paken.}. tasks. \todo{ Hoe… bij elk leerdoel meerdere alternatieven.. ook even misconceptions noemen als extra tasks }

Input for the potential observations were assessment strategies as implemented by ECS, AP CS principles, International Baccalaureate Computing Science SL/HL, and NZQA Digital Technologies (New Zealand Qualifications Authority) and AQA AS/A. Mappings from learning objectives to assessments were reviewed.

Analysis of the performances which could elicit relevant evidence proficiency for each FKSA was established. A framework of three models originated:

* Student model: FKSAs were selected (and in some cases combined) to cover particular learning objectives;
* Evidence model: measurement model describing how tasks are to be scored and what the scores mean;
* Task model: describing the number and types of tasks to be included as well as specific requirements of the assessment (such as the format of the items (digital, paper and pencil, open/closed book) and the amount of time that must be allocated for completion.

Again, in an iterative fashion, the intermediate results were prone to panel review to ensure that the assessment tasks adequately assessed the objectives, and in addition were furthermore practical and deployable in the classroom situation.

The result of this phase was a design pattern consisting of three aligned models which together form the assessment specifications: the foundation for the new assessment.

**MISCONCEPTIONS:**

**Programming**: zie sorva 2002/2003?

**Algorithm efficiency:**

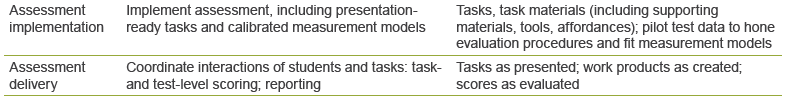
\todo{cite}[2002 Gal\_ezer Algorithm Efficiency] assessed high school students' misconceptions about algorithm efficiency. They found that students mistakenly relate the efficiency of an algorithm to its length or amount of variable used. Furthermore they incorrectly assume that if two programs accomplish the same task they are equally efficient.

EXAMPLE QUESTIONS ARE GIVEN IN: }[2002 Gal\_ezer Algorithm Efficiency]

FROM 2013Zur-Burguray ISRAEL DO SIMILAR YOURSELF

Before the exam’s questions were composed, an exam map was drawn up including subjects and thinking levels according to Bloom’s Taxonomy [2] and its revision [1]. The subjects consisted of serial execution (8%), conditions (30%), forever / forever-if (12%), and for loop (50%). The thinking levels were taken in pairs: Remembering & Understanding (~50%), Applying & Analyzing (~20%), and Evaluating & Creating (~30%).

# Assessment implementation



[source: sri 2017 assessment design]

The assessment layer encompassed the iterative implementation of assessment tasks and piloting the procedures.

First, the design pattern was used to create an assessment, complete with a scoring model. Through an iterative process, the assessment underwent panel-review and was piloted in the assessment delivery layer.

Subsequently, participatory design way used in which teachers were involved in, and played an active role in creating assessments. Teachers were given the design pattern and accompanying example and were asked to create a personalized assessment of their own. They were asked to jot down the process, how they approached it, how much time they spent, if they encountered any problems, what they did to overcome these hurdles, and any suggestions for improvement. The resulting assessment was analyzed according to the quality criteria established in the first study: characteristics typify a qualitative assessment for algorithms. As a part of this, mapping to the relevant FKSAs was performed, as well as noting the use of variable features of tasks (difficulty, contexts), and any other particular adjustments made or characteristics added.

# Assessment Delivery

In the bottom layer of the ECD framework, students actually performed the assessment and their work was evaluated and scored.

Data was collected from several sources:

* For validation purposes, think-aloud sessions were held with the students. The goal was to ensure that the intended objectives were being assessed, and at the intended level.
* A sample of student work was scored according to the rubric. To evaluate objectivity, the scores were compared to those given by the teacher.
* In a semi-structured interview, the teachers were asked about their approach and experience using the design template. Topics considered were ease-of-use, use-for-purpose and their perception on the learning curve, and if they had any suggestions for improvements.. The teachers were asked to convey the notes they had made during the implementation step.

The process and results were analyzed and evaluated and provided input for the following iteration in which the design template was refined.

After the pilot has been performed, the candidate template assessment tasks be deployed at multiple institutions for full-scale testing. The results will be analyzed and evaluated, and the instrument refined.