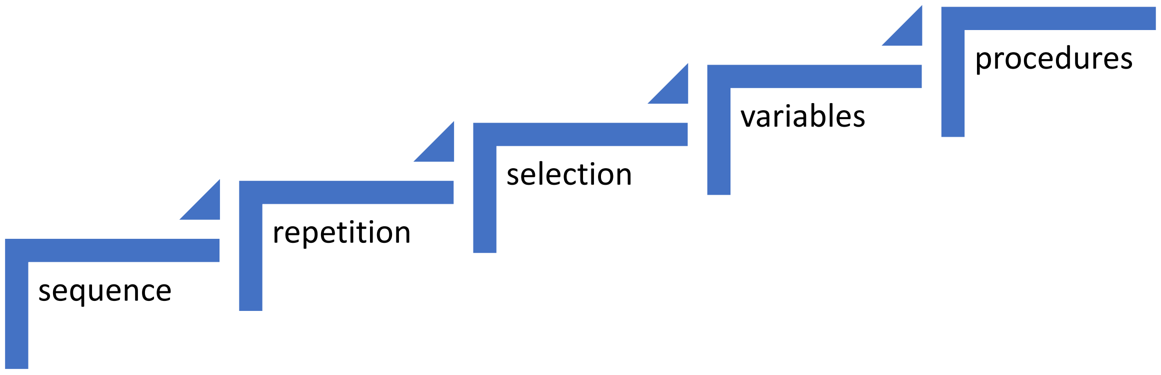
A review of Cognitive Load Theory: Lessons for Teaching Computing

Cognitive load theory has been around for over twenty years. In this article I evaluate what it is and how it can inform good teaching and learning in computing.

Totally new information is first processed by the brain's limited working memory, which is typically able to hold 2-6 elements at a time (depending on the complexity of the data and surrounding distractions). This information is then transferred into the brain's unlimited long-term memory. If the brain's short-term memory is overloaded with new information, it hampers, reduces or stops the transfer of information into long term memory which is necessary to retain learning.

Managing intrinsic load

Intrinsic relates to the complexity of NEW information that needs to be processed. In computing terms, most people would agree that the basic concept of sequence is less intrinsically complex than the basic concept of repetition although we might differ over higher order concepts.



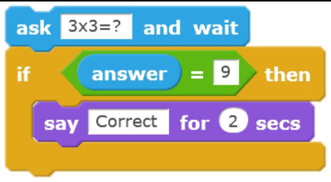
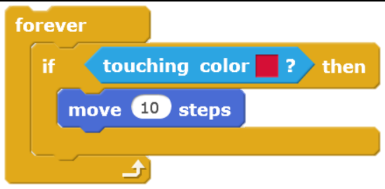
Complexity scale 1

Once we have decided on a scale of least to most complex in all areas of computing, we can use that to develop a progressive scheme of work that teaches the concepts, encourages the skills and attitudes and enables pupils to develop independent agency.

The problems come when educators introduce too many NEW concepts at the same time. To create even the most elementary programs you often need many concepts and it can be tempting to use ones that pupils haven't learnt about yet. We like what it does, and we want to share it with our pupils, but we are creating more cognitive load that will lead to less long-term assimilation of ideas. An expert educator might get around some of these issues in programming by abstracting some of the complexity away from a project through build you own block procedures in Scratch, but we would need to be careful as even the presence of more information could increase cognitive load (see redundancy effect below).

We can also be tempted to introduce a concept using complex examples. The golden rule is: if the idea is totally NEW, strip away all complexity and present it in its most simple form with the least information possible and increase complexity gradually.

Compare these three examples of conditional selection

A	B	C
If you are hungry clap once	 ask 3x3=? and wait if answer = 9 then say Correct for 2 secs	

Example A presents a simple condition that starts an action that will only be checked once. It is written in everyday readable language with the only nod to computing being the layout and the indentation. It builds on pupils understanding of language which they study on a daily basis.

Example B also uses the same condition that starts an action that will only be checked once. It uses the popular quiz format. There is extra complexity in understanding how the answer variable works with the input ask block and in making sense of the blocks. Although I am a great advocate of block-based programming, pupils will have more experience of working with text than blocks.

Example C also uses the same condition that starts an action but to really represent a typical use we have to present it within a continuous loop (as the condition will need to be checked repeatedly) which adds more complexity.

All of these examples are useful and will build understanding, but there is less cognitive load in example A, so I would recommend it as the simplest starting point. It also has the added bonus of developing algorithmic language that pupils can use during the planning/algorithmic level that you can read about here.

Reducing extraneous load

Extraneous cognitive load is affected by how the information is presented and what the learner has to do as part of the learning process. How can the computing teacher design resources and learning experiences that keep cognitive load to a minimum whilst introducing NEW ideas?

Goal Free principle

The idea here is that by holding the method and the goal in short term memory at the same time, we are increasing cognitive load. Replacing a conventional goal with a wider non-specific goal reduces the number of specific things the user has to hold in their short-term memory. Instead of asking if pupils can use conditional selection to steer a character right in Scratch, or turn a buzzer on using the Crumble, we might ask how many methods they can find to use condition-starts-action in Scratch or how many ways can we use condition-starts-action with a button using the Crumble. This goal-free principle is also closer to the spirit of constructionism that we value in computing.

Worked Examples

Worked examples allow the learner to see how a specific type of problem can be solved. The learner doesn't have to hold the information in their working memory because it is presented to them and they can see the steps to solving the problem.

If we are using Lee et al's[1] '*Use, modify, create'* method you are probably already advocates of worked examples. Pupils start with a worked code example for the learner to execute the code and observe how it works. Some pupils may also need to see how it is constructed before they can make meaningful sense out of a preconstructed example. If we subscribe to the levels of abstraction, we may also want to model the idea level, planning level which includes algorithm, initialisation and objects as well as code and execution levels.

Completion Problems

This is an adaptation of the worked example where the learner has a partially worked example that they have to finish. This forces them to think hard about the information being presented. If ultimately our aim is that pupils will be able to write their own programs of any given type, then the worked example might be seen as the first step followed by the partially worked example of a completion problem before writing their own programming. Computing has considerable pedigree in this area as it was Merriënboer and Krammer[2] who first suggested using completion problems for novice computer programmers.

Split attention effect

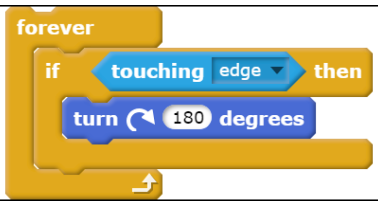
This is where two sources of information are presented alongside each other. Neither of which are effective on their own but in which both are needed for understanding. The learner has an increased cognitive load from trying to integrate both sources of information into a whole. Educators can remove the split attention effect by integrating both sources into a single whole.

Maybe we give the learner a basic diagram illustrating the pathway that data packets take from our home computing device to a waiting web server to retrieve a web page. Alongside that we give them a description of each item in the chain and what it does. Neither example gives the full explanation on its own. If this is NEW information, then significant cognitive load will be expended trying to integrate the sources.

Redundancy Effect

The redundancy effect occurs when learners are presented with two or more sources of information at the same time that could stand on their own. Cognitive load can be decreased by presenting just one source of information at a time.

It might be tempting to introduce an everyday algorithmic example of conditional selection alongside a programming example but if this is NEW knowledge then it would be far better to present them sequentially rather than at the same time.

Loop always If partner taps your shoulder Jump once	 forever if touching edge then turn ↻ 180 degrees

Modality Effect

This came from the idea that working memory can be subdivided into an auditory working memory and a visual working memory and so combining these types of information avoids the split attention and redundancy effects and avoids increasing the cognitive load on a learner.

Teachers should be heartened by the modality effect as it suggests that well chosen teacher commentary can enhance a textual or pictorial visual example without increasing cognitive load.

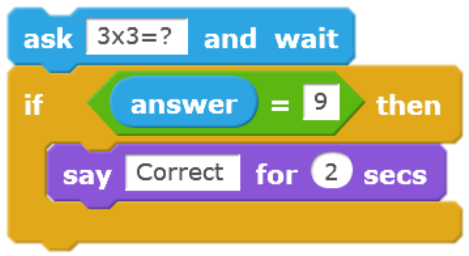
Optimising Germane load

Germane load is best described as the learning that actually occurs. The knowledge that started in working memory and is processed into long term memory and is available for long term recall or to build new schemas.

Variability Effect

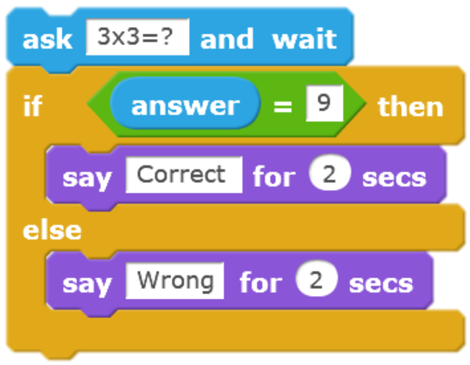
Variability of example bucks the trend in that it increases cognitive load AND increases transfer of learning into longer term memory. However, variability is best expressed through worked examples or completion problems as variability in open ended problems is likely to result in cognitive load for the novice that hinders learning transfer.

For example, the programmer new to conditional selection will benefit from multiple examples of how the same form of conditional selection can be used in many different ways. Once multiple forms of conditions have been introduced, pupils will benefit from seeing and experimenting with one form of program, such as the quiz and how it can be modified using different forms of selection such as those illustrated below.



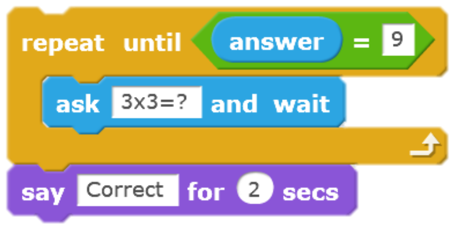
```
ask 3x3=? and wait
if answer = 9 then
    say Correct for 2 secs
```

Condition starts action worked example in a quiz



```
ask 3x3=? and wait
if answer = 9 then
    say Correct for 2 secs
else
    say Wrong for 2 secs
```

Conditions switches between actions worked example



```
repeat until answer = 9
    ask 3x3=? and wait
say Correct for 2 secs
```

Condition stops a continuous loop worked example

Variability suggests that a novice to conditional selection within a loop would benefit from an activity like completing multiple incomplete conditional statements after a partial framework and suggestions have been provided, before moving on to writing their own everyday examples, before moving onto code.

For example

Complete the following algorithms to move and steer characters in a game

Loop always
    If____________
        Move back
	Loop always
	    If key a is used
	        _____________
		_____________
		    If touching red colour
		        Say red love it!

Conditions you can use

If any key is pressed

If touching another character

If touching a colour

Actions you can use

Move forwards

Move backwards

Turn right

Turn left

Stop

Limits of cognitive load theory

Whilst these strategies are good for processing NEW information, they can get in the way of learners who have already assimilated the knowledge. So, when designing schemes of work in computing we need to reduce scaffolding and strategies such as worked examples or completion problems until learners are independently working on every stage of the levels of abstraction.[3]

Phil Bagge

1st Feb 2019

The ideas in this article are taken from these two papers

Cognitive Architecture and Instructional Design: 20 Years Later

John Sweller & Jeroen J. G. van Merriënboer & Fred Paas 2019

Cognitive load theory in health professional education design principles and strategies

Jeroen J G van Merrienboer & John Sweller 2009

References

[1] Irene Lee et al Computational thinking for Youth in practice (2011)

Use and modify involves exploring premade code before modifying it and finally building your own program. A process that takes the user from non-ownership to ownership.

[2] van Merriënboer, J. J. G., & Krammer, H. P. M. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. Instructional Science, 16,251–285

[3] Comparing K-5 teachers' reported use of design in teaching programming and planning in teaching writing, Waite (2018)