Exploring and understanding pupils’ lack of perseverance and autonomy with debugging in computing

Gurmit Uppal
University of East London, UK

ABSTRACT

The National Curriculum for Computing in England expects that primary-school-aged pupils (5- to 11-year-olds) will be able to correct programming errors in age-appropriate contexts (DfE) 2013). This correction of errors in computing is known as debugging. Utilising a broadly autoethnographic approach, this paper draws upon the writer’s positionality as a computing teacher in primary school and as a teacher educator in a university-based setting. Reflecting upon experiences of teaching computing (specifically debugging) to primary school pupils, the paper goes on to outline and explore potential reasons for pupils’ lack of perseverance and autonomy when engaged with debugging activities. Ideas around learnt helplessness and cognitive load theory are analysed as potential barriers to pupils’ progress when it comes to debugging. The reflective process concludes with suggestions to further develop pupils’ independence with debugging activities, as well as considering the importance of teacher educators’ own practice-based experiences to share with new teachers.

KEYWORDS

COMPUTING

DEBUGGING

AUTOETHNOGRAPHY

LEARNT HELPLESSNESS

COGNITIVE LOAD THEORY

TEACHER EDUCATION.

INTRODUCTION

As a teacher educator who continues to cross the boundary between roles in higher education (HE) and computing teaching in the primary (ages 5 to 11) classroom, I consider myself to have benefited from the polycontextuality (Kidd, 2012, cited by Czerniawski, 2018) which these two roles bring. When teaching and tutoring trainee teachers I am able to draw upon recent and relevant experiences which provide context to theory. Similarly, when teaching computing in the classroom setting there is a depth to my practice which has been enhanced through my years of working in university- based teacher training.

Computing is my area of expertise as a teacher educator on the Primary Postgraduate Certificate in Education (PGCE) course, which is a one-year course for trainee teachers in England. As a teacher educator, I am acutely aware that the subject knowledge, theory and pedagogy which I advocate in the lecture theatre may, at times, be far from easy to follow through into classroom practice, even for experienced teachers like myself. With this in mind, I have taken a broadly autoethnographic approach to reflect on my own practice and beliefs in relation to a specific area of computing teaching. Specifically, I wanted to better understand whether I was promoting pupils' autonomy in the computer science area of debugging.

AUTOETHNOGRAPHY

Autoethnography is a qualitative research method which draws on one's personal experience to describe, reflect upon and critique practices and beliefs (Adams et al. 2014). Unlike ethnography, it is pragmatic in acknowledging and owning the subjectivity in the relationship between researcher and the researched (Ellingson & Ellis 2008).

Autoethnography is rooted in an active and probing form of reflection known as reflexivity, where research analysis is focused on how the researcher's 'thoughts, feelings, values, identity [impact] … upon others, situations, and professional and social structures' (Bolton 2010). In my role as teacher, it provided a relevant and in-the-moment approach which allowed me to consider how my thought processes, beliefs and identity were influencing my pedagogical decision making in the classroom.

Autoethnographic research does not attempt to offer findings that can be widely generalised; instead the focus is specific, and studies should be judged on the story and its impact on improving the lives of participants, readers or indeed the researcher's own (Ellis 2004). For me, adopting this approach was as much about taking the time and space to reflect on my practice as it was about improving my practice for pupils and future teachers alike.

SUBJECT CONTEXT:

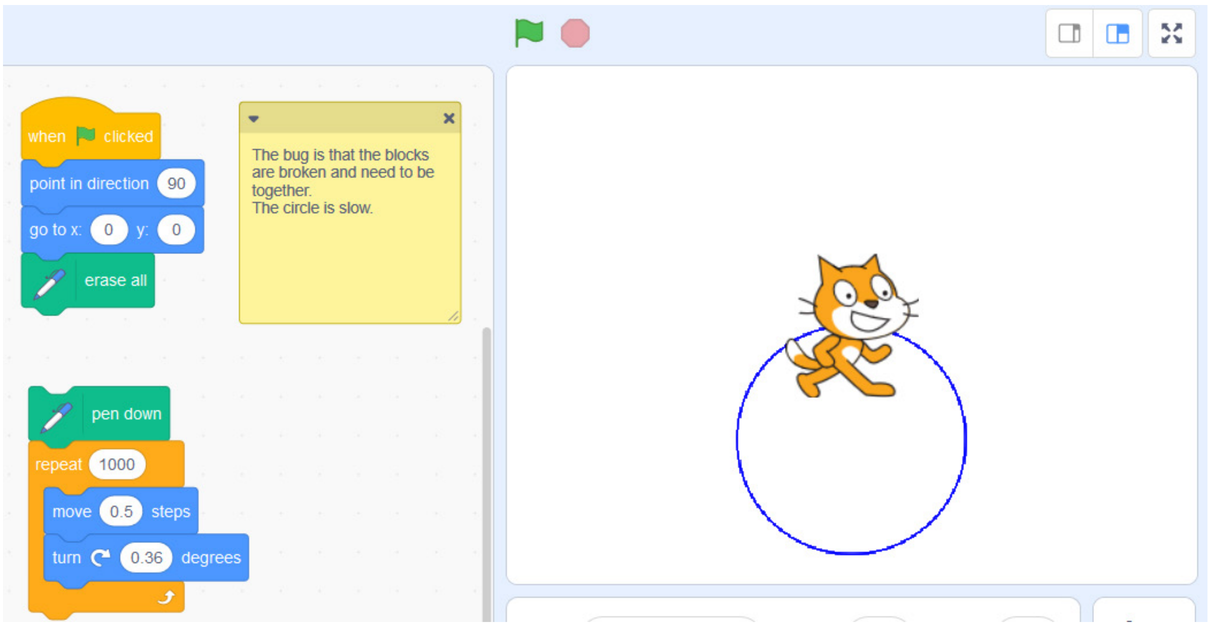
COMPUTING, COMPUTATIONAL THINKING AND DEBUGGING

The research evidence which is the basis of this paper relates to an area of computer science known as debugging. Debugging must first behhhhh introduced and explained within the wider context of the National Computing Curriculum (DfE 2013) which is currently in place in England, and computational thinking which is at its very core. Computational thinking can be defined as the process deployed to solve problems and outline solutions by drawing on the fundamental skills of computer science (Papert 1980; Wing, 2006, 2011). The development of computational thinking skills is central to the successful implementation and impact of computing in schools. The knowledge and skills developed through this curriculum will also equip learners with transferable tools which they can apply across other areas of learning (Morris et al. 2017). Despite its name, computational thinking should be most valued for its development of pupils' skill set in tackling problems in increasingly efficient ways, regardless of whether technology is involved or not.

Computational thinking concepts which are regularly referred to in computing education include: logical reasoning, decomposition, abstraction, pattern recognition, algorithms and evaluation. In addition to these concepts, approaches to the tackling of problems include: tinkering, creating, debugging, persevering and collaborating (Berry 2015). In this paper, I will focus specifically on perseverance and debugging. Debugging is the identification and correction of errors in computer science. National Curriculum requirements in England state that by the end of Key Stage 1, 'pupils should be taught to create and debug simple programs', and by the end of Key Stage 2, 'design, write and debug programs that accomplish specific goals' (DfE 2013).

RESEARCH CONTEXT

In addition to my role as a teacher educator, I am also a computing subject leader and teacher in a primary school, where I teach pupils between the ages of seven and eleven (known as Key Stage 2 in English primary schools). As a subject leader, I acknowledged that assessment data captured from the previous school year demonstrated that pupils' debugging skills were an area of weakness for many

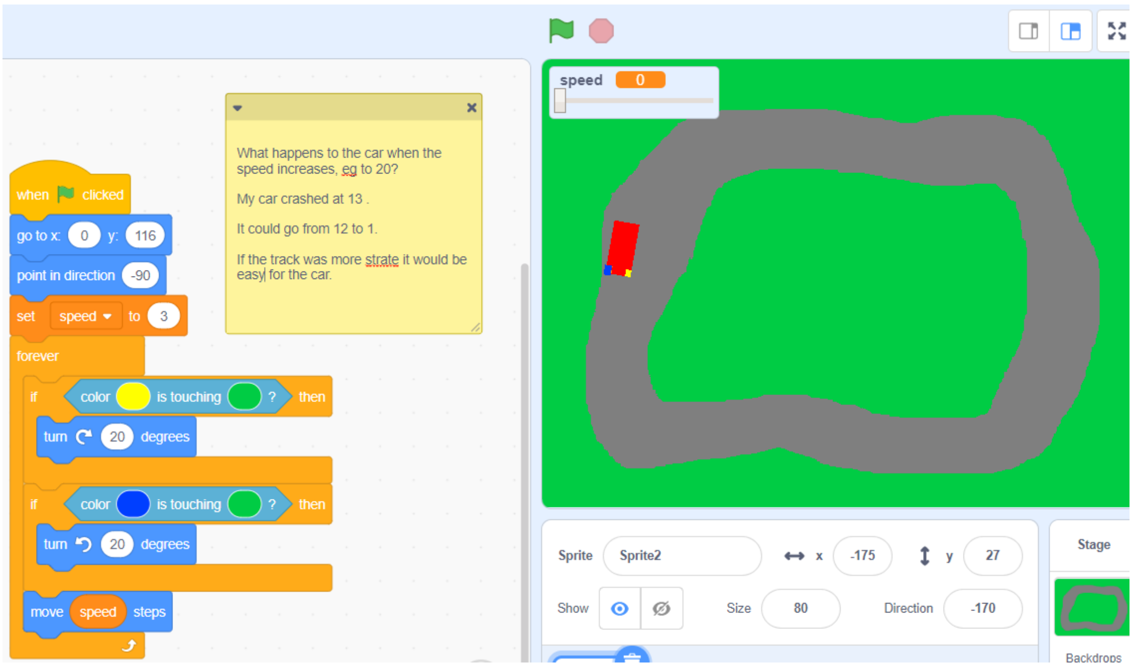


```
when [flag] clicked
point in direction (90)
go to x: (0) y: (0)
erase all

pen down
repeat (1000)
    move (0.5) steps
    turn ↻ (0.36) degrees
```

The bug is that the blocks are broken and need to be together.
The circle is slow.

Fig. 1. This example shows a performance bug, as well as a break in the program. Pupils' annotations show that they were able to recognise the blocks needed to be connected and they also recognised that the drawing of the circle on screen was slow. They were unable to improve the performance of the program without significant scaffolding and guidance.



```
when 🏁 clicked
go to x: 0 y: 116
point in direction -90
set speed to 3
forever
    if color 🟨 is touching 🟩 ? then
        turn ↻ 20 degrees
    if color 🟦 is touching 🟩 ? then
        turn ↺ 20 degrees
    move speed steps
```

What happens to the car when the speed increases, eg to 20?

My car crashed at 13 .

It could go from 12 to 1.

If the track was more strate it would be easy for the car.

Fig. 2. This example shows an exploratory task where pupils were required to investigate and explain what happened to the car on the track when the variable (speed) was increased. After initial guidance, pupils were able to identify that the car stayed on track with an optimum speed of 12 but came off the track when speed was increased to 13. Through prompted questioning, pupils were able to identify how the track shape may need to be adapted to obtain faster speeds.

pupils across all year groups in Key Stage 2. As a result, I decided to make this a specific focus for my research in an attempt to further understand and explore potential reasons and solutions for this area where pupil progress was lacking.

A six-week unit of work with a specific focus on debugging was carried out with a group of Year 3 (seven- to eight-year-old) pupils. The planning for the unit was based around a unit of work called, 'We Are Bug Fixers' from a published scheme of work by Rising Stars (Berry 2014). Learning outcomes for the unit were around pupils developing their strategies for finding errors in programs and their development of resilience and strategies for problem solving.

A typical lesson consisted of introducing pupils to a type of computer bug, then modelling the issue through an example created in Scratch programming software (MIT 2019). This was then followed by facilitating a shared discussion around why the program was not working or how it could be improved. Pupils then moved on to explore the program for themselves (in mixed attainment pairs) to identify and explain the error through on-screen annotations, as can be seen from the two examples shown (Figs. 1 and 2).

REFLECTION AND EMERGING THEMES

The weekly lessons delivered over a six-week period were enjoyed by the pupils as each of the programs they explored was contextualised within a theme or scenario which was of interest, for example, drawing, a racing car track, correcting muddled joke dialogue, etc. In addition, the majority of pupils were able to identify and explain the bug in each type of program and make general suggestions for what needed to be done to improve the program.

It was the bridge between identifying the bug in the program and going on to correct it which is the area I believe required further consideration. Early on, it was evident that many of the pupils were unable or reluctant to try to correct the programming for themselves. This was manifested in many ways, from pupils expecting help with every step, stating the task was too hard, lacking perseverance and at times demonstrating off-task behaviour. It must be acknowledged at this point that there were indeed times when the mathematical knowledge required to understand the program was beyond the maths-based curriculum which had to be covered in Year 3. This is something which I have encountered before with Scratch programming, and I have always ensured that the necessary pre-teaching was in place to provide the required background understanding. For example, the lesson depicted in Figure 1 had begun with pupils pretending to be human robots, to enable understanding of direction and angle.

Before pupils explored the programs for themselves, they had been exposed to vocabulary explanations, step-by-step modelling and prompts to promote ways in which they too could tackle the task. Despite the steps taken to make the learning content accessible for pupils, I observed a neediness from them which impacted on my pedagogical approach. I have identified the themes of learnt helplessness and cognitive load theory to analyse and discuss further with the aim of providing possible reasoning and rationale for the pupils' lack of autonomy when debugging, as well as understanding and learning from my own practice in these lessons.

LEARNT HELPLESSNESS

Learnt helplessness is a phenomenon which has been observed in classrooms where some pupils believe that their success in achieving goals is out of their control (Seligman 2018). This is not something unique to computing and is often seen in other STEM (science, technology, engineering and mathematics) subjects where pupils' confidence may be low, and success or failure are more obvious (Yates 2009). Phil Bagge has further explored this issue in relation to the teaching of computing and the development of pupil autonomy in problem solving. Bagge (2015) defines learnt helplessness as times when pupils try to get 'other people to solve problems for you, these others may be the teacher, classroom assistant or other pupils'. Learnt helplessness from pupils has many forms: pupils may be demanding of adult help, they may lack perseverance with tasks and they may become upset or display signs of low-level disruption. If enough pupils are displaying some of the traits of learnt helplessness, a teacher may feel that it is their fault if pupils are not achieving and the lesson is not successful.

As a teacher educator, I have regularly stressed the importance of teachers not debugging programming problems for their pupils and instead promoting strategies which allow pupils to take ownership of this area. However, in the midst of a busy classroom where as a teacher I was multitasking on many levels, it was all too easy to over-scaffold and provide too much direction, to give the appearance of a seemingly successful lesson. As a result, the decisions which I took in the moment were not those which I would advocate to others, having reflected on the situation post-event (Schön 2016).

Despite the new Computing Curriculum being introduced in 2013, teacher confidence in computing still varies considerably (The Royal Society 2017) and it is acknowledged that the ambition and demands of the Computing Curriculum should be prioritised as a key area for the professional development of teachers (Myatt 2018). Confidence could be an important factor in relation to why some teachers may not be promoting debugging skills amongst pupils. Encouraging pupils to identify, explain and correct errors with increasing autonomy would mean that teachers would also need to understand the errors in the algorithms themselves, but it would also run the risk of pupils appearing to be stuck and not being successful. For me, as an experienced computing teacher, I believe my reasons for not fully handing over the debugging process to pupils was more about a need for control and wanting pupils to feel that they had been successful in the lessons. With this in mind, I will now move on to explore one possible theory as to why the debugging process may have posed difficulties for pupils.

COGNITIVE LOAD THEORY

The Office for Standards in Education (Ofsted, 2019a) recently published an overview of the research evidence which supports its new inspection framework. With an increased focus on the intent, implementation and impact of a well-sequenced, knowledge-rich curriculum, this has once more brought cognitive load theory to the fore (Ofsted, 2019b). Cognitive load theory is based around changes in short-term working memory and the subsequent impact on the infinite longer-term memory (Sweller, 1998). As short-term working memory can only process a limited amount of information at any given time, this working memory can become overloaded and result in errors: the inability to follow or remember instructions; place-keeping errors; incomplete recall or task abandonment (Gathercole & Alloway, 2007, p.15).

Cognitive load theory identifies three types of cognitive load:

- Intrinsic cognitive load: the inherent difficulty of the material itself, which can be influenced by prior knowledge of the topic
- Extraneous cognitive load: the load generated by the way the material is presented and which does not aid learning
- Germane cognitive load: the elements that aid information processing and contribute to the development of 'schemas'. (Shibli & West, 2018)

The requirements for debugging in the lessons I delivered presented a high intrinsic load for pupils. The mathematical subject knowledge was new to them, and the programs they were using were different to their previous learning experiences, therefore pupils may have been unable to draw upon previously stored knowledge. Although well intentioned, I may have also inadvertently increased the extraneous load for pupils through modelling that consisted of too many steps, and visual slides that contained multiple prompts and diagrammatic steps to also represent processes. As a result, the high intrinsic and extraneous loads may have had a negative impact on the germane load.

Cognitive learning theory asserts that pedagogical approaches and lesson materials can be adapted to reduce the extraneous load. This allows learners to focus on the germane (or relevant) processes required for constructing schemas through the recognition of patterns, organising information and linking previous learning and new. In relation to computing, Bagge (2019) believes that cognitive load theory can help develop children's agency, through strategies such as avoiding introducing too many concepts at the same time. In many ways the concepts and approaches that make up computational thinking already promote an ideal instructional design framework for cognitive load theory, as problems are tackled through efficient approaches. Carefully thought-out medium- and long-term planning sequences in computing could also improve the retention, recall and application opportunities pupils require to facilitate the changes in long-term memory and schemata. This could help to reduce the load on working memory the next time new learning in a related area is encountered.

The significance of cognitive load theory on learning has been questioned. De Jong (2010) suggests that cognitive load theory is difficult to disprove, as the three types of load will always present in learning scenarios, and success or failure can always be attributed to one or the other. Indeed, he suggests that what may be identified as extraneous in one case may be germane in another. Whilst the principles of cognitive load theory appear sound, other classroom and individual factors must also be considered when identifying impact on cognitive load. For this reason, more recent waves of research on cognitive learning theory have moved forward to acknowledge and analyse factors such as instructional design, and environment-related factors such as emotions, stress and uncertainty (Sweller et al. 2019). This has shown that such factors can impinge on cognitive load and consequently negatively impact working memory.

CONCLUSION

My intention through this research paper was to reflect on my own practice to explore whether I was promoting pupils' autonomy in the area of debugging. As a teacher educator this process has made me recognise that there are times when I may not apply the approaches which I advocate to others, instead allowing the demands of a progress-orientated classroom setting to lead the choices I made to provide short-term success over longer-term learning.

Through reflection and analysis of my own practice, I acknowledge that I need to hand over more of the debugging process to pupils. Bagge (2015) acknowledges that this is a process which takes time for teachers, before they can then promote greater autonomy amongst pupils. My natural instinct as a teacher was to intervene to support pupils when they appeared stuck; however, there were times when this led to an overuse of scaffolding and prompts for pupils, which detracted from the development of pupils' own debugging skills. It is important that pupils recognise debugging as part of their learning in computer science, and specific strategies need to be taught to pupils to foster independence. Whilst as a teacher I may need to step back, collaboration should be encouraged in computer science, with studies showing peer work to have a positive impact on risk taking and perseverance (Baroutsis et al. 2019). Paired and group work is an approach which I encourage in my computing lessons, although my choice of pairings has often been based on attainment rather than potential impact on perseverance with problem solving, which is certainly worthy of consideration in future lessons.

Subject knowledge development is paramount if learners are to use and apply prior knowledge from long-term memory to new tasks. In the lessons which I delivered, the pupils had not encountered the maths and some of the programming elements which they were now being asked to examine and correct. As a result, they were unable to make links to previously stored learning and their working memories were overloaded with too much new information to process. Not having previously taught this year group, I was unable to ascertain the gaps in their learning until after the unit was underway, and consequently the unit of work needed significant adaptation and pre-teaching on a weekly basis. The sequence in which curriculum topics are taught is an area of increased focus identified in the recent Ofsted framework (Ofsted 2019b). The teaching of a unit once during a Key Stage cannot be assumed to be learning which is embedded or transferable. As a subject leader, I have found that my experience and reflection on teaching this unit will help me to move forward with long-term planning to ensure that pupils have opportunities to revisit concepts regularly to develop their retention and application of knowledge. Nevertheless, it must not be assumed that all pupils will experience cognitive overload. Kalyuga (2007) suggests that the instructional strategies which work for novice learners could have a negative impact on expert learners who are already able to make connections between new and prior learning. Therefore, it is imperative that teachers are able to design learning sequences that incorporate pedagogical approaches which suit the subject knowledge, as well as the learning dispositions and prior knowledge of their pupils.

As a teacher educator it is all too easy to become detached and focus on how education should be, as opposed to dealing with the reality of the pressures and stresses which impact on teachers' daily decision making. This journey of reflection has provided a timely reminder of the challenges which new teachers will face as they try to implement the approaches which they have learned about, and how as teacher educators we must continue to have one foot in the classroom to provide an authentic evidence-based training experience for our new teachers. ■

REFERENCES

Adams, T. E., Jones, S. H. & Eliis, C. (2014). *Autoethnography: understanding qualitative research*. Oxford: Oxford University Press.

Bagge, P. (2015). 'Eight steps to promote problem solving and resilience and combat learnt helplessness in computing'. Retrieved from *ICT in Practice*: http://www.ictinpractice.com/eight-steps-to-promote-problem-solving-and-resilience-and-combat-learnt-helplessness-in-computing-by-phil-bagge/

Bagge, P. (2019). 'Cognitive load theory in the computing classroom'. *Hello World*, May, 34–5.

Baroutsis, A., White, S., Ferdinands, E., Goldsmith, W. & Lambert , E. (2019). 'Computational thinking as a foundation for coding: developing student engagement and learning'. *Australian Primary Mathematics Classroom*, 24 (2) pp. 10–15.

Berry, M. (2014). *Switched on computing – Year 3*. London: Rising Stars UK Ltd.

Berry , M. (2015). *Computing at School (CAS) quick start computing: a CPD toolkit for primary teachers*. London: Department for Education.

Bolton, G. (2010). *Reflective practice: writing and professional development.* London: SAGE.

Czerniawski, G. (2018). *Teacher educators in the twenty-first century: identity, knowledge and research*. St Albans: Critical Publishing.

De Jong, T. (2010). 'Cognitive load theory: educational research, and instructional design: some food for thought'. *Instructional Science*, 38(2), 105–34.

Department for Education (DfE). (2013). 'England: computing programmes of study'. Retrieved from *GOV.UK:* https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Ellingson, L. L. & Ellis, C. (2008). 'Autoethnography as a constructionist project'. In J. A. Holstein & J. F. Gubium, *Handbook of constructionist research*, pp. 445–6. London: The Guilford Press.

Ellis, C. (2004). T*he ethnographic I: a methodological novel about autoethnography.* Walnut Creek, CA: Altamira Press.

Gathercole, S. E. & Alloway, T. (2007). *Understanding working memory: a classroom guide*. London: Harcourt Assessment.

Kalyuga, S. (2007). 'Expertise reversal effect and its implications for learner-tailored instruction'. *Educational Psychology Review*, 19, 509–39.

Massachusetts Institute of Technology (MIT) (2019). 'About Scratch'. Retrieved from *Scratch*: https://scratch.mit.edu/

Morris, D., Uppal, G. & Wells , D. (2017). *Teaching computational thinking and coding in primary schools*. London: SAGE Learning Matters.

Myatt, M. (2018). *The curriculum: gallimaufry to coherence*. Woodbridge: John Catt Educational Ltd.

Office for Standards in Education (Ofsted) (2019a). 'Education inspection framework: overview of research'. Retrieved from *GOV.UK*: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/813228/Research_for_EIF_framework_100619__16_.pdf

Ofsted (2019b). 'Education inspection framework (EIF)'. Retrieved from *GOV.UK*: https://www.gov.uk/government/publications/education-inspection-framework

Papert, S. (1980). *Mindstorms: children, computers and powerful ideas.* New York: Basic Books.

The Royal Society. (2017). *After the reboot: the state of computing education in UK schools and colleges*. London: The Royal Society.

Schön, D. (2016). *The reflective practitioner.* Abingdon, Oxon: Routledge.

Seligman, M. (2018). *The optimistic child.* London: Nicholas Brealey Publishing.

Shibli, D., & West, R. (2018). 'Cognitive Load Theory and its application in the classroom'. Retrieved from *Impact - Journal of Chartered College of Teaching:* https://impact.chartered.college/article/shibli-cognitive-load-theory-classroom/

Sweller, J. (1998). 'Cognitive load during problem solving: effects on learning'. *Cognitive Science*, 12(2), 257–85.

Sweller, J., van Merriënboer, J. J. & Paas, F. (2019). 'Cognitive architecture and instructional design: 20 years later'. *Educational Psychology Review*, 31(2), 261–92.

Wing, J. M. (2006). 'Computational thinking'. *Communications of the Association for Computing Machinery (ACM)*, 49(3), 33–5.

Wing, J. M. (2011). 'Research notebook: computational thinking – what and why?' *The Link –Magazine of the Carnegie Mellon University School of Computer Science*, 8, 20–3.

Yates, S. (2009). 'Teacher identification of student learned helplessness in mathematics'. *Mathematics Education Research Journal*, 21(3), 86–106.