Teaching Computational Thinking and Fostering Literacy Through Computer Programming in Primary Education

Joëlle Hanna Hibbel

2023

Table of Contents

Summary

This research project investigated how facilitating the computational thinking process of 10-12 year old students through structured decomposition instruction influenced their programming ability. In addition, it tested if programming lessons in Scratch, with or without the inclusion of decomposition instruction, resulted in more complete problem decomposition in writing. Data was collected from 137 students at two Dutch primary schools. No significant differences in programming ability were found between students who received decomposition instruction and those who did not, possibly due to the transfer paradox described in previous research. Furthermore, programming lessons were not found to impact students' mean scores on problem decomposition in writing, but a considerable larger spread in results was observed for students that did not participate in the programming lessons. Based on these findings, the study suggests that the programming lessons might have had a positive impact, especially on lower performing students. Results of the study partially aligned with the Computational Thinking Process Model, which was proposed by the research to accommodate to the scholarly expressed need of a concise and versatile definition of computational thinking for educational science and practice.

Introduction

Curriculum Development in Technology Education

Educational curricula are an ongoing topic of debate in the Netherlands, both in the public domain and in the political arena. The focus of this debate adheres to altering public opinions and associated political views as time passes (Agirdag et al., 2021; Nieveen & Kuiper, 2012; Visser, 2016). Over the last decade, a persistent demand for investments in science and technology education was put forward in partnership by national and local governments, educators and employers (Techniekpact, 2013). Moreover, the governmental advisory board on curriculum revision strongly emphasized the importance to include digital literacy and computational thinking (CT) as learning objectives in primary and secondary school curricula (Platform Onderwijs 2032, 2016). These recommendations were met with scepticism by leading teacher organizations, who were unsatisfied with their marginal role in the formulation of the new curriculum objectives and, among other things, compelled for a more moderate expansion of technology in education (Visser, 2016).

Meanwhile, in the aftermath of the COVID-19 pandemic, the Dutch Educational Department has been emphasising the need to strengthen core competences in reading, writing and arithmetic (Swanborn, 2023). However, ongoing digitalisation and an undiminished demand for highly skilled technical employees urge for expanded efforts in science and technology education concurrently, and debates on appropriate curriculum innovations continue worldwide. As a result, some European countries have imposed computer programming in primary education, including Great-Britain, Finland (Balanskat & Engelhardt, 2014), Denmark, Sweden (Bråting & Kilhamn, 2022) and Italy (Chiazzese et al., 2019). Several non-European countries like Russia, Australia and the USA have included computer science (CS) in the K-12 curriculum as well (Falloon, 2016; Grover & Pea, 2013; Tedre & Denning, 2016). The global Organization for Economic Co-operation and Development (OECD) highly values this development, praising digital technologies as "key resource for OECD school systems, holding immense promise for education and skill development" (Nusche & Minea-Pic, 2020, p.3). However, these praises come with serious warnings, as OECD data shows that merely increasing use of digital technology does not equivalently increase learning outcomes. On the contrary, detrimental effects on test scores for reading, science, mathematics, cooperative problem solving and well-being are reported in relation to both very low and very high levels of ICT use. Therefore, a thoroughly substantiated pedagogy is needed to magnify learning through technology education, and curriculum innovations must be guided by ample research to achieve cognitive benefits for learners that exceed mastering a particular skill within the limited scope of the training situation. (Falloon, 2016; Kumar et al., 2023; Nusche & Minea-Pic, 2020; Pea & Kurland, 1984). Regarding the implementation of computer programming in the K-12

curriculum, questions remain on age appropriate learning objectives and on how to design effective learning activities to promote those objectives (Lee et al., 2023; Popat & Starkey, 2019).

Research Topic

This research project investigated how implementing computer programming lessons in primary education could foster CT proficiency and consequently enlarge problem solving skills in other domains. One group of students (i.e.: experimental condition A) received programming lessons augmented with targeted instruction in problem decomposition, an essential sub-skill of CT. Another group of students (i.e.: experimental condition B) received programming lessons without explicit attention to problem decomposition. The programming results of both groups were compared to examine how augmenting programming lessons with instruction in problem decomposition influenced programming ability. Furthermore, the research explored how learning in the CS domain could transpire to other domains. More specifically, it was tested if such transfer occurred between the domains of CS and literacy. To do so, the results on drafting a problem decomposition for writing a persuasive letter were compared between students who had received programming lessons in experimental condition A or B, and students of a baseline group, who had not received any programming lessons at all.

Theoretical Framework

Programming as a Vehicle for Computational Thinking

Computer programming has become accessible to primary school children in the 1980s through LOGO (Papert, 1980), a programming language that enabled children to draw geometrical shapes by programming a turtle's pathway across the computer screen. Papert expected great learning gains from computer programming, exceeding the initial benefits of acquiring mere programming skills to enhanced conceptual understanding and general problem solving ability. While Papert referred to the idea of expansive computational ability as *Computational Thinking* (CT), the author did not define it as a detailed concept (Papert, 1980, p.182). Contrary to Papert's expectations, extensive research conducted between 1980 and 1990 revealed that most learners struggled to acquire competence in programming itself and that knowledge transfer from programming to other disciplinary areas was rare (Littlefield et al., 1988; Mayer et al., 1986; Salomon & Perkins, 1989; Tedre & Denning, 2016).

Pleas to include CT in school curricula have been brought forward with renewed enthusiasm since Jeannette Wing in 2006 advanced the concept of CT as an indispensable skill to function in contemporary society (Grover & Pea, 2013; Lodi & Martini, 2021; Tedre & Denning, 2016; Wing, 2006). Wing asserted CT as a "universally applicable attitude and skill set" (2006, p. 33), a vision broadly acknowledged by other scholars (e.g., Brennan & Resnick, 2012; Voogt et al., 2015). It involves thinking about all kinds of systems and processes and applying strategies used in CS to get a better understanding of them (Grover & Pea, 2013). The increased availability of digital tools and technological infrastructure (e.g. wi-fi) as well as the development of programming languages and devices tailored to children's needs and interests made programming in primary education a more feasible exercise than ever before, subsidising the plea to include CT in compulsory education from early age (Falloon, 2016; Fagerlund et al.,2021; Lodi & Martini, 2021; Rodríguez-Martínez et al, 2020; Tedre & Denning, 2016).

In order to reach educational objectives of CT, implementing computer programming into the educational curriculum seems a logical choice. Undoubtedly, the concept of CT is rooted in computer science (CS), with programming as its fundamental practice. Therefore, programming should be considered a primary vehicle to develop and demonstrate CT competencies (Bråting & Kilhamm, 2022; Fagerlund et al., 2021; Grover & Pea, 2013; Kwon et al., 2021; Tedre & Denning, 2016; Voogt et al., 2015). Additionally, CT can be applied and trained in so called unplugged situations in which computational problems solving is practiced without using a computer (Bar & Stephenson, 2012; Grover & Pea, 2013; Tang et al., 2020; Voogt et al. 2015). Those unplugged activities could involve tasks with more or less resemblance to computer programming, such as programming an educational robot or playing a board game (Tsarava et al., 2019). However, as the concept is not unambiguously defined,

it remains unclear which skills and attitudes CT actually comprises (Brennen & Resnick, 2012; Grover & Pea, 2013; Lodi & Martini, 2020; Rodríguez-Martínez et al., 2020; Tang et al. 2020; Tedre & Denning, 2016; Voogt et al., 2015).

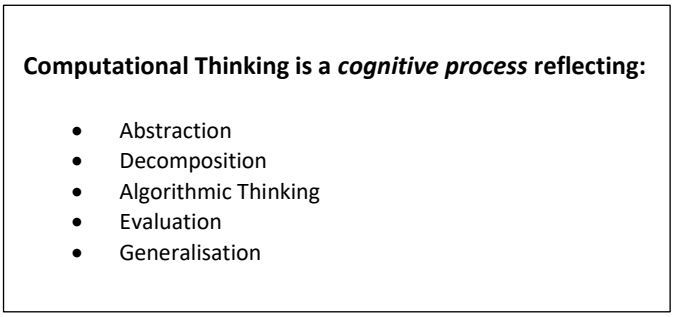
Proposed Definitions of CT

From 2006, CT has been slowly redefined to establish its comprising elements. For example, Wing proposed to define CT as "the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer, human or machine, can effectively carry out" (Wing, 2017, p.8). Wing consistently emphasized the importance of CT for everyone in society as an "intellectual framework for thinking" (2017, p.7) and has upheld CT to comprise numerous strategies derived from CS and other disciplines (2006, 2017).

Other proposed definitions show lots of similarities, however, they diverge on many aspects as well, including the number of containing elements and their level of abstraction. For example, Brennen & Resnick (2012) incorporated technical concepts that can be applied during the programming process, such as loops, conditionals and operators, in addition to computational practices formulated at a process level, such as testing and debugging. Moreover, they included perceptions of programmers about themselves and their work, labelling those views as computational perspectives (e.g., connecting). As a comparison, Grover & Pea (2018) also differentiated between computational concepts and practices, whereas they formulated computational concepts at the level of mental processes (e.g.: pattern recognition), that is, more abstract than the concepts included by Brennan and Resnick (2012). A concise definition of CT as a thought process, comprising five distinct thinking skills was proposed by Selby & Woollard (2014) (see Figure 1). They argued to leave out all elements prone to multiple interpretations (e.g., logical thinking) or elements merely evidencing the use of skills. Debugging, for instance, though considered a fundamental practice in CS, is aided by the application of CT skills, thus not to be identified as an element of the thought process itself.

Figure 1

Computational Thinking Definition by Selby and Woollard (2014)



Contrastingly, after reviewing sixteen often cited CT definitions, Lodi & Martini (2020) proposed unifying those into one comprehensive list of twenty fundamental CT characteristics, divided over four categories, namely mental processes (e.g. logical thinking), methods (e.g. data collection, analysis and representation), practices (e.g. experimenting) and transversal skills (e.g. perseverance).

Many elements proposed by researchers to constitute CT are not exclusively affiliated with CS, such as group problem solving (Bar & Stephenson, 2011), being incremental and iterative (Brennan & Resnick, 2012) investigating a complex system as a whole (Weintrop et al., 2016), creativity (Grover & Pea, 2018), planning and prewriting (Hassenfeld et al., 2020) and designing (Kwon et al., 2021). On the one hand, including elements associated with particularly human abilities in comparison to more mechanical qualities underscores the human nature of CT, emphasised by many scholars (e.g., Voogt et al., 2015). On the other hand, concerns have been raised about too broad definitions of the concept. Including elements with ambiguous or indefinite meaning, or allowing for multiple interpretations, is complicating the discussion among researchers, and the development of instruments aimed at measuring the construct. Those instruments, however, are essential to experimental research and curriculum design (Grover & Pea, 2013; Román-González et al., 2018; Selby & Woollard, 2014; Shute et al., 2017; Tang et al., 2020; Tsarava et al., 2022; Zhang & Nouri, 2019).

Furthermore, including elements that are not restricted to CS might lead to a bleaker understanding of the core qualities of CT inside and outside of the research community (Voogt et al., 2015). Especially people less familiar to CS and its core practices and concepts, including the majority of curriculum designers and educators, might be inclined to focus on those more peripheral elements that they have better understanding of (Bråting & Kilhamn, 2022; Corradini et al., 2017; Nouri et al., 2020; Lodi & Martini, 2020; Tedre & Denning, 2016). Nevertheless, definitions that predominantly entail elements derived from computer programming practices could obscure the divergence between CS and CT as related but distinct concepts (Tang et al., 2020; Voogt et al., 2015). Avoiding described ambiguities, integration of CT in education would benefit more from a compact description of what matters most in CT, than from an exhaustive definition of the concept, according to Voogt et al. (2015).

Misconceptions of CT

Poor understanding of CT is not a strictly theoretical issue in primary education (Bråting & Kilhamn, 2022; Corradini et al., 2017; Kang et al., 2020). For example, Kang et al. (2020) revealed that American primary school teachers had limited understanding about CT and concurrently expressed less confidence in teaching CT related concepts compared to other elements of the science and technology curriculum. Limited understanding of CT was also reported by Corradini et al. (2017) in Italian primary school teachers, who participated in a professional development course on CS and ICT. Upon course completion, only a small minority of the participants was able to provide a satisfactory definition of

CT. Corradini et al. (2017) put forward that a significant proportion of Italian teachers identifies CT as being instrumental to other causes, such as creativity and collaboration, rather than valuable in itself. However, these conclusions should be met with some caution, as the teachers in this study might not have been aware they were expected to provide a thorough definition of CT. Instead, the question that was asked to them, namely to complete the sentence "In my view, CT is…", could have prompted the teachers to offer their opinion on the broader value of CT for education. Moreover, teachers' unsatisfactory definitions might not necessarily reflect misconceptions about CT, but could also origin from insufficient specific vocabulary knowledge to display their understanding of the concept at a scholarly level. However, where findings of Corradini et al. (2017) may remain inconclusive, Bråting and Kilhamn (2022) pointed to similar concerns of limited understanding of CT, in this case in educational developers. After reviewing the compulsory implementation of CT in Swedish primary schools, the researchers concluded that many textbook tasks demanded application of an isolated CT skill, without combining various CT elements and without connecting the application of the skill across various contexts. In other words, the analysed textbooks did not offer students the opportunity to practice CT extensively. In addition, Bråting and Kilhamn (2017) found that multiple tasks labelled as CT tasks in the textbooks under review, used to be part of the traditional Swedish primary school mathematics curriculum, and thus should not be considered an enhancement of curriculum.

In conclusion, the research discussed above illustrates how inadequate understanding of CT may hamper educational practices, and thus corroborates the necessity of a clear and concise definition of CT as requested by various researchers (e.g.: Tsarava et al., 2022; Voogt et al., 2015). However, any definition that enlists comprising elements of CT carries the risk of simplified application in educational practise, such as identified in the Swedish curriculum by Bråting and Kilhamn (2017), with instruction predominantly focussing on isolated sub-skills, rather than the complex coherence between the comprising elements of CT. Therefore, the field of education would benefit from adopting a definition of CT that, besides being clear and consice, emphasises the complexity of CT as a learning process.

Identification and Assessment of CT

The body of literature on defining CT is largely made up of theoretical contributions to the discussion, as reviewed above (Scherer et al., 2019; Ye et al., 2022). Descriptive or experimental research testing proposed definitions and frameworks through measurements is much less common (Fagerlund et al., 2021; Tang et al., 2020; Zhang & Nouri, 2019). Tang et al. (2020) conducted a systematic review of empirical studies and found that most tools used to assess CT were tailored to the specific context of the research, thus providing little insight on effectiveness of the intervention on CT as a general thinking skill. Moreover, most assessment tools followed a skill-based approach, that is, CT is assessed

based on the demonstration of a certain skill, whereas CT is broadly understood as a process of thought. This indirect assessment should evoke cautious interpretation of the results with regards to the assumed thinking involved (Brennan & Resnick, 2012; Boom et al., 2018; Bereiter & Scardamalia, 1987; Fagerlund et al., 2020; Grover & Pea, 2018; Tang et al., 2020). Moreover, instruments to measure CT independent from a specific learning environment, for example a programming language like Scratch, have barely been developed, let alone evaluated in terms of validity and reliability (Boom et al., 2018; Chen et al., 2017; Román-Gonzáles et al., 2018; Tang et al., 2020; Tsarava et al., 2019).

Some exemptions have emerged, for example the Computational Thinking Test (CTt), developed by Román-González (2015) as a generic CT test originally aiming at 12-13 year old students and eventually established and validated for assessment of 10-16 year old students (Román- Gonzalez et al., 2017; 2018). Subsequently, the CTt has been adapted to serve assessment of younger primary school students, resulting in a dedicated version for beginners (BCTt), especially applicable to students aged under 10 (Tsarava et al., 2022, Zapata-Cáceres et al., 2020). Psychometric measurements, performed at various stages of the development and validation process of CTt, have contributed to the understanding of CT as a construct. Results, for example, corroborated previous findings on the intersection of CT and other cognitive constructs, such as mathematical abilities, verbal and non-verbal reasoning, and general intellect (Mayer et al.; 1986; Román-González et. al., 2017; 2018; Tsarava et al. 2019; 2022). Although CT shares some properties with other constructs, varying levels of CT skills cannot be explained entirely by the variance of concurring skills, as shown by Román-Gonzáles et al. (2017) and Tsarava et al., (2022). These findings substantiated the identification of CT as a problem solving skill, distinct from other abilities. Furthermore, findings suggested alterations in the course of time of cognitive skills correlating CT. Whereas in younger primary school students CT correlated with numerical skills, this correlation was not found in older students (Román-González et al. 2017, Tsarava et al., 2022).

Román-González et al. (2018) found a strong correlation between the CTt and a measure of general problem solving, the Resolución de problemas (RP30)(Seisdedos, 2002), which is in accordance with established theory, identifying CT as a specific form of problem solving. Interestingly, students identified as high achievers by this research, showing much faster progress in programming compared to their peers as well as extremely high scores on the CTt, yielded only moderate scores on the RP30. This is another indication that underlying abilities of CT might not be evenly distributed over all developmental stages and levels of expertise. Noteworthy is that the CTt was developed under the assumption that code literacy (i.e., being able to read and write code) is an essential part of CT (Román-González, 2015). Moreover, the operational CT definition driving test development consisted of five computer programming concepts (i.e., basic sequences, loops, iteration, conditionals and functions & variables) and for most questions, the multiple choice answers were expressed as block-based

computer code. Therefore, the scope of this assessment tool might be narrower than most definitions of CT proposed, with a very strong focus on CS.

In comparison, the global annual Bebras challenge is aiming to engage primary and secondary school students in CT, without presuming any specific CS knowledge or programming skills, even though Bebras was developed as a tool to attract more students to computer programming (Dagiene & Stupuriene, 2016). Bebras challenges are developed for different age groups, ranging from 6-18 years old. Noticeably, Bebras is not conceived as an instrument for formal assessment, as each year's edition contains freshly developed tasks, not subjected to psychometric measurements and other validation. On the contrary, the tasks encompassing Bebras are formulated to evoke and develop CT in learners and should therefore be primarily viewed as a teaching tool (Chiazzese et al., 2019; Dagiene & Dolgopolovas, 2022). However, because Bebras is administered in over 70 countries with more than 3 million students participating (Dagiene & Dolgopolovas, 2022), it is yielding valuable data for researching CT. For example, Araujo et al. (2019) investigated if psychometric evidence could be found of five distinct concepts underlying CT, as proposed by Selby and Woollard (2014) and adopted to guide Bebras test development (Dagiene & Stupuriene, 2016). Araujo et al. (2019) only found two separate underlying constructs, which did not seem to be unambiguously distinct. The researchers suggested that this could be explained by the nature of Bebras tasks, namely that solving those tasks most probably involves more than one of the abilities underlying CT, rendering it impossible to detect separate abilities by statistical factor analysis. Based on principal component analysis and the classification of tasks according to the developers, the research proposed that the two constructs underlying CT could be labelled *evaluation ability* (including abstraction, generalisation and decomposition) *and algorithmic thinking and logical reasoning.*

Because generic measures such as the CTt and Bebras can be applied irrespectively from the specific learning environment, they allow for comparison across learning- and research contexts.
Besides the CTt and Bebras, the Computational Thinking Scale (CTS), developed by Korkmaz et al. (2017) has also been used in primary educational context. As this instrument is entirely based on self-assessment, its relevance for experimental research is limited. Although labelled by its developers as an instrument to measure CT skills, CTS more appropriately qualifies as a measurement of self-efficacy in relation to CT.

Qualitative Exploration of CT

Another way to deepen understanding of CT is to qualitatively investigate the process of computational problem solving, for example by examining thinking aloud protocols or comparing how more and less successful students engage in the problem solving process (Fagerlund et al., 2021; Falloon, 2016; McCoy Carver, 1988). Such approaches can bypass the disadvantages of assessment purely based on

test results mentioned above (Scardamalia et al., 1984). Qualitative approaches are often time consuming and therefore difficult to implement on a scale that warrants generalisation, yet they can offer insight in how learning processes occur and what elements might be worth further investigation.

For example, van der Linde et al. (2021) qualitatively evaluated CT in 6-12 year old students working on a problem solving task involving the programming of an educational robot. Thinking aloud was encouraged by letting the students work in pairs and video recordings were captured to enable detailed evaluation of the problem solving process. In general, younger students showed less sophisticated CT, in correspondence with lower task completion. Evidence of decomposition skills started to clearly emerge from around 8 years of age, and around the age of 10, students started to show sustained CT on al indicators (i.e., decomposition, abstraction, testing and organizing data). Especially debugging tasks appeared to be very difficult across all ages, as found by several other researchers as well (Falloon, 2016; McCoy Carver, 1988; Zhang & Nouri, 2019). In successful pairs, the researchers observed more behaviour constituting CT, such as visualising, planning, deductive reasoning, organising the data, goal setting and evaluation, whereas less successful pairs predominantly relied on unstructured trial and error. However, van der Linde et al. (2021) did not investigate to what extent CT related problems impacted successful task completion compared to other types of problems, such as ineffective colouring of the coding strips that the robot had to read. This could be a relevant issue, given that only 18 of 42 pairs were able to complete the task successfully. Moreover, this research was not carried out in normal class settings, because one pair at the time worked on the task in a separate room, limiting environmental validity of the study.

Falloon (2016) also studied conversations of pairs of students working on a programming task to obtain insight in CT processes, but these students were younger in average (i.e., between 5 and 6 years old) and worked in regular class settings. The study aimed to investigate which thinking processes could be identified in students while programming, arguing for programming as a way to consecutively advance development of those types of thinking. The study found evidence of low generalisation skills in this age group, as well as great difficulties in conducting debugging tasks, mentioning for example couples that simply re-ran ill-functioning code over and over again, or couples that only seemed to make random changes without any strategic approach. This is in accordance with findings of van der Linde et al. (2021) as discussed above, even though Falloon (2016) did find some evidence of emerging predictive thinking, a thinking process aiding debugging strategies (McCoy Carver, 1988). In addition, Falloon (2016) did identify fairly sophisticated decomposition strategies, with couples commonly reorganising the task at hand in manageable subtasks, efficiently advancing the problem solving process. This might indicate that even students as young as 5 year old could successfully use decomposition as a CT strategy, provided the task at hand is in accordance with their level of development. However, because this study did not relate observed thinking to problem solving

success, little evidence can be derived as to what extend various types of thinking inform CT. Moreover, it remains difficult to estimate the generalisability of its findings, due to the lack of quantification of the obtained data.

Programming Lessons to Develop CT

Over the last decade, implementation of CT related programs in compulsory education has increased, as well as experimental research examining its effects on students' learning and development (Fagerlund et al., 2021; Scherer et al., 2018; Tang et al., 2020; Tsarava et al., 2019; Wing, 2017; Ye et al., 2022). Experimental research on CT has predominantly focussed on the presumed transferable qualities of CT across situations and domains, that is, the investigated educational interventions aimed to improve learning outcomes beyond the narrow topic of instruction. For example, Tsarava et al. (2019) developed a ten-week CT course for 7-10 year old children participating in an extra-curricular study program for gifted children in Germany (n=31). The study revealed significant gains in CT, comparing pre- and post-test scores on the previously discussed CTt. However, as participants were mostly boys who were not randomly selected, the results cannot be generalised. Nevertheless, the study exemplifies how well-designed learning activities can indeed enhance at least some aspects of CT (i.e., as measured by CTt) in young learners.

Similar effects were reported by Kwon et al. (2021), who designed CT lessons aimed at 6th grade students (i.e., 11-12 year old) at four public schools in the United States, following a problem-based learning (PBL) approach. The study found that after the introductory programming lessons, significant learning gains in CT were observed and that students had significantly improved their understanding of almost all concepts. Interestingly, the study reported larger learning gains for students that scored relatively low on the pre-test, compared to students that scored relatively high. Moreover, where the high scoring group continued to improve on tasks involving debugging, the low scoring group lost some of their previous learning gains during the PBL phase that followed after the introductory programming lessons. These findings show that programming instruction in Scratch can indeed foster understanding of certain CT concepts in both students with high and low prior knowledge. Furthermore, students with lower prior knowledge seem to benefit especially from structured programming lessons, whereas PBL predominantly seems to enhance learning in students with higher prior knowledge.

In a Chinese primary educational context, Ma et al. (2021) did also find increased scores on CTt after 14 weeks of problem based programming lessons in Scratch, as well as increased self-efficacy scores as measured by CTS (see above). A control group that participated in programming lessons with the same learning content, but following a reportedly more traditional Chinese educational approach, with students mainly listening to the teacher and copying his actions, did not gain in CT or self-efficacy.

The study found that girls benefitted to a greater amount of the problem based programming lessons, narrowing the existing gender gap in CTt performance from pre- to post-test. Although girls had significantly lower prior knowledge scores than boys, the problem based approach in this study did not seem to hamper their learning, contrary to findings of Kwon et al. (2021) discussed above.

As a final example of recent research measuring CT effects of programming education, Chiazzese et al. (2019) found that Italian third and fourth grade primary school students benefitted from participating in a robotic laboratory, where they build Lego robots and programmed them using a visual programming language resembling Scratch. The CT-scores, as measured by a set of Bebras tasks (see above), of the experimental group were significantly higher, compared to the scores of a control group of students from the same school and the same grades, but from classes that did not participate in the robotics laboratory. Although the study did not include a pre-test, it assumed that higher scores in the experimental group were due to the robotics intervention. Interestingly, third and fourth graders reached similar CT scores after participating in the robotic laboratory, whereas third graders in the control group scored significantly lower than their fourth grade counterparts, pointing to greater learning gains in third grade participants in the robotic laboratory. In other words, participants with presumably lower prior knowledge might have benefitted more from the programming intervention. As an explanation, the researchers suggested that the robotic challenges could have been too easy for grade four students to stimulate CT, whereas third grade students were appropriately challenged to developed their CT.

In conclusion, all presented studies in this section substantiated claims of programming learning effects beyond the actual application of learned programming skills in the initial learning context. These effects largely qualify as so called *near-transfer* effects, because the measurement of CT was closely related to the initial programming tasks (Barnett & Ceci, 2002; Tang et al., 2020). They are consistent with earlier findings presented by Scherer et al. in their 2018 meta-analysis of transfer effects in computer programming education.

Programming Lessons Aiming for Learning Gains in Literacy

Although near-transfer of skills is fairly undisputed in literature, less agreement exists on the topic of *far-transfer*, that is, the transmission of CT skills across domains, for example from computer programming into literacy (Grover & Pea, 2018; Scherer et al., 2020; Tedre & Denning, 2016). Some scholars have stated that this type of transfer does most probably not exist (Mayer et al., 1986; Tedre & Denning, 2016). Others have argued for the reasonable probability of CT transfer across domains, while acknowledging the need to expand experimental research in order to develop a better understanding of the phenomenon (Grover & Pea, 2018; Jacob & Warschauer, 2018; Rodríguez-Martínez et al. 2020; Scherer et al. 2018). Presenting findings of their aforementioned meta-analysis,

Scherer et al. (2018) stated that they "could not confirm the doubts surrounding the far transfer of programming skills", as they found convincing evidence of both near- and far-transfer. Since the publication of this meta-analyses, more studies have emerged that focussed on transfer of CT skills into the literacy domain.

For example, a cross-disciplinary programming approach was researched in the Netherlands by Yeni et al. (2022), who presented research showing that digital storytelling in Scratch can offer an opportunity to practice both (foreign) language and computer programming at the same time. The study reported that six out of nine groups of collaborating students successfully completed a digital story, but no inferences were made as to which elements had benefitted or hampered successful task completion. Although the study aimed to investigate the effect of digital storytelling on learning outcomes in English and CT, measurements were based on self-assessment, without including a pre-test or a control group for comparison. Therefore, little can be concluded about the effectiveness of the lessons and assertions about improved skills in creativity, language and CT were not substantiated by this study's findings. However, the study specifically exemplifies how students can be guided in decomposing a storytelling task into smaller subtasks, concurrently fostering meaningful decomposition of programming tasks, thus pointing to opportunities for transfer of CT skills across domains.

Another study focussing on digital storytelling in Scratch in the context of learning English as a second language was conducted by Parsazadeh et al. (2021) in Taiwanese fifth grade students. In the experimental condition, students received programming lessons focussing on CT strategies (e.g.: decomposition), and used Scratch to develop a digital story. These students showed higher learning gains in grammar and vocabulary, as well as better storytelling, compared to a control group of students who followed the standard curriculum, without programming lessons. However, the findings of this study should be met with caution, as some issues arose. Firstly, the large difference in the standard deviation of the results on the grammar- and vocabulary pre-test between the experimental group (M = 30.62, SD = 3.37) and the control group (M = 29.04, SD = 6.08) points to unequal distributions of English language ability in both groups, possibly resulting in tainted findings. Furthermore, the scoring rubric that was used to evaluate the storytelling assignment about daily life events was developed to evaluate a different writing goal (i.e.: a persuasive essay), within a different age group (i.e.: grade three and four students) and targeted at first language learners instead of second language learners. Therefore, questions remain on the validity of the measurements of storytelling abilities in this study. Altogether, the study of Parsazadeh et al. (2021) should not be considered strong support of CT augmenting literacy abilities.

Hassenfeld et al. (2020) presented a study in second grade students from Virginia, USA, who participated in a series of lessons integrating literacy activities and programming activities using the

educational KIBO robot. This study showed a weak positive correlation between early literacy scores and KIBO programming skills. Although results indicated some prerequisites in literacy might exist in order to succeed in programming, no evidence was provided corroborating the claim of both skills mutually supporting each other. Extended research of the same series of KIBO lessons was presented by Bers et al., (2022), who, diverting from Hassenfeld et al. (2020), did not report a correlation between early literacy and KIBO programming skills, but found that measurements of early literacy skills were predictive for CT-related problem solving. Similarly as in the aforementioned study of Hassenfeld et al. (2020), these presented findings point to some shared properties between literacy and CT abilities, without additionally providing evidence of transfer of skills between the domains of programming and literacy.

To summarise, findings from the latest research substantiate urgings for a sustained critical attitude towards claims of evident transfer of CT skills across domains, especially regarding literacy. However, this does not necessarily reflect general absence of transfer of CT, but is rather pointing to scarcity of sound research on the topic. In a systematic review of CT transfer literature, only three studies were found that included literacy as domain of interest, out of 55 interventional studies qualifying for analysis (Ye et al., 2022).

Teaching for Transfer

Even though the properties of CT remain under discussion, in all proposed definitions and frameworks, CT inherently assumes transfer, as its pertinence is not in writing flawless computer programs, but rather in solving novel problems in all possible domains. Transfer has been a topic of interest within educational science and psychology for over a century (Barnett & Ceci, 2002). Even before CT as a concept surfaced, research in CS and other domains had already identified *content*, *context* and *educational design* as influential factors regarding transfer of learning.

Content characteristics that affect transfer. Learning content can be very specific, for example on how to use *wait blocks* between actions in programming in Scratch, to make an animation look better on screen. In comparison, learning content can be more general, for example on determining what part of the code is repeated when drawing a geometrical figure. General principles, in the given example pattern recognition, have shown to transfer more likely across contexts than specific content knowledge. Notably, it is unlikely for students to derive general principles from content specific tasks and examples on their own (Barnett & Ceci, 2002; Engle et al., 2012; McCoy Carver, 1988; Salomon & Perkins, 1989). Even more so, the initial learning content has to be mastered beyond mere application of a procedure, that is, at a deeper level of understanding, before opportunity of transfer arises (Barnett & Ceci, 2002; McCoy Carver, 1988; Perkins & Salomon, 2012).

Contextual factors that affect transfer. In general, transfer is more likely to happen between more similar contexts (Barnet & Ceci, 2002; Perkins & Salomon, 2012). So far, this research has distinguished between near- and far-transfer based on whether the transfer occurred inside or outside the original learning domain, thus taking only one contextual dimension into account. However, many more contextual elements are establishing the likelihood of transversal learning, including social circumstances and physical aspect of the surroundings (Barnett & Ceci, 2002; Engle et al., 2012; Ye et al., 2022). Barnett and Ceci (2002) have provided a taxonomy of far transfer including six contextual dimensions, aiming for a better-informed evaluation of transfer, or failure of transfer, observed in education and research. In addition to (1) knowledge domain and (2) physical and (3) social context, they identified the (4) functional context as factor of influence, pointing to the difference between formal school settings and more informal settings. Properties of the transfer task (5) should be considered another relevant contextual factor. For example, a paper-pencil based CT test is a task format quite dissimilar from solving computer programming problems in Scratch and as such could be an example of far-transfer along the so called modality dimension. The last contextual factor included in the taxonomy is (6) temporal dimension, denoting the proximity in time between the initial learning and the observation of transfer effects.

Educational design aiming for transfer. Through educational design, the content and context of learning can be deliberately manipulated to optimize both direct learning outcomes and transfer. This is of utmost importance, because broad consensus under researchers exists that transfer should not be left to chance, as it is extremely unlikely to happen incidentally, that is, without guidance of the process (Barnett & Ceci, 2002; Engle et al., 2012; Grover & Pea, 2018; Littlefield et al., 1988; McCoy Carver, 1988; Meyer et al., 1986; Salomon & Perkins, 1989; Ye et al., 2022).

Content selection. When teaching for transfer, the educational design process should start with identifying transferable elements of learning, thereby focussing on general principles that could aid problem solving across domains and situations, also labelled as *high-level problem solving skills* (e.g., McCoy Carver, 1988) or *higher-order thinking skills* (e.g., Falloon, 2016; Grover & Pea, 2018; Scardamalia et al., 1984; Tang et al., 2020). Subsequently, the targeted skill must be explored and all thinking steps involved precisely plotted, in order to design for explicit instruction of the skill (McCoy Carver, 1988). However, the targeted skill must not be practiced in isolation solely, as a stand-alone task, but in interdependency with other sub-tasks that are comprised in the bigger learning task at hand (i.e., programming). In other words, the execution of the targeted skill must remain a meaningful part of a bigger picture, or students will not be able to integrate the skill in situations that are not very similar to the practice situation (Kirschner & Van Merriënboer, 2008).

Enlarging context similarity. Contextual learning factors can be purposefully designed to enlarge similarity between initial learning and transfer situations. Domain integration, as applied in

several of the discussed articles in the preceding sections, inherently increases context similarities (e.g., Bers et al., 2022; Rodríguez-Martínez et al., 2020; Yeni et al., 2022) and as such, could foster transfer through so called *bridging* between circumstances of initial learning and transfer (Perkins & Salomon, 2012).

Even without full integration of domains, expansive framing of the learning content, as proposed by Engle et al. (2012), serves as a bridge to enable transfer of learning between domains and situations. Essentially, expansive framing corresponds to establishing connections (i.e., bridging) between the present learning situation and any possible future transfer situation, by using language that points to usability of the learning content far beyond the bounded situation of the learning at hand. Examples of this kind of language would include relating the present learning to past and future learning (i.e., bridging temporal contexts), hinting at alternative settings in which the present learning could be applied and asking students to think of other examples (i.e., bridging domains and/or physical and functional contexts) (see also: Littlefield et al, 1988). The design of a variety of tasks in which the students deploy varying roles (e.g., group discussion leader, cooperation partner, independent learner) contributes to expansive framing on the dimension of social context, just as crediting a wide variety of contributions to the learning process, from inside the classroom (e.g. students contributions to the discussion) and from outside the classroom (e.g. specific insights derived from forgoing discussions with colleagues).

More generally, offering students opportunities for practice through a wide variety of tasks enlarges the likelihood of transfer (Kirschner & Van Merriënboer, 2008). Notably, Engle et al. (2012) suggest that expansive framing in general could be a time-efficient substitute for content specific learning design aiming for transfer of a selected skill. However, based on synopsis of the body of literature, combining expansive framing to create a learning context generally making students more susceptible for transfer with learning design targeting a specific skill to facilitate its anticipated transfer into another context appears to be the adequate direction.

Process Similarities Between Programming and Writing

Comparing the domains of CS and literacy, at first glance mutual differences seem more apparent than resemblances and therefore, transfer between the two would qualify as far-transfer (Barnett & Ceci, 2002). Likewise, comparing paramount practices from both domains, programming and writing (i.e., text composition) might seem to share little commonalities. However, on a process level, they reveal more resemblance than appears from superficial comparison. Specifically, both writing and programming are creative processes that require *iterative, recursive and parallel* thinking, elements that are widely considered to be embodied in CT (Brennen & Resnick, 2012; Fagerlund et al., 2021; Grover & Pea, 2013; 2018; Lodi & Martini, 2020). For example, Brennan and Resnick (2012) underscore

how programmers need to apply an adaptive approach, redesigning already invented solutions in response to their advancing insight in the course of project development.

A major challenge in computer programming is managing a broad variety of sub-processes, such as evaluating the code written-so-far for effectivity and efficiency and systematically planning for improvements (Perkins et al., 1988). In similar vein, text writing requires simultaneous management of sub-processes, such as generating new ideas, putting ideas into words and evaluating current writing (Bouwer et al., 2018; Hayes, 2012; Scardamalia & Bereiter, 1991). As such, both writing and programming are characterised by intensive information processing, as input from various sources must be evaluated, essential information abstracted and generalisations made in order to decide on how to continue the process at hand. A first source of such information is the composed text or program under construction, in its unfinished state. Secondly, internal content- and context related knowledge informs both processes, for example on the targeted audience/user, the goals aimed for and restrictions that apply, and procedural knowledge on how to perform writing or programming tasks that are stored in long-term memory. Thirdly, depending on the situation, information derived from external sources, such as task requirement descriptions, feedback from colleagues or content specific literature has to be processed to serve the ongoing writing or programming process.

To resume, both processes are characterized by the same sort of complexity, with high probability of cognitive overload in learners (Bouwer et al., 2018; De Jong, 2010; Pea & Kurland; 1984). This complexity is regarded to hold difficulties as well as opportunities for learning. Poor learning outcomes have been reported and commented extensively in both fields (e.g., Bouwer et al., 2018; Grover, Pea, & Cooper, 2015; Pea & Kurland, 1984; Perkins et al., 1988). At the same time, both programming and writing have been proposed as educational means to foster *knowledge creation* in learners, as opposed to activities merely serving the transmission of knowledge. Scardamalia and Bereiter (1991; 2010; see also Scardamelia et al., 1984) have extensively researched the writing process and signalled the difference between *knowledge telling* and *knowledge transformation* in writing. Whereas young or inexperienced writers tend to put down on paper one idea or piece of information after another (i.e. knowledge telling), more experienced writers engage in a process of ordering, combining and altering ideas to produce a coherent text that reflects their understanding of the content. This process, on its turn, leads to new ideas and refined understanding (i.e. knowledge transformation) (Graham & Harris, 2016; Scardamalia et al., 1984).

To reap the fruits of knowledge building trough writing, authentic writing tasks have to be provided, in order to engage learners in the rich problem solving process of writing a text that is meaningful to them. *Authentic learning tasks* are referring to assignments that resemble real-world-problems as much as possible, in all their complexity. (Bouwer et al., 2018; Kirschner & Van Merriënboer, 2008; Scardamalia & Bereiter, 2010). In the same manner, programming activities that

go beyond simple task completion by narrowly following instructions or copying procedures, but instead are reflecting creative problem solving, hold the potential of knowledge transformation and thus higher learning gains (Brennan & Resnick, 2012; Grover et al., 2015). Essentially, this kind of knowledge transformation could be viewed equivalent to transfer effects ardently strived for in education and research, as this is where learning happens beyond the topic of instruction.

Computational Thinking Process Model

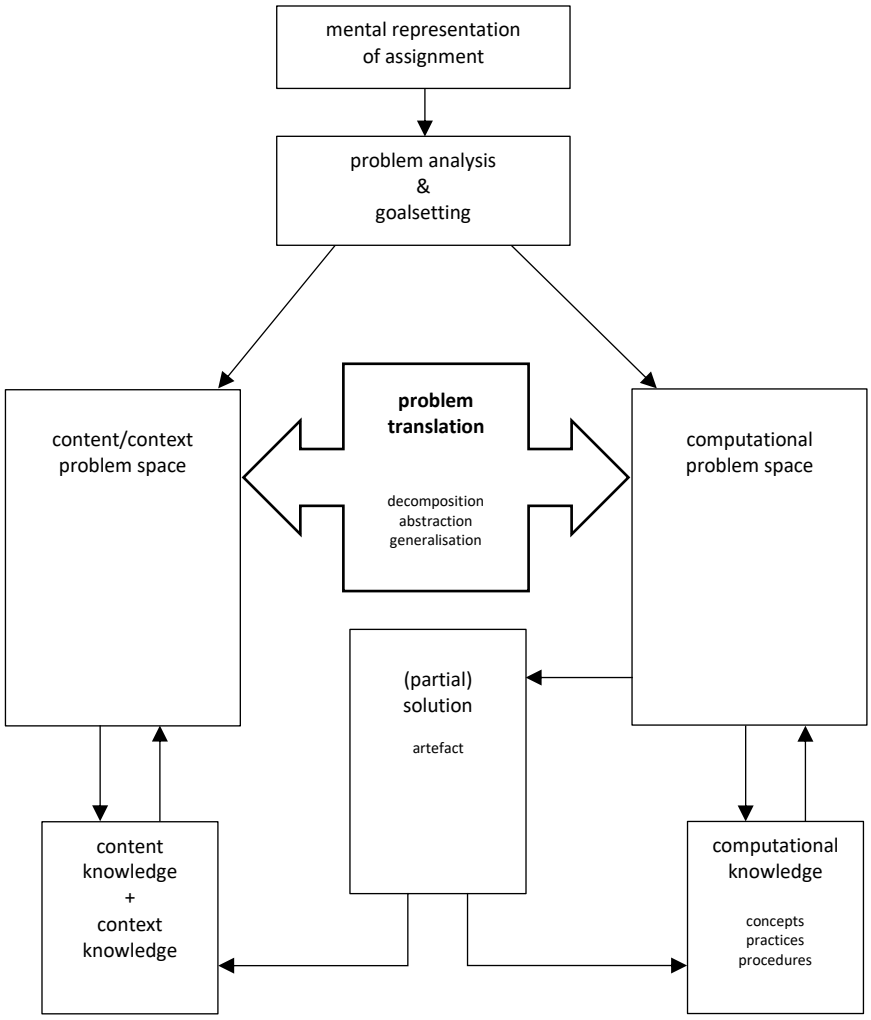
This research operates under the assumption that CT and writing are resembling problem solving processes, both serving the production of a creative product or artefact, by applying and monitoring a variety of sub-processes. CT is defined by this research as *the complex thought process of designing a computational solution*. Furthermore, it is considered that CT is enacted trough computer programming in Scratch, a block-based computer language developed by the Massachusetts Institute of Technology (Brennan & Resnick, 2012). Additionally, this research proposes the Computational Thinking Process Model (CTPM, see Figure 2), an adaptation of Scardamalia and Bereiters (1991) model of the writing process, to represent the CT process. Furthermore, based on similar complexities of CT and the writing process, it is assumed that both CT and writing would benefit from *structured problem decomposition*, an element that is mentioned in most definitions of CT and entails breaking down a complex problem into smaller parts, that could be solved more easily one after the other.

Problem translation in CT. As exhibited by the CTPM, central to the CT process is the occurrence of *problem translation*, a process of dialectical exchange of thought between two different *problem spaces*, namely the computational problem space on the one hand and the problem space involving content and context on the other hand. In both problem spaces, choices have to be made on how to continue the problem-solving process at hand. In the *content/context problem space*, ideas are formulated on what to achieve, more specifically, on how an effective solution would look like and function in the given context. These thinking processes result in the drafting of objectives, to be passed to the *computational problem space*. In the computational problem space, ideas are posited about what computational practices could be employed, what computational concepts apply, or, at a more practical level, how to write (i.e., code) procedures that advance a solution. If no immediate solution can be achieved, judgements have to be made about what information is failing. These thinking processes result in the formulation of questions to be referred to the content/context problem space. It is apparent that no computational solution or artefact emerges when all thinking processes remain in the realm of the content/context problem space, whereas it is possible to produce computer code without posing questions on the content/context problem space. For example, a student that has some experience in using Scratch might know what blocks to use to achieve a certain effect when executing the code, and might decide to experiment by randomly changing the parameters to observe different

effects. Although the students is making use of some rudimental programming knowledge, according to the proposed model, no computational thinking is involved effectively. Students that are using unstructured trial-and-error as a debugging strategy are another example of 'being stuck' in the computational problem space. They probably fail to identify the bug, because no question is referred to the content/context problem space as to how exactly the result of executing the code is differing from the desired result, and therefore, no inferences can be made on where the bug could be located (McCoy Carver, 1988).

Figure 2

Computational Thinking Process Model



Knowledge transformation trough CT. Provided that iterations of translations between both problem spaces do occur, knowledge transformation (i.e., transfer of learning) is likely to result. For example, the computational problem of having to specify the turning angle of the robot you are programming may result in a better understanding of unit-circle degrees, or the computational challenge to use as little programming blocks as possible to make your object draw a specific geometrical shape may prompt pattern recognition (i.e., near transfer) and consequently result in more conscious geometrical knowledge (i.e., far transfer). At a professional level, a developer of adaptive educational software can solve the problem of how to make a learning prompt pop up on screen solely within the computational problem space. However, to make the program select a learning prompt that is relevant to the user, considerations about the learning process have to be made within the problem space of content and context, urging the developer to close eventual knowledge gaps. In general, the CTPM holds that increasing the frequency of problem translations between problem spaces (i.e.: domains) leads to more transformation of knowledge, thus promoting transfer of learning.

Research Questions and Hypotheses

Based on the CTPM, the aim of this research is to explore how instruction in problem decomposition, could facilitate the CT process. Furthermore, the research aims to investigate the transfer of decomposition ability from the CS domain into the literacy domain. The research questions it seeks to answer are:

1. Does instruction in structured problem decomposition during programming lessons lead to better programming results in fifth- and sixth grade Dutch primary school students?
2. Does instructing fifth- and sixth grade Dutch primary school students in problem decomposition during programming lessons lead to better problem decomposition in writing, compared to students who received programming lessons without decomposition instruction, or to students who did not receive programming lessons at all?
3. Does programming instruction without structured problem decomposition lead to better problem decomposition in writing in Dutch fifth- and sixth grade primary school students, compared to students who did not receive programming lessons?
4. Can differences between boys and girls and between fifth- and sixth grade students with regard to the effect of decomposition instruction on programming results and with regard to the effect of computer programming lessons, with or without decomposition instruction, on the quality of decomposition in writing, be observed in Dutch primary school students?

In accordance with the CTPM, it is expected that students who received instruction in decomposition achieve better results in programming, as decomposition is an essential sub-skill of CT that facilitates the process of problem translation, that in turn enables knowledge transformation. Furthermore, students who participated in the programming lessons are expected to score higher on decomposition in writing, as a result of transfer of decomposition proficiency, attainted through programming, into the literacy domain. Moreover, students who's programming lessons were augmented with targeted decomposition instruction are expected to achieve the highest scores of all participants on decomposition in writing.

Because CT correlates to cognitive development (Román-González et al., 2017), age effects on the learning of programming and decomposition are hypothesized. More specifically, as grade six students have more prior knowledge, it is expected that they will benefit more from programming lessons and decomposition instruction than grade five students. As the literature is inconclusive about the influence of gender on CT (Sun et al., 2022), no hypothesis is formulated on differences in findings between boys and girls.

Method

Study Design

To analyse the effects of instruction in computer programming and structured decomposition on both programming skills and decomposition ability in writing, this study adopted a quasi-experimental post-test only design with two experimental conditions and a baseline comparison group. The study was carried out over a six week period. The students assigned to experimental conditions A and B received weekly 70-minutes programming lessons in the first four weeks. In condition A, the programming lessons were augmented with decomposition instruction. In condition B, the students participated in a filler assignment instead of receiving decomposition instruction. In the fifth week, students from condition A and B took a programming test. In the last week, students from all conditions participated in a decomposition test in the context of writing (i.e., composing a text) (See Table 1 for an overview).

It was decided not to include pre-testing in the study's design, and to rely on a baseline group for comparison of the results on the decomposition test in writing instead. Including a pre-test could have obscured the findings of the study, as it could have served as an additional instructional moment, rendering it difficult to measure the effect of the intended intervention. Regarding programming skills, no pre-test was included because prior to the programming lessons, students had no programming knowledge to apply to such a test.

Table 1

Study Design

	Week 1-4	Week 5	Week 6
Baseline	No intervention	No intervention	Decomposition test in writing
Condition A	Programming lesson + decomposition instruction	Programming Test	Decomposition test in writing
Condition B	Programming lesson + filler assignment	Programming Test	Decomposition Test in writing

Sample

Data was collected from 137 students at two Dutch elementary schools in the rural eastern Netherlands operating under the same school board. Students came from two fifth-grade classes, two sixth-grade classes and one combined fifth- and sixth-grade class. Two classes were randomly assigned to serve as baseline group and three classes were randomly assigned to serve as experimental classes. Each of the three experimental classes were randomly divided into halves, and each halve of a class was randomly assigned to one of the two experimental conditions.

Altogether, this resulted in three groups of participating students: the baseline group (*n* = 58, *M* age = 10.97, *SD* = 0.62), experimental condition A, receiving both decomposition and programming instruction (*n* = 40, *M* age = 11.24, *SD* = 0.74) and experimental condition B, receiving programming instruction only (*n* = 39, *M* age = 11.35, SD = 0.58).

Interventions

Programming lessons (condition A and B). All lessons were taught by the same researcher, a qualified elementary school teacher. Due to limited wi-fi capacity, the students at one school used laptops and the offline version of Scratch 2.0 (*n* = 21). Programming results were saved to the hard drive and copied to individually labelled memory sticks by the researcher afterwards. At the other school, students used Chromebooks and the online version of Scratch 2.0 (*n* = 58). Their work was saved to personal Scratch accounts dedicated to this research and downloaded by the researcher afterwards. The students did not have access to their accounts outside programming lessons.

Each programming lesson contained two unfinished Scratch programs for the students to work with on their own devices, after a lesson phase with teacher led, whole class exploration and instruction on how to solve a specific programming problem. The unfinished Scratch programs contained several (sub)tasks (i.e.: completion problems), and served as a scaffold so that even beginner programming students could work on fairly sophisticated programming problems in the context of an authentic programming task. In doing so, the lessons provided the necessary whole-task practice when learning a process as complicated as programming.

Figure 3 presents an example of a programming task (in Dutch), in which students had to use loops to complete the code that draws the requested pattern, which was provided as an illustration on the white board in front of the class (see Figure 4). Upon opening the unfinished Scratch program, the students were provided with a selection of programming blocks on the programming canvas, annotated with instruction on how to complete the program. The (sub)tasks were numbered to facilitate identification, as in some cases an annotation was also used to only explain the function of a certain piece of code. In the example presented, the first task was to identify the piece of code that could be used to draw a triangle and finish it by adding a loop with the right amount of repetitions. The second task was to identify the code that, if executed, results in the next triangle to get a different colour and a different orientation. This code too had to be completed with a loop, but this time, the amount of repetitions (12) was specified. The third task was to combine both pieces of code (i.e.: nest the loops) to arrive at the desired pattern. After each task, the students were prompted to test their code. At the end of an annotation, students are prompted to continue with the next task, including a

hint on where to look for it or where to click to find it, as sometimes they would need to go to another programming canvas (within the same program) to continue finishing the assignment.

Figure 3

Programming Task from the fourth programming lesson

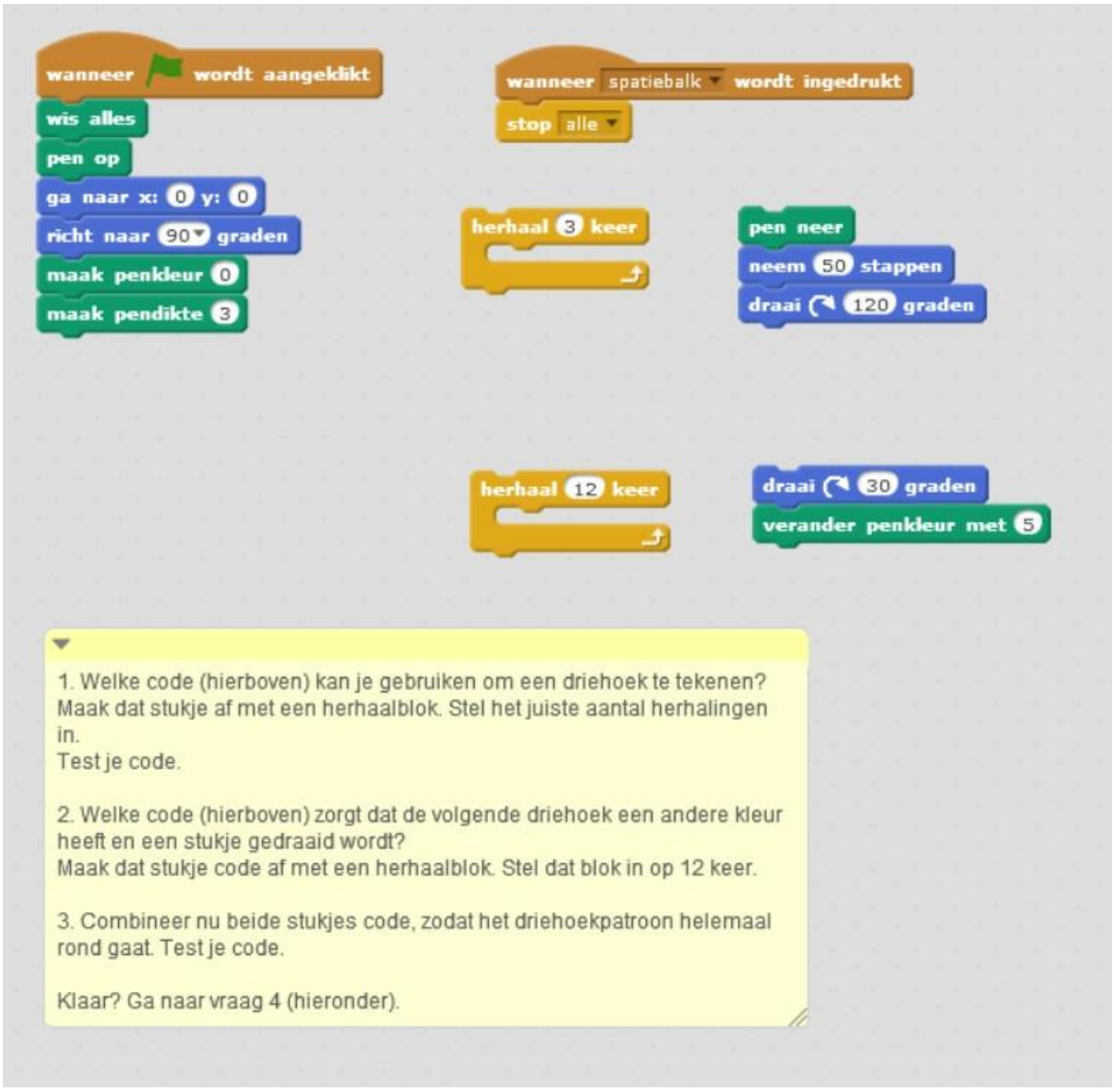
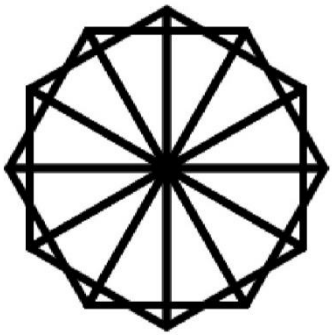


Figure 4

Requested pattern of triangles



All students had to complete the programming problems on their own device, but were encouraged to discuss problems and strategies with their neighbouring peers. Students were not allowed to touch each other's devices, in an attempt to grand all students the experience of autonomous programming and to promote formulation of problems and strategies, instead of having the bolder or quicker students simply fixing their classmates problems and coincidently taking away their opportunities for learning.

For the largest part, the lesson content in both experimental groups was similar, except for a 20 minutes instruction period in the middle of each lesson. In this phase, students from condition B participated in a filler-assignment, focussing on aspects of CS not directly related to programming, which was anticipated not to influence programming ability. For example, the students were introduced to Charles Babbage as the inventor of the first computer and Ada Lovelace as the first computer programmer.

Decomposition instruction (condition A only). Students of experimental condition A received decomposition instruction during the 20 minutes instruction period in the middle of each lesson. The decomposition instruction was tailored to the principle of expansive framing of the learning content, to enhance the opportunity for transfer (see above). During these instruction periods, various examples of problem decompositions were explored and represented in decomposition frames. The implementation of decomposition frames was adopted from Code-IT, the CS study program based on Scratch for UK primary schools (Bagge, 2015) (see Figure 5 for a general example of such a decomposition frame). At the start of the instruction, the teacher presented a real-world problem and explored with the students how this problem could be broken up in smaller problems that could be solved more easily. The students offered suggestions for components and the teacher included those in the problem decomposition frame presented on the smartboard (see Appendix A for a filled out example, in Dutch).

Figure 5.

Decomposition Frame

(Ideas for) Solutions → ↓				
	Subgoals/ Subtasks → ↓			
		Goal/ Assignment		

The real-world problems were chosen to have some structural resemblance to the programming problem of the day. For example, during the lesson in which the programming content focussed on parallelism and coordination between objects, the real-world decomposition problem derived from writing a simple recipe, prompting the identification of linear and parallel processes. Moreover, the programming lesson focussing on pattern recognition and repetition was accompanied by a real-world decomposition problem of writing a farewell song for a teacher who was leaving the school, linking the alternation of verses and chorus to programming patterns and re-use of code. Those real-world problem decompositions served as expansive examples and were not arduously explored. Instead, after concise whole-class exploration, the students worked individually on the structured decomposition of a problem related to the programming lesson content of the day. For this purpose, they received a work sheet with a decomposition frame adjusted to the specific problem, that is, with the specific problem description provided in the centre (see Appendix B for an example in programming context, in Dutch). The worksheets were collected by the teacher, but not further assessed in the context of this research.

Instruments

Programming test. To assess the programming abilities of the students after four weeks of programming instruction in Scratch, the students were presented with three incomplete Scratch programs in which they had to adjust and complete pieces of code. Students were used to this format, as equal task formats were used in all programming lessons, the only difference being that they had to work strictly individual during the fifth lesson, in which the measurement was conducted. As in the previous programming lessons, each Scratch program contained various pieces of code, some of them to leave untouched by the students. Again, the pieces of code were annotated, explaining the students what to do with them, and programming tasks were numbered, to facilitate identification. Noticeably, although all problems presented in the programming test resembled problems the students had previously encountered during the programming lessons, certain aspects of the tasks differed. For example, during the programming test, students had to build code to draw a pattern consisting of squares, whereas in the previous lesson, the students had worked on a triangle-based pattern. This way, students could not solely rely on memory while completing the problems and therefore, the results of the programming test were assumed to reflect conceptual understanding and CT proficiency.

Each incomplete program contained three programming tasks and completing all tasks would lead to a fully functional program. However, as pieces of code can be executed in Scratch by clicking on them, it was possible for students to experience some success, even if not all tasks were completed. Supposedly, this helped them to remain motivated while working on the series of tasks. By replacing students names with randomly generated codes, students work was pseudonymised before evaluation.

Scoring of the programming test. A codebook was created to guide the rating of the students' work (see Appendix C, in Dutch). For each task, the optimal solution was described and an illustration of the associated code was provided. This solution would yield three points, whereas suboptimal solutions would yield two, one or zero points. Through expert consultation, prerequisites for each score were determined. The flawless completion of a program, with three distinct programming tasks, would lead to the perfect score of nine points. The total programming score was calculated by adding up the scores for all three programs, resulting in a total programming score between 0 and 27.

To estimate interrater reliability, two raters independently scored all programming tasks of five randomly selected students (n = 45). Intraclass correlation coefficient (ICC) estimates and their 95% confident intervals were calculated using SPSS, based on a single rating, absolute-agreement, 2-way mixed-effects model. This resulted in an unsatisfying estimated ICC of .628 (95% confidence interval = .411 - .777). Therefore, both raters consulted to arrive at a shared understanding and adjusting the code book for unambiguity. For example, the criteria for flawless game control (yielding

three points) were specified by the exact elements that had to be included in the script. The distinction between moderate control (two points) and limited control (one point) was no longer left to interpretation of the raters, but specified by the amount of imperative elements present in the script. Consequently, both raters independently coded a new set of programming tasks of six randomly selected students (*n* = 54). This time, results revealed an estimated ICC of .966 (95% confidence interval = .943 - .980), indicating excellent interrater reliability (Koo & Li, 2016).

Decomposition in writing test. Students' decomposition ability in writing was assessed during 50 minutes literacy lessons in normal class settings. All classes were taught by the same researcher, who had also taught the computer lessons. In this lesson, the students were introduced to the decomposition frame as a mean to aid their writing, labelled as their *writing schedule* (see Appendix C, in Dutch). Students from the decomposition experimental group would recognise it by appearance, and to them, it was acknowledged that the writing schedule was essentially a decomposition frame. To all experimental groups, firstly, an example of writing a birthday party invitation was thoroughly explored through class-based, teacher led discussion. The elements of the decomposition were identified and added to the decomposition frame on screen by the teacher, modelling how to use the frame. After that, students were introduced to the assignment that would serve as a measurement of their decomposition abilities in writing.

The assignment was adopted from the program for periodical assessment of writing quality in Dutch primary schools conducted by the Inspectorate of Education (Krom et al., 2004), with some small textual adjustments (e.g., *cassette tape* was changed to *DVD*). Further, in this research students had to complete the structured decomposition of the writing task, by filling in a A3 sized decomposition frame tailored to this specific task (see Appendix D, in Dutch). The large paper format allowed for normal handwriting. The goal of the assessment was to deliver a writing schedule that was as complete as possible. The assignment was orally explained by the teacher and all students received a copy of the written assignment. The students had 30 minutes to complete the assignment which virtually all students managed to do. All students work was scanned and student names were replaced by randomly generated codes before evaluation.

Scoring of the decomposition in writing test. The evaluation framework (see Appendix E) included 11 required content elements to be rated as present or absent in the decomposition. The number of elements present in the decomposition represented the decomposition score. To establish interrater reliability, two independent raters evaluated the decompositions of 15 randomly selected students. Intraclass correlation coefficient (ICC) estimates and their 95% confident intervals were calculated using SPSS, based on a single rating, absolute-agreement, 2-way mixed-effects model. This resulted in an unsatisfying estimated ICC of .767 (95% confidence interval = .431 - .916). Therefore, both raters gathered to identify and solve ambiguities in the coding framework. After consultation, the

work of another set of 15 randomly selected students was independently assessed by both raters, resulting in a good estimated ICC of .922 (95% confidence interval = .743 - .967) (Koo & Li, 2016).

Statistical Data Analysis

From 79 students participating in the programming lessons, 70 students completed the programming test. Because none of these students had missed more than one programming lesson, all results were accepted in the analysis. Furthermore, results of the decomposition in writing test of all 128 students that completed the test were analysed. As all conditions in this study had more than 30 participants, adequate statistical power to detect medium to large effect sizes could be assumed (Van Voorhis & Morgan, 2007). All statistical calculations were performed using SPSS 28.

To examine whether including decomposition instruction into the programming lessons lead to better programming results (RQ 1), the scores on the programming test of experimental condition A and B were compared through an independent samples *t*-test. To evaluate the effects of the programming lessons and the inclusion of decomposition instruction in the programming lessons on students' decomposition ability in writing (RQ 2 and 3), a Kruskal-Wallis test was performed to compare the results of the decomposition test between both experimental conditions and the baseline group. The non-parametric Kruskal-Wallis test was chosen because the assumption of normality of the residuals of the data was not met.

In order to find out if the effects of the programming lessons and the decomposition instruction differed between boys and girls, and between students from grade five and grade six (RQ 4), an analyses of covariance (ANCOVA) was performed on the results the programming test. In addition, the non-parametric Quade test, a rank-based analysis of covariances (Quade, 1967), was performed on the results of the decomposition test.

Results

Effect of Decomposition Instruction on Programming Competency (RQ 1)

From experimental group A, the group that received programming lessons augmented with decomposition instruction, 37 out of 40 students completed the programming test (M = 10.81, SD = 6.16). From experimental group B, the group that received programming lessons without decomposition instruction, 33 out of 39 students completed the programming test (M = 12.15, SD = 6.90). Assumptions of normality of the data were checked by inspecting histograms and performing Kolmogorov-Smirnov's test of normality. The histograms deviated from the normal curve, but the tests scores were non-significant for both group A ($D(37)$ = .137, p = .076) and group B ($D(33)$ = .136, p = .128) and thus normality could be assumed. Calculating Levene's statistic revealed that assumptions of homogeneity of variances were met ($F(1,68)$ = 1.225, p = .272). To compare programming scores in both experimental groups, an independent samples t-test was performed. There was no significant difference ($t(68)$ = 0.859, p = .393, d = 0.02) in programming scores between experimental group A and B. Therefore, it can be concluded that adding decomposition instruction to the programming lessons did not impact the programming results.

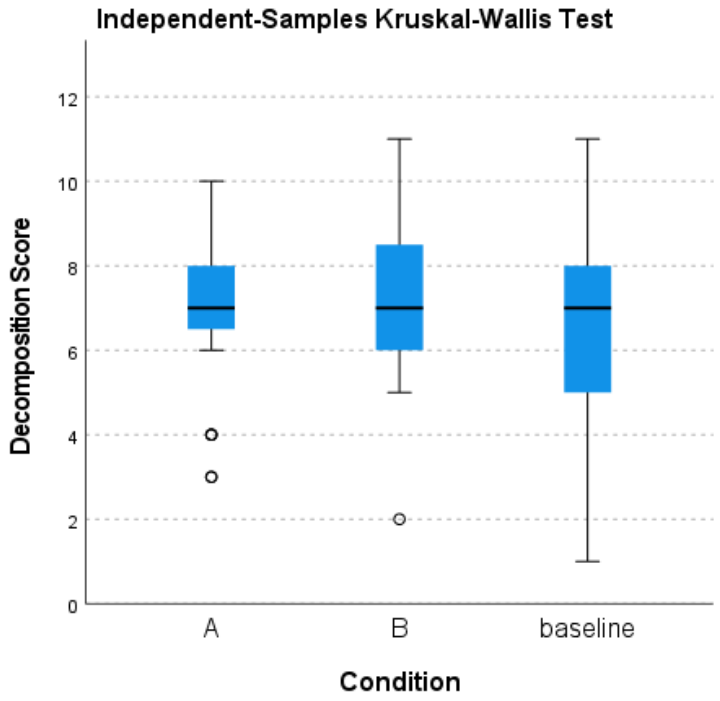
Effect of Programming Instructions on Decomposition Proficiency in Writing (RQ 2 & 3)

A total of 128 students completed the decomposition test, namely all 40 students assigned to experimental group A (M = 7.10, SD = 1,66), 36 out of 39 students assigned to experimental group B (M = 7.42, SD = 1.80), and 52 out of 58 students assigned to the baseline group (M = 6.75, SD = 2.16). Assumptions of normality of the data were checked by inspecting histograms, which showed deviations from the normal curve. Kolmogorov-Smirnov's test of normality revealed significant scores for experimental group A ($D(40)$ = .226, p = <.001) and the baseline group ($D(52)$ = .180, p = <.001). Because the assumptions of normality were violated, the non-parametric Kruskal-Wallis Test for independent samples was performed to compare decomposition scores across all three conditions. The result was not significant ($H(2)$ = 1.340, p = .512, η^2 = - .005). In other words, no overall effect was found of the programming lessons, whether augmented with decomposition instruction or not, on decomposition proficiency. Pairwise comparisons showed that there were also no significant differences between conditions A and B ($H(2)$ = -3.692, p = .659, η^2 = .014), between the baseline group and condition A ($H(2)$ = 5.263, p = .492, η^2 = .026) or between the baseline group and condition B ($H(2)$ = 8.955, p = .257, η^2 = .056). However, inspecting boxplots of the Kruskal-Wallis Test (see Figure 4) revealed a remarkably larger spread of results within the baseline group than in both experimental groups, suggesting a positive effect of the programming lessons, yet not resulting in significantly higher

mean scores. Pairwise comparisons of the spread of results were made by calculating Levene's statistic. These calculations showed a significant difference between condition A and the baseline group (F(1,90) = 4.212, p = .043). No significant differences were found between condition A and B (F(1,74) = 0.459, p = .500), or between condition B and the baseline group (F(1,86) = 1.747, p = .190).

Figure 4

Spread of results of the decomposition test



Gender- and class grade related differences (RQ 4)

Even though no general effect of instructional condition was found on the students' programming scores, an analysis of covariance was conducted to see if the results differed across gender and class grades. There was no significant effect of gender (F(1,66) = 2.972, p = .089, η^2 = .014) nor of class grade (F(1,66) = 1.444, p = .234, η^2 = .007). Because the assumptions of normality were not met (see above), a non-parametric Quade-test was performed to investigate if gender and class grade had influenced the decomposition scores. Because the result was not significant (F(2,125) = .0248, p = .781, η^2 = -0.015), gender and class grade were not found to be relevant co-variables. Pairwise comparisons did also not show impact of gender and class grade on the differences between conditions A and B (t(125) = -.511, p = .610), between the baseline group and condition A (t(125) = .150, p = .881) or between the baseline group and condition B (t(125) = .687, p = .494).

Discussion

Conclusions

This research project investigated how implementing computer programming lessons in primary education could foster computational thinking proficiency and consequently enlarge problem solving skills in computer programming and other domains. Based on the CTPM, this study tested if instruction in problem decomposition facilitated CT in programming. Furthermore, the study tested if transfer of decomposition skills could be observed from the CS domain into the domain of literacy.

Impact of decomposition instruction on programming proficiency (RQ1). Contrary to expectations, students in experimental condition A, who received programming instruction combined with instruction in problem decomposition, did not outperform students in condition B, who received programming instruction only. Condition A performed even slightly worse on the programming test compared to condition B, although the difference was not statistically significant. This could indicate that adding decomposition instruction to the lessons hampered programming progression of the students, possibly due to cognitive overload (De Jong, 2010). Whereas the filler assignment for the other students might have alleviated cognitive demands between the programming assignments and stimulated replenishment of cognitive resources accordingly, the students that received decomposition instruction had to switch from one cognitively demanding task to another, thereby hampering programming progression. Moreover, the time for individual practice of decomposition skills might have been too limited to compensate for the enlarged cognitive demands of engaging with the concept. This phenomenon is described by Kirschner and van Merriënboer (2008) as the *transfer paradox*, because educational design optimized to promote transfer (e.g., by shifting between a variety of tasks) tends to decrease the efficiency of initial learning of the content. In general, it must be concluded that the decomposition instruction has not been effective in achieving better programming results.

Transfer of decomposition skills from CS into literacy (RQ 2 & 3). Computer programming lessons, whether augmented with decomposition instruction or not, did not lead to higher decomposition proficiency in writing, as the test scores were balanced across all groups of students. However, a considerable larger spread of results was observed in the baseline group, indicating that especially lower scoring students could have benefited from the programming lessons. This finding might suggests that 1. lower performing students could have been the first to profit from the structured programming lessons, whereas the effect in higher performing students might only have become apparent after a prolonged intervention, or 2. lower performing students could benefit more in writing from structured programming lessons than their higher performing counterparts in general. Either way, these findings signal that computer programming lessons should not be considered to

merely increase workload, thereby possibly hampering already lower performing students. Instead, these findings indicate that adding computer programming lessons to the primary school curriculum could be beneficial to both higher and lower performing students, as in addition to acquiring programming knowledge, students could benefit from enlarged CT skills in other domains, such as literacy. Larger CT gains in students with lower prior knowledge have been reported by several researchers (Chiazzese et al., 2019; Kwon et al., 2021; Ma et al., 2021) and the results of this study seem to align with those findings. However, because no pre-test was conducted, it remains uncertain if the observed difference in spread of results across conditions on the decomposition in writing test was indeed an effect of the programming lessons.

Altogether, this study did not provide strong evidence of transfer of knowledge between the domains of CS and literacy. The relatively short time span of the intervention could be an explanation for this, as programming is a complex process that takes time to comprehend (Mayer et al. 1986). Furthermore, because the study did not include assessment of decomposition ability in programming, it is not known to what extend the students had mastered this skill in the initial context, before they had to apply decomposition in the literacy context. As is known from literature, a more than shallow understanding of a concept is a prerequisite for transfer (Barnett & Ceci, 2002; McCoy Carver, 1988; Perkins & Salomon, 2012). Another possible explanation is that the decomposition test used to measure the transfer of knowledge into the literacy domain was not sensitive enough to detect existing differences between the groups. The tendency of the spread of results supports this explanation, as the smallest spread is observed in the group that received structured decomposition instruction as part of the programming lessons and the biggest spread is found in the group that did not participate in the programming lessons at all. The group that received programming lessons without structured decomposition instruction came in between, in line with expectations based on the CTPM. According to the CTPM, the programming itself already fostered the transformation of knowledge, by instigating continuous translations between problem spaces and thus enabling transfer between contexts. Adding decomposition instruction presumably enhanced that process and bolstered the effect on the spread that was observed.

Effects of gender and class grade (RQ 4). This study did not observe differences in results between fifth- or sixth grade students or between boys and girls. More specific, the impact of adding decomposition instruction to the programming lessons was found not to differ significantly across grades or gender and the same goes for the impact of programming lessons on decomposition scores, whether decomposition instruction had been included in the programming lessons or not. Regarding gender effects, previous research has shown mixed results. Both higher and lower learning gains for girls have been reported, as well as the absence of gender effects (Sun et al., 2022). This research

shows that, at least under contextual circumstances similar to those of the current research, it is possible to design programming education that achieves equal results for boys and girls.

Regarding differences between students from grade five and six, the anticipated higher benefits of programming lessons and decomposition instruction of grade six students, based on their larger prior knowledge, was not shown by this study's results. This indicates that the differences in prior knowledge between grade five and grade six students had no relevance to the programming lessons. Therefore, it can be assumed that for initial programming education, students of grade five and six can be taught in combined classes without hampering the learning outcome of students from the higher or the lower grade. However, this study did only compare outcome scores between conditions, thus no conclusions can be reached about possible differences in learning gains between various subgroups of students, such as boys and girls or students from grade five and grade six. In other words, it remains possible that one of the subgroups had lower prior knowledge at the start of the interventions but was able to catch up as a result of the interventions, thus obscuring possible effects of class grade and gender.

Scientific and practical relevance of the CTPM. This study proposed the CTPM to represent CT as a creative, knowledge transforming, problem solving process. Some of this study's findings have corroborated the model, in particular the differences across the conditions in the spread of results on the decomposition test, as discussed above. Adopting the CTPM holds several benefits for research and educational practices. By defining CT as overarching process, regulating a variety of sub-processes in the development of a computational artefact, there is no need to identify exhaustively which computational processes, practices and perspectives constitute CT. Depending on the context, the relevance of those elements can differ, whereas the overarching process remains pertinent. Thus, the CTPM offers a concise description of the core qualities of CT, as has been scholarly requested (Voogt et al., 2015). Furthermore, the CTPM incorporates an explication of near- and far-transfer processes, as equivalents of the knowledge transforming thinking processes resulting from the translation of problems between the computational problem space and the content/context problem space.

Besides aiding scientific understanding, the model could inform educational design aiming for transfer. Moreover, based on the similarity of problem solving processes between computer programming and writing (i.e.: text composition) as indicated by the CTPM, educational science and programming education could draw from the extensive body of literature on writing education to comprehend the challenges of effective programming education.

Limitations and Future Directions

As already mentioned upon, a major limitation of this study is that no pre-tests were included. This is restricting the generalisability of the finding that decomposition scores of programming students

were more closely gathered around the mean than the scores of students that did not participate in computer programming lessons. Furthermore, because of the lack of pre-testing, it cannot be known for sure that both experimental groups encompassed equal variances of prior knowledge and therefore, it remains uncertain if the statistically similar test scores on the programming test and the decomposition test indeed reflected equal learning gains in both groups. Therefore, including pre-tests should be considered in subsequent research. In addition, a more elaborate measurement of decomposition skills in the literacy domain would make a study like the one presented here more robust. In the current study, only one writing task was included, solely targeting one specific writing genre accordingly (i.e., a persuasive letter). This is not sufficient to generalise findings to students' writing in general and therefore, it is advised to include multiple measurements using a variety of text genres (Bouwer et al., 2018).

To advance the understanding and implementation of CT is primary education, it is recommended that future research adopts a longitudinal approach, to create the essential opportunity for gradual development of CT. This approach would stimulate the transformation of knowledge in learners, thus enlarging opportunities for transfer to other domains. Moreover, a longitudinal approach, with multiple measurements during the intervention, would yield a more detailed view of the development of CT proficiency within various groups of students over time, aiding the formulation of age-appropriate learning objectives, and possibly clarifying the differences between lower and higher performing students observed in the current study.

Including generic measurements of CT (for example the CTt), would give a broader view on the development of CT as a multi-facetted process and it would assist the research community in comparing interventions across studies. Moreover, to further test the CTPM as a valuable model to guide CT research and the implementation of programming education in primary schools, it would be worthwhile to design and assess instructional interventions that target other skills than decomposition that could facilitate the CT process, such as generalisation or abstraction. In addition, assessment of those skills in both the initial learning context (i.e.: programming) as well as the transfer context (i.e.: literacy) would enable examining the correlation of those skills across contexts, which in turn could provide further validation of the CTPM.

References

Agirdag, O., Biesta, G., Bosker, R., Kuiper, R., Nieveen, N., Raijmakers, . . . Boogaard, M. (2021). *Samenhang in het curriculum. Verdiepende studie Wetenschappelijke curriculumcommissie.* [Coherence in the curriculum. In-depth study of the scientific committee for the curriculum.] Retrieved from: https://www. curriculumcommissie.nl/documenten

Araujo, A. L. S. O., Andrade, W. L., Guerrero, D. D. S., & Melo, M. R. A. (2019). How many abilities can we measure in computational thinking? A study on Bebras challenge. *Proceedings of the 50th ACM technical symposium on computer science education* (pp. 545-551). doi:10.1145/ 3287324.3287405

Bagge, P. (2015). *Code-it primary programming; How to teach primary programming using Scratch.* Buckingham, United Kingdom: University of Buckingham Press.

Balanskat, A., & Engelhardt, K. (2015). *Computing our future. Computer programming and coding. Priorities, school curricula and initiatives across Europe*. Retrieved from European Schoolnet website http://www.eun.org/documents/411753/817341/Coding+initiative +report +Oct2014/2f9b35e7-c1f0-46e2-bf72-6315ccbaa754

Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological bulletin*, *128*(4), 612. doi:10.1037/0033-2909.128.4.612

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?. *ACM Inroads*, *2*(1), 48-54. doi:10.1145/1929887.1929905

Bers, M. U., Govind, M., & Relkin, E. (2022). Coding as another language: Computational thinking, robotics and literacy in first and second grade. In *Computational thinking in prek-5: empirical evidence for integration and future directions* (pp. 30-38). doi:10.1145/3507951.3519285

Boom, K. D., Bower, M., Siemon, J., & Arguel, A. (2022). Relationships between computational thinking and the quality of computer programs. *Education and information technologies*, *27*(6), 8289-8310. doi:10.1145/3197091.3197104

Bouwer, R., Koster, M., & van den Bergh, H. (2018). Effects of a strategy-focused instructional program on the writing quality of upper elementary students in the Netherlands. *Journal of Educational Psychology*, *110*(1), 58. doi:10.1037/edu0000206

Bråting, K. & Kilhamn, C. (2022). The Integration of Programming in Swedish school mathematics: Investigating elementary mathematics textbooks. *Scandinavian Journal of Educational Research, 66*(4), 594-609. doi:10.1080/00313831.2021.1897879

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 annual meeting of the American Educational Research Association.* Retrieved from http://scratched.gse.harvard.edu/ ct/files/AERA2012.pdf

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & education, 109*, 162-175. doi:10.1016/j.compedu.2017.03.001

Chiazzese, G., Arrigo, M., Chifari, A., Lonati, V. & Tosto, C. (2019). Educational robotics in primary school: Measuring the development of computational thinking skills with the Bebras tasks. *Informatics*, *6*(4), 43-54. doi:10.3390/informatics6040043

Corradini, I., Lodi, M., & Nardelli, E. (2017). Conceptions and misconceptions about computational thinking among Italian primary school teachers. *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 136-144). doi:10.1145/ 3105726.3106194

Dagiene, V., & Dolgopolovas, V. (2022). Short Tasks for Scaffolding Computational Thinking by the Global Bebras Challenge. *Mathematics*, *10*(17), 3194. doi:10.3390/math10173194

Dagiene, V., & Stupuriene, G. (2016). Bebras: A sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in education*, *15*(1), 25-44. doi:10.15388/infedu.2016.02

De Jong, T. (2010). Cognitive load theory, educational research, and instructional design: Some food for thought. *Instructional science*, *38*(2), 105-134. doi:10.1007/s11251-009-9110-0

Engle, R. A., Lam, D. P., Meyer, X. S., & Nix, S. E. (2012). How does expansive framing promote transfer? Several proposed explanations and a research agenda for investigating them. *Educational Psychologist*, *47*(3), 215-231. doi:10.1080/00461520.2012.695678

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with Scratch in primary schools: A systematic review. *Computer Applications in Engineering Education*, *29*(1), 12-28. doi:10.1002/cae.22255

Falloon, G. (2016). An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. on the iPad. *Journal of Computer Assisted Learning*, *32*(6), 576-593. doi:10.1111/jcal.12155

Graham, S., & Harris, K. R. (2016). A path to better writing: Evidence-based practices in the classroom. *The Reading Teacher*, *69*(4), 359-365. doi:10.1002/trtr.1432

Graham, S., McKeown, D., Kiuhara, S., & Harris, K. R. (2012). A meta-analysis of writing instruction for students in the elementary grades. *Journal of educational psychology*, *104*(4), 879.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38-43. doi:10.3102/0013189X12463051

Grover, S., & Pea, R. (2018). Computational thinking: A competency whose time has come. *Computer science education: Perspectives on teaching and learning in school*, *19*(1), 19-38. doi:10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer science education*, *25*(2), 199-237. doi:10.1080/08993408.2015.1033142

Hassenfeld, Z. R., Govind, M., de Ruiter, L. E., & Bers, M. U. (2020). If you can program you can write: Learning introductory programming across literacy levels. *Journal of Information Technology Education*, *19*. doi:10.28945/4509

Hayes, J. R. (2012). Modeling and remodeling writing. *Written Communication 29*(3), 369-388. doi:10.1177/0741088312451260

Jacob, S. R., & Warschauer, M. (2018). Computational thinking and literacy. *Journal of Computer Science Integration*, *1*(1). doi:10.26716/jcsi.2018.01.1.1

Kang, E. J., Donovan, C., & McCarthy, M. J. (2018). Exploring elementary teachers' pedagogical content knowledge and confidence in implementing the NGSS science and engineering practices. *Journal of Science Teacher Education*, *29*(1), 9-29. doi:10.1080/1046560X.2017.1415616

Kirschner, P., & Van Merrienboer, J. J. (2008). Ten steps to complex learning: A new approach to instruction and instructional design. In T. L. Good (Ed.), *21st century education: A reference handbook* (pp. 244-253). Tucson, AZ: Sage. Retrieved from https://research.ou.nl/files/941352/Ten%20Steps%20to%20Complex%20Learning%20-%20Sage%2021st%20Century.pdf

Koo, T. K., & Li, M. Y. (2016). A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of chiropractic medicine*, *15*(2), 155-163. doi:10.1016/j.jcm.2016.02.012

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in human behavior*, *72*, 558-569. doi:10.1016/j.chb.2017.01.005

Krom, R., Van de Gein, J., van der Hoeven, J., van der Schoot, F., Verhelst, N., Veldhuijzen, N., & Hemker, B. (2004). *Balans van het schrijfonderwijs op de basisschool. Uitkomsten van de peilingen in 1999* [The state of writing education in primary schools. Results of the 1999 investigation]. Retrieved from CITO website https://cito.nl/media/04tp54hf/balans-van-het-schrijfonderwijs-op-de-basisschool-ppon-reeks-nummer-28.pdf

Kuhlemeier, H., van Til, A., Hemker, B., de Klijn, W., & Feenstra, H. (2013). Balans van de schrijfvaardigheid in het basis- en special basisonderwijs 2. [The state of writing education in primary schools and primary schools for special educational needs 2] Retrieved from CITO

website https://cito.nl/kennisbank-stichting-cito/balans-van-de-schrijfvaardigheid-in-het-basis-en-speciaal-basisonderwijs-2-ppon-reeks-nummer-53/

Kumar, A. N., Becker, B. A., Pias, M., Oudshoorn, M., Jalote, P., Servin, C., . . . Anderson, M. D. (2023). A Combined Knowledge and Competency (CKC) Model for Computer Science Curricula. *ACM Inroads*, *14*(3), 22-29. doi:10.1145/3605215

Kwon, K., Ottenbreit-Leftwich, A. T., Brush, T. A., Jeon, M., & Yan, G. (2021). Integration of problem-based learning in elementary computer science education: effects on computational thinking and attitudes. *Educational Technology Research and Development*, *69*, 2761-2787. doi:10.1007/s11423-021-10034-3

Lee, H. Y., Lin, C. J., Wang, W. S., Chang, W. C., & Huang, Y. M. (2023). Precision education via timely intervention in K-12 computer programming course to enhance programming skill and affective-domain learning objectives. *International Journal of STEM Education*, *10*(1), 52. doi:10.1186/s40594-023-00444-5

Liao, Y.k. (2000). A Meta-analysis of Computer Programming on Cognitive Outcomes: An Updated Synthesis. In J. Bourdeau & R. Heller (Eds.), *Proceedings of ED-MEDIA 2000--World Conference on Educational Multimedia, Hypermedia & Telecommunications* (pp. 598-604). Montreal, Canada: Association for the Advancement of Computing in Education (AACE). Retrieved from https://www.learntechlib.org/primary/p/16132/.

Littlefield, J., Delclos, V. R., Lever, S. L., Clayton, K. N., Bransford, J. D., & Franks, J. J. (1988). Learning LOGO: Method of teaching, transfer of general skills, and attitudes toward school and computers. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming; Multiple Research Perspectives.* (pp. 111-136). Hillsdale, NJ: Lawrence Erlbaum.

Lodi, M., & Martini, S. (2021). Computational thinking, between Papert and Wing. *Science & Education*, *30*(4), 883-908. doi:10.1007/s11191-021-00202-5

Ma, H., Zhao, M., Wang, H., Wan, X., Cavanaugh, T. W., & Liu, J. (2021). Promoting pupils' computational thinking skills and self-efficacy: A problem-solving instructional approach. *Educational Technology Research and Development*, *69*(3), 1599-1616. doi: 10.1007/ s11423-021-10016-5

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: what's the connection?. *Communications of the ACM*, *29*(7), 605-610. doi:10.1145/6138.6142

McCoy Carver, S. (1988) Learning and transfer of debugging skills: Applying task analysis to curriculum design and assessment. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming; Multiple Research Perspectives.* (pp. 259-298). Hillsdale, NJ: Lawrence Erlbaum.

Nieveen, N., & Kuiper, W. (2012). Balancing curriculum freedom and regulation in the Netherlands. *European Educational Research Journal*, *11*(3), 357-368. doi:10.2304/eerj.2012.11.3.357

Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2020). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry*, *11*(1), 1-17. doi: 10.1080/20004508.2019.1627844

Nusche, D., & Minea-Pic, A. (2020). *ICT resources in school education: What do we know from OECD work.* (OECD Education Working Paper EDU/EDPC/SR/RD 2020(2)). Retrieved from OECD website https://one.oecd.org/document/edu/edpc/sr/rd(2020)2/en/pdf

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

Parsazadeh, N., Cheng, P. Y., Wu, T. T., & Huang, Y. M. (2021). Integrating computational thinking concept into digital storytelling to improve learners' motivation and performance. *Journal of Educational Computing Research*, *59*(3), 470-495. doi:10.1177/0735633120967315

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New ideas in psychology*, *2*(2), 137-168. Retrieved from https://web.stanford.edu/~roypea/RoyPDF%20folder/A44_Kurland_etal_*REDO.pdf

Perkins, D. N., & Salomon, G. (2012). Knowledge to go: A motivational and dispositional view of transfer. *Educational Psychologist*, *47*(3), 248-258. doi:10.1080/00461520.2012.693354

Perkins, D. N., Schwarz, S., Simmons, R. (1988). Instructional Strategies for the Problems of Novice Programmers. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming; Multiple Research Perspectives.* (pp. 153-178). Hillsdale, NJ: Lawrence Erlbaum.

Platform Onderwijs2032, Advisory board on the revision of the Dutch educational curriculum. (2016). *Ons onderwijs2032 eindadvies* [Our education in 2032; Final recommendations]. Retrieved from http://onsonderwijs2032.nl/wp-content/uploads/2016/01/Ons-Onderwijs2032-Eindadvies-januari-2016.pdf

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365-376. doi:10.1016/j.compedu.2018.10.005

Quade, D. (1967). Rank analysis of covariance. *Journal of the American Statistical Association*, *62*(320), 1187-1200. doi:10.2307/2283769

Rodríguez-Martínez, J. A., González-Calero, J. A., & Sáez-López, J. M. (2020). Computational thinking and mathematics using Scratch: an experiment with sixth-grade students. *Interactive Learning Environments*, *28*(3), 316-327. doi:10.1080/10494820.2019.1612448

Román-González, M. (2015). Computational thinking test: Design guidelines and content validation. In *EDULEARN15 Proceedings* (pp. 2436-2444). IATED. Retrieved from https://www.researchgate.net/profile/Marcos-Roman-Gonzalez/publication/290391277_COMPUTATIONAL_THINKING_TEST_DESIGN_GUIDELINES_AND_CONTENT_VALIDATION/links/56966c7c08aea2d7437469c2/COMPUTATIONAL-THINKING-TEST-DESIGN-GUIDELINES-AND-CONTENT-VALIDATION.pdf

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in human behavior*, *72*, 678-691. doi:10.1016/j.chb.2016.08.047

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction*, *18*, 47-58. doi:10.1016/j.ijcci.2018.06.004

Salomon, G., & Perkins, D. N. (1989). Rocky Roads to Transfer: Rethinking Mechanism of a Neglected Phenomenon. *Educational Psychologist, 24*(2), 113. doi:10.1207/s15326985ep2402_1

Scardamalia, M., & Bereiter, C. (1991). Literate expertise. *Toward a general theory of expertise: Prospects and limits*, *172*, 194. Retrieved from https://www.academia.edu/download/36339732/Scardamalia___Bereiter_1991.pdf

Scardamalia, M., & Bereiter, C. (2010). A brief history of knowledge building. *Canadian Journal of Learning and Technology/La revue canadienne de l'apprentissage et de la technologie*, *36*(1). Retrieved from https://www.learntechlib.org/d/43123

Scardamalia, M., Bereiter, C., & Steinbach, R. (1984). Teachability of reflective processes in written composition. *Cognitive science*, *8*(2), 173-190. doi:10.1016/S0364-0213(84)80016-6

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, *111*(5), 764. doi: 10.1037/edu0000314

Selby, C., & Woollard, J. (2014). Refining an understanding of computational thinking. *University of Southampton Institutional Repository.* Retrieved from https://eprints.soton.ac.uk/372410/1/372410UnderstdCT.pdf

Seisdedos, N. (2002). *RP-30: Resolución de problemas*; *Manual.* Barcelona, Spain: TEA Ediciones.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational research review*, *22*, 142-158. doi:10.1016/j.edurev.2017.09.003

Sun, L., Hu, L., & Zhou, D. (2022). Programming attitudes predict computational thinking: Analysis of differences in gender and programming experience. *Computers & Education*, *181*, 104457. doi:10.1016/j.compendy.2022.104457

Swanborn, M. (Ed.). (2023). *De staat van het onderwijs 2023.* [The state of education 2023]. Retrieved from https://www.onderwijsinspectie.nl/binaries/onderwijsinspectie/documenten/rapporten/2023/05/10/rapport-de-staat-van-het-onderwijs-2023/Staat+van+het+Onderwijs+2023.pdf

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798. doi: 10.1016/j.compedu. 2019.103798

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. *Proceedings of the 16th Koli Calling international conference on computing education research*, 120-129. doi:10.1145/2999541.2999542

Techniekpact, Task Force on Technology in Education and Economy. (2013). *Nationaal Techniekpact 2020* [National Technology Covenant 2020]. Retrieved from http://techniekpact.nl/cdi/files/f1441a07a7dab41382fd20095b16c618ad14773c.pdf

Tsarava, K., Moeller, K., Román-González, M., Golle, J., Leifheit, L., Butz, M. V., & Ninaus, M. (2022). A cognitive definition of computational thinking in primary education. *Computers & Education*, *179*, 104425. doi:10.1016/j.compedu.2021.104425

Tsarava, K., Leifheit, L., Ninaus, M., Román-González, M., Butz, M. V., Golle, J., ... & Moeller, K. (2019). Cognitive correlates of computational thinking: Evaluation of a blended unplugged/plugged-in course. *Proceedings of the 14th workshop in primary and secondary computing education* (pp. 1-9). doi:10.1145/3361721.3361729

Van der Linde, D., Voogt, J., & van Aar, N. (2021). Computational Thinking skills of young children working on a programming task. *Journal of Computers in Mathematics and Science Teaching*, *40*(4), 357-376. Retrieved from https://www.learntechlib.org/primary/p/215691/

Van Voorhis, C. W., & Morgan, B. L. (2007). Understanding power and rules of thumb for determining sample sizes. *Tutorials in quantitative methods for psychology*, *3*(2), 43-50. Retrieved from https://pdfs.semanticscholar.org/8199/1f3c923b3927164f6b7c86c78e508980f61f.pdf

Visser, J. (2016). Onderwijs 2032 moest dé vernieuwing worden. Maar leraren zien er weinig in. [Education 2032 had to become the big innovation, but teachers don't think much of it.] *De correspondent.* https://decorrespondent.nl/5688/onderwijs2032-moest-de-vernieuwing-worden-maar-leraren-zien-er-weinig-in/eb2c77a1-d90a-03e1-1317-b470e492f3b4

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, *20*(4), 715-728. doi:10.1007/s10639-015-9412-6

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of science education and technology*, *25*, 127-147. doi: 10.1007/s10956-015-9581-5

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. doi:10.1145/1118178.1118215

Wing, J. M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology*, *25*(2), 7-14. doi:10.17471/2499-4324/922

Ye, J., Lai, X., & Wong, G. K. W. (2022). The transfer effects of computational thinking: A systematic review with meta-analysis and qualitative synthesis. *Journal of Computer Assisted Learning*, *38*(6), 1620-1638. doi:10.1111/jcal.12723

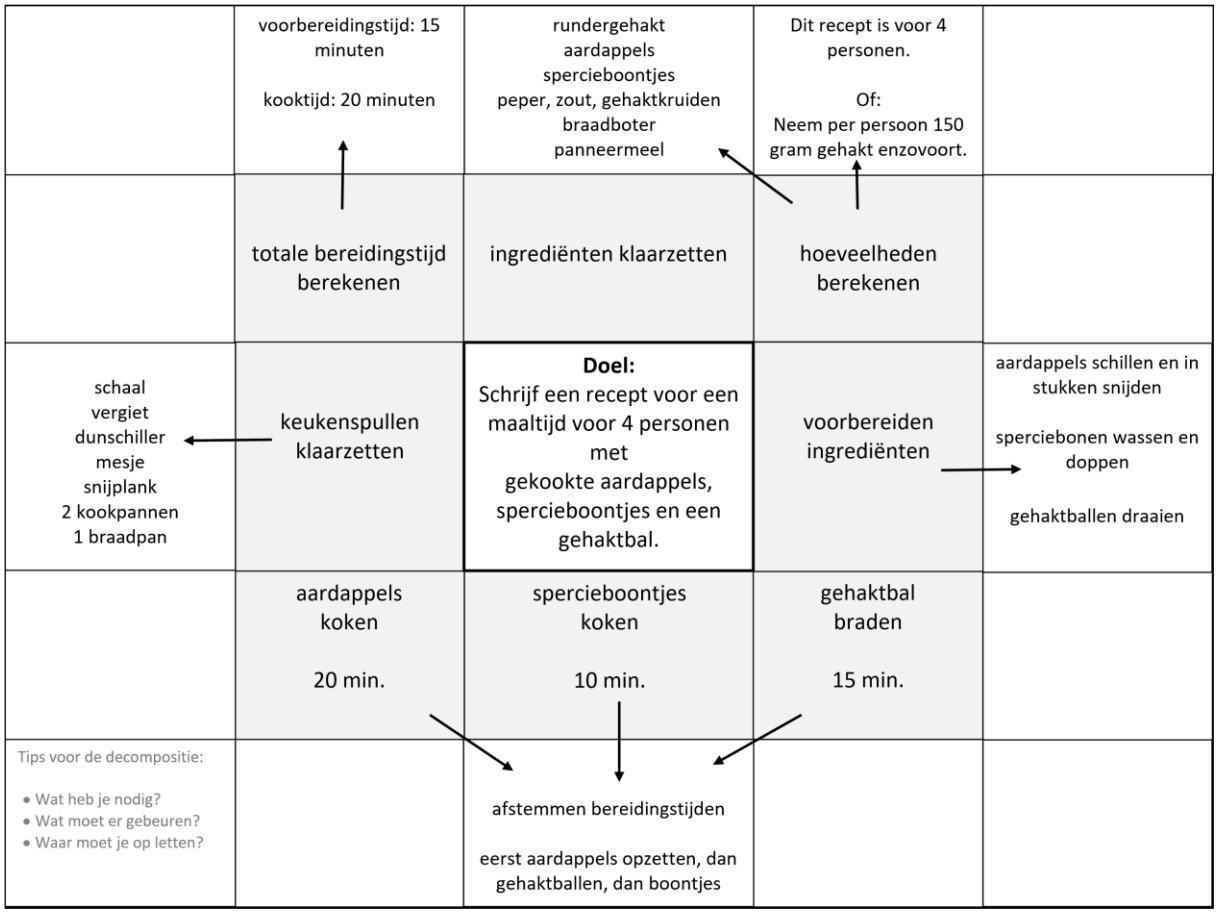
Yeni, S., Nijenhuis-Voogt, J., Hermans, F., & Barendsen, E. (2022). An Integration of Computational Thinking and Language Arts: The Contribution of Digital Storytelling to Students' Learning. *Proceedings of the 17th Workshop in Primary and Secondary Computing Education* (pp. 1-10). doi:10.1145/3556787.3556858

Zapata-Cáceres, M., Martín-Barroso, E., & Román-González, M. (2020, April). Computational thinking test for beginners: Design and content validation. In *2020 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1905-1914). IEEE. doi:10.1109/EDUCON45650.2020.9125368

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607. doi:10.1016/j.compedu.2019.103607

Appendix A

Decomposition frame: real world example



Appendix B

Decomposition frame: programming example

		Doel: In dit spel verstoppen 3 beren zich steeds op verschillende plekken in het bos. Iedere keer als je een beer aanklikt, krijg je een punt, tot de tijd op is.		
Tips voor de decompositie: • welke sprites heb je nodig? (8) • welke uiterlijkheden (hoeveel) heeft iedere sprite nodig? • welke achtergronden (hoeveel) voor het speelveld heb je nodig? • welke scripts (instructies) moet je schrijven? • waar is herhaling/hergebruik van code mogelijk? (rood) • waar moet je afstemmen tussen sprites? (buitenste vakken) • welke programmeerideeën zijn handig? (buitenste vakken)				

Appendix C

Code book programming test

Programmeertest Codeboek

Beschrijving test

De leerlingen werken drie keer aan een onvolledig programma, waarbij ze iedere keer drie programmeerproblemen voorgeschoteld krijgen (completion problems). De problemen zijn zo ontworpen dat de leerlingen ook gedeeltelijke oplossingen zouden kunnen vinden. De drie opdrachten binnen één programma zijn betrekkelijk onafhankelijk vormgegeven, zodat eerder gemaakte fouten de oplosbaarheid van het volgende probleem zo min mogelijk beïnvloeden. Voor ieder foutloos/optimaal opgelost probleem worden 3 punten toegekend, suboptimale oplossingen kunnen 1 of 2 punten opleveren. Per programma kan de leerling dus een score halen van 0-9 punten en over de hele test een maximale totaalscore van 27 punten.

Programma 1: Tikkertje met Gobo & Nano

opdracht 1: afstemmen

De spraakscripts van Gobo en Nano moeten op elkaar worden afgestemd. Daarvoor moet de leerling aan beide scripts twee wachtblokken toevoegen én de wachttijden juist instellen (argument aanpassen).

punten:	**resultaat:**	**criteria (één van deze):**
NB:	***Ev ntuele gedupliceerde groene vlag scripts moeten buiten beschouwing worden gelaten bij het beoordelen van deze opdracht (=> waarschijnlijk resultaat van opdracht 2).***	
3	De tekst van beide sprites sluit**, in de juiste volgorde**, **naadloos** op elkaar aan, **zonder overlap**. *(wachttijd komt overeen met spreektijd)*	• in beide scripts zijn 2 wachtblokken op de juiste plek ingevoerd én alle wachtblokken zijn op de juiste tijd (**zie illustratie**), er zijn daarbij géén blokken verwijderd
2	Beide sprites wisselen elkaar **in de juiste volgorde** af bij het spreken, maar met kleine overlap of pauze. **Of:** De tekst van beide sprites sluit naadloos, **in de juiste volgorde** én zonder overlap aan (wachttijd = spreektijd) maar niet geheel conform opdracht (bijv: Nano begint niet 1 sec. na verschijning te spreken).	• alle wachtblokken op de goede plaats ingevoegd én het eerste wachtblok van Gobo is ingesteld op 4 seconde (daarbij eerste wachtblok Nano onveranderd, dus 1 sec.)* • alle wachtblokken op de goede plaats ingevoegd, maar maximaal 2 tijden verkeerd ingesteld. • optimale code (zie illustratie) voor één van de sprites, 1 fout blok (plaats en/of tijd) voor tweede sprite • scripts van beide sprites zijn optimaal afgestemd (wachttijd = spreektijd andere sprite) maar daarbij zijn extra blokken veranderd, toegevoegd, verwijderd of verplaatst.

1	Beide sprites wisselen elkaar af bij het spreken, maar niet in de juiste volgorde en/of met overlap en/of pauze(s).	• aan beide scripts zijn op de juiste plek 2 wachtblokken toegevoegd (tussen paarse blokken), waarbij meer dan 2 wachttijden niet zijn aangepast (1 seconde) of foutief zijn aangepast** • aan beide scripts zijn 2 wachtblokken toegevoegd waarvan maximaal 1 op de verkeerde plek, waarbij de wachttijden zijn aangepast aan de spreektijden (3 sec bij onaangepaste spreekblokken)
0	Het groene vlag script functioneert niet of nauwelijks.	• aan geen van bovenstaande criteria is voldaan

* Deze code weerspiegelt het inzicht dat Gobo ná wachttijd en spreektijd van Nano (1 + 3 seconde) moet gaan spreken. Ook als het afstemmen verderop minder goed gelukt is, moet hier 2 punten voor worden toegekend.

** Hoewel een wachttijd van bijv. 5 seconden voor alle blokken tot een redelijk vloeiende dialoog leidt, weerspiegelt deze oplossing eerder een trial-and-error strategie dan het inzicht spreektijd=wachttijd en levert daarom slecht 1 punt op.

Illustraties bij programma 1

optimale oplossing opdracht 1 (3 punten):

↑ tekstscript Gobo
→ tekstscript Nano

optimale oplossing opdracht 2 en 3 (3 punten):

NB: bij het beoordelen van opgave 2 blijven de instellingen van de blokken buiten beschouwing.

← spatiebalkscript Gobo
↑ sprite-kliksript Gobo
→ game-overscript Gobo

opdracht 2: dupliceren/hergebruiken van code

De scripts die Nano aan het spel laten deelnemen zijn gegeven. De leerling moet Gobo aan het spel laten deelnemen door de juiste scripts (3) te kopiëren en aan het scriptveld van Gobo toe te voegen.

punten:	**resultaat:**	**criteria (één van deze):**
NB1: ***Bij het beoordelen van deze opdracht alléén op de aanwezigheid van de juiste scripts let en. De instellingen (argumenten) worden bij opdracht 3 beoordeeld.*** ***NB2:*** ***Let bij beoordelen primair op code van Gobo, hier moet de leerling (3) scripts hebben toegevoegd. De gegeven code van Nano hoort onveranderd te zijn.***		
3	Nano en Gobo nemen beiden deel aan het spel. Ze verschijnen/verdwijnen op willekeurige posities gedurende de speeltijd en verdwijnen als de speeltijd voorbij is. Als ze worden aangeklikt, verandert de score.	• de juiste drie scripts (spatiebalk, sprite-klik en game-over) zijn van het scriptveld van Nano **gekopieerd** naar het scriptveld van Gobo (dus op beide scriptvelden aanwezig).
2	Nano en Gobo nemen beiden deel aan het spel, maar met beperkte functionaliteit.	• er zijn **van de 3 bovengenoemde scripts minimaal 2** aanwezig (groene vlag script telt niet mee) waaronder het spatiebalkscript en daarbij is er géén redundante code toegevoegd (losse blokken tellen niet mee) en zijn de gekopieerde scripts óók nog op het scriptveld van Nano aanwezig.*
1	Gobo neemt deel aan het spel, maar het programma functioneert niet optimaal.	• van de 3 bovengenoemde scripts is er minimaal 1 aanwezig op het scriptveld van Gobo* (groene vlag script telt niet mee)
0	Gobo neemt niet deel aan het spel	• op het scriptveld van Gobo is geen van de 3 juiste scripts aanwezig
* De aanwezigheid van het spatiebalkscript is voorwaarde voor het functioneren van Gobo zoals beschreven. Een oplossing zonder spatiebalkscript weerspiegelt dit fundamentele inzicht niet, en kan daarom geen 2 punten opleveren. Zonder dit script is hij echter wél aanwezig op het scherm (vanwege het groene vlagscript) en kan dus punten scoren en/of verdwijnen bij game-over, in welk geval 1 punt wordt toegekend.		

opdracht 3: aanpassen scripts

De leerling moet door het aanpassen van de juiste scripts, maar zonder het gebruik van extra blokken/code, Gobo ~~twee keer zo vaak~~ laten verschijnen binnen de speeltijd als Nano en de puntentelling kloppend maken met de uitleg uit opdracht 1.

punten:	resultaat:	criteria (één van deze):
NB1: De bedoeling is dat de leerlingen Gobo vaker laten verschijnen dan Nano. In de opdracht staat "2x zo vaak," maar dat levert een onbedoeld wiskundig probleem op (de duur van de complete verdwijn/verschijncyclus moet worden gehalveerd). Alle oplossingen waarbij Gobo vaker verschijnt dan Nano moeten daarom worden goedgekeurd. **NB2: Beoordeel primair de code van Gobo. De code van Nano is volledig gegeven en hoeft niet te worden aangepast (tenzij door leerling gekozen oplossing daarom vraagt).** **NB3: Indien een script geheel ontbreekt, geldt het voor de beoordeling als "niet ingesteld/aangepast".**		
3	Het spelletje functioneert precies zoals in het groene vlagscript beschreven. *oftewel: verdwijnen/verschijnen én score kloppen*	• De wachtblokken in de spatiebalkscripts van Nano en Gobo zijn dusdanig ingesteld dat Gobo vaker verschijnt dan Nano én van het sprite-klik script van Gobo is blok *verander score* ingesteld op 1 én er is geen extra code veranderd, verwijderd of toegevoegd.
2	Het spelletje functioneert, maar niet precies zoals in het groene vlagscript beschreven. *oftewel:* *verdwijnen/verschijnen klopt, maar score klopt niet.* *óf:* *verdwijnen/verschijnen klopt niet helemaal (aanpassing aanwezig), maar score klopt wél.*	• De wachtblokken in de spatiebalkscripts van Nano en Gobo zijn dusdanig ingesteld dat Gobo vaker verschijnt dan Nano, maar van het sprite-klik script van Gobo is blok *verander score* niet ingesteld op 1 • Er zijn wachtblokken aangepast in het script van Gobo en/of Nano, maar niet dusdanig dat Gobo vaker verschijnt dan Nano, daarbij is in het sprite-klik script van Gobo het blok *verander score* wél ingesteld op 1.

1	Het spelletje functioneert, maar niet zoals in het groene vlagscript beschreven. *oftewel:* *alleen de score klopt* *óf:* *Nano verschijnt vaker dan Gobo (= verkeerd om) en de score klopt niet.*	• Gobo's blok *verander score* is ingesteld op 1, maar er zijn geen wachtblokken aangepast (zodanig dat de spatiebalkscripts van Gobo en Nano van elkaar verschillen) • Gobo's blok *verander score* is niet aangepast, maar in het spatiebalkscript van Gobo en/of Nano zijn wachtblokken aangepast (zodanig dat de beide
		spatiebalkscripts van elkaar verschillen)
0	Het spelletje functioneert niet zoals in het groene vlagscript beschreven.	• aan geen van bovenstaande criteria is voldaan

Programma 2: Tekenpatroon

opdracht 1: teken vierkant
De leerling schrijft het script voor het tekenen van een vierkant, waarbij gekozen moet worden uit een selectie van blokken en het argument van het herhaalblok moet worden aangepast.

punten:	resultaat:	criteria (één van deze):
NB1: O dat bij opdracht 3 code gecombineerd moet worden, moet bij het beoordelen van de e opdracht er alleen op worden gelet of de gevraagde code ergens op het scriptveld aa wezig is, hetzij los, hetzij als onderdeel van een groter script.		
3	Als de (geïsoleerde) code wordt aangeklikt, tekent de sprite een vierkant met zijden 50, zonder overlap of met precies een half vierkant overlap.*	• herhaallus is ingesteld op 4 of 6, in de lus 3 blokken: *pen neer* en dan (volgorde vrij) *neem 50 stappen* en *draai rechts 90 graden* • blok *pen neer*, dan herhaallus ingesteld op 4 of 6, in de lus 2 blokken (volgorde vrij): *neem 5 stappen* en *draai rechts 90 graden* • als hierboven, waarbij onderaan blok *pen op* is toegevoegd (binnen of buiten de lus)
2	Als de (geïsoleerde) code wordt aangeklikt, tekent de sprite een vierkant met zijden 50, zonder overlap of met precies een half vierkant overlap.*	• de code is langer dan nodig, maar het juiste herhaalblok is gekozen en voor de gebruikte code passend ingesteld.
1	Als de (geïsoleerde) code wordt aangeklikt, tekent de sprite een vierkant (grootte onbelangrijk), meerdere vierkanten of blijft vierkanten tekenen.	• geen herhaalblok gebruikt, maar herhaling van code • verkeerde herhaalblok gebruikt • juiste blokken, maar te veel herhalingen ingesteld • teveel blokken binnen de herhaallus
0	Er is geen (isoleerbare) code aanwezig die de sprite een vierkant laat tekenen	• aan geen van bovenstaande criteria is voldaan
* *Half vierkant overlap toegestaan omdat dit deel uit kan maken van de door de leerling gekozen oplossing voor opdracht 2*		

opdacht 2: een rij vierkantjes
De leerling moet het script maken om een rij van 10 vierkantjes te krijgen. Daarvoor moeten drie vastgestelde blokken worden gebruikt, het argument van het herhaalblok juist worden ingesteld, en de verkregen code gecombineerd met het script van opdracht 1.

punten:	resultaat:	criteria (één van deze):
NB1: ***O dat bij opdracht 3 code gecombineerd moet worden, moet bij het beoordelen van de e opdracht er alleen op worden gelet of de gevraagde code ergens op het scriptveld aa nwezig is, hetzij los, hetzij als onderdeel van een groter script.*** ***NB2:*** ***O dat de proporties van het vierkant bij opdracht 1 zijn beoordeeld blijven deze buiten be chouwing bij het beoordelen van deze opdracht. In theorie zou een rij van 10 dri hoekjes hier dus maximale punten kunnen opleveren.*** ***NB3:*** ***Let bij het beoordelen van deze opdracht op het daadwerkelijk gebruik van de drie aa ngeboden blokken (herhaal .. keer, pen op en verander x met 40). Oplossingen waarbij an lere blokken zijn gebruikt, leveren maximaal 1 punt op.***		
3	Als de (geïsoleerde) code wordt aangeklikt, tekent de sprite een rij van 10 vierkantjes.	• buitenste herhaallus is ingesteld op 10, in de lus de code van opdracht 1* , dan *pen op* en *verander x met 40* (evt. redundant blok pen op negeren). • als hierboven, waarbij *pen op* niet binnen de buitenste lus is toegevoegd maar wél aanwezig is in de binnenste lus (onderaan).
2	Als de (geïsoleerde) code wordt aangeklikt, tekent de sprite een rij vierkantjes, maar niet helemaal conform voorbeeld/opdracht.	• als hierboven, maar in de buitenste lus volgorde incorrect (eerst *pen op* en *verander x*, dan binnenste lus of eerst *pen op*, dan binnenste lus en dan *verander x*** of *verander x* vóór *pen op*) waarbij *pen op* niet in de buitenste lus aanwezig hoeft te zijn als deze wél aanwezig is in de binnenste lus (onderaan).
1	Als de (geïsoleerde) code wordt aangeklikt, verspringt de sprite 10x naar rechts zonder vierkant te tekenen of Als de (geïsoleerde) code wordt aangeklikt ontstaat er een rij vierkanten op het scherm, maar niet conform opdracht.	• de code is correct (herhaalblok 10 keer, daarin in juiste volgorde: *pen op* en *verander x*) maar niet (juist) gecombineerd met code opdracht 1 • de code is niet samengesteld uit de bij opdracht 2 gegeven blokken (*herhaal .. keer*, *verander x* en zo nodig *pen op*), maar is wél genest met de code van opdracht 1 (zodat een rij vierkantjes wordt getekend).

0	Er is geen (isoleerbare) code aanwezig die de sprite 10 keer naar rechts laat verplaatsen	• aan geen van bovenstaande criteria is voldaan

* on
toeeacht de kwaliteit van deze code, voor zover er bij opgave 1 minimaal 1 punt voor is gekend, zie ook NB1 en NB2

** dee code levert weliswaar een rij van 10 vierkantjes op, maar deze verspringt in afwijking n vaet getoonde voorbeeld naar rechts

opdracht 3: het hele blad vol

De leerling krijgt het script om het speelveld vol rijen vierkantjes te tekenen en moet dit script op de juiste manier combineren met de code van opdracht 1 en 2. Daarbij moeten in totaal 3 herhaallussen in elkaar geschoven worden.

punten:	**resultaat:**	**criteria (één van deze):**
3	Als de groene vlag wordt aangeklikt* wordt het speelveld volgetekend met 6 rijen in verschillende kleuren van 10 vierkantjes	• buitenste lus *herhaal 6 keer*, daarin de correcte code van opdracht 2 (dubbele lus) dan *verander x met -400*, *verander y met -60* en *verander penkleur met 10* (waarbij *verander penkleur* ook elders in de buitenste lus mag zijn geplaatst)
2	Als de groene vlag wordt aangeklikt* wordt het speelveld volgetekend met 6 rijen vierkantjes, maar niet helemaal volgens voorbeeld.	• het script van opdracht 3 vormt de buitenste lus, maar de code van opdracht 2 is op de verkeerde plek in de lus gevoegd (ná of tussen de blokken in de binnenste lus).
1	Als de groene vlag wordt aangeklikt* wordt er ergens op het speelveld iets getekend, maar niet volgens voorbeeld.	• de code bevat 3 geneste lussen waarbij de code uit opdracht 1 de binnenste lus is, daarbij zijn mogelijk blokken verwijderd, toegevoegd of verplaatst binnen de code.
0	Als de groene vlag wordt aangeklikt* wordt er iets of niets getekend.	• aan geen van bovenstaande criteria is voldaan

* Als de leerling vergeten is de gecombineerde code aan het startscript te plakken levert dat geen puntenverlies op.

Illustraties bij programma 2

optimale oplossingen:

→ opdracht 1
↘ opdracht 2
← opdracht 3

suboptimale oplossingen opdracht 2 (onder andere):

← 2 punten
→ 1 punt

suboptimale oplossingen opdracht 3 (onder andere):

← 2 punten
→ 1 punt

Programma 3: Cat vliegt

opdracht 1: besturing
Een serie vastgestelde blokken moet door de leerling zo worden gecombineerd dat Cat met de pijltjestoetsen bestuurd kan worden.

punten:	resultaat:	criteria (één van deze):
3	De besturing functioneert vlekkeloos. *oftewel: van uiterlijk, as en verplaatsing zijn **alle drie de elementen** overal correct uitgevoerd.*	• alle vier de besturingsscripts bestaan uit: hoed (gebeurtenis) corresponderend (aangepast) blok *verander uiterlijk naar* en corresponderend (aangepast) blok *verander x/y met 10/-10*
2	De besturing functioneert gedeeltelijk. *oftewel: van uiterlijk, as en verplaatsing zijn **twee elementen** overal correct uitgevoerd.*	• alle vier de besturingsscripts bestaan uit de bovengenoemde blokken, waarbij óf uiterlijk niet overal goed is aangepast, óf x/y niet overal goed is gekozen, óf 10/-10 niet overal correct is ingesteld.
1	De besturing functioneert beperkt. *oftewel: van uiterlijk, as en verplaatsing is **één element** overal correct uitgevoerd*	• Elk script bestaat uit de bovengenoemde blokken, waarbij óf uiterlijk overal goed is aangepast, óf x/y overal goed is gekozen, óf 10/-10 overal goed is ingesteld.
0	De besturing functioneert niet of nauwelijks.	• aan geen van bovenstaande criteria is voldaan

opdracht 2: nog een keer
De leerling moet het script schrijven dat Cat bij het einde van het spel weer naar zijn startpositie laat gaan. Daarbij moet de leerling **twee van vijf** aangeboden blokken selecteren en correct instellen.

punten:	resultaat:	criteria (één van deze):
NB:	**Bij het beoordelen van deze opdracht niet alleen letten op functionaliteit, zie criteria. De optimale code bevat slechts de hoed en twee blokken. Extra blokken zijn (bij verder goed functionerende code) redundant en weerspiegelen niet het inzicht dat groene vlag script slechts geduplicеerd hoefde te worden.**	

3	Na een succesvolle finish ontstaat exact gelijke beginsituatie als wanneer groene vlag wordt aangeklikt.	• op het scriptveld van Cat: hoed *wanneer ik signaal bericht1* ontvang, daaronder (ingestelde/aangepaste) blokken *ga naar x -194 y 136* en *verander uiterlijk naar Cat naar rechts* (volgorde niet van belang*)
2	Na een succesvolle finish ontstaat (ongeveer) gelijke beginsituatie als wanneer groene vlag wordt aangeklikt.	• als bovenaan, maar met **maximaal twee**** extra blokken toegevoegd (redundante code) • als bovenaan, maar x en/of y verschillen maximaal 20 posities, daarbij maximaal 1 extra blok toegevoegd (redundante code) • als bovenaan, waarbij blok verander uiterlijk ontbreekt en geen andere blokken zijn toegevoegd
1	Na een succesvolle finish keert Cat (ongeveer) terug naar beginpositie.	• script bevat minimaal blok *ga naar x/y*, ingesteld voor een positie in de linker bovenhoek van het speelveld
1	Na een succesvolle finish verdwijnt Cat, waardoor geen gelijke beginsituatie ontstaat met wanneer groene vlag wordt aangeklikt.	• script bevat (o.a.) ingestelde en aangepaste blokken *ga naar x 194 y 136* en *verander uiterlijk naar Cat naar rechts* maar toegevoegd blok *verdwijn* voorkomt dat Cat zichtbaar wordt op beginpositie.
0	Na een succesvolle finish keert Cat niet terug naar beginpositie.	• aan geen van bovenstaande criteria is voldaan

* Het inzicht dat het beginscript gedupliceerd diende te worden, is al getoond door de exacte instelling van de x- en y-positie.

** Bij meer dan 2 overbodige blokken heeft de leerling geen keuze gemaakt, maar slechts alle aangeboden blokken aan elkaar geklikt. Hoewel de code goed kan functioneren, weerspiegelt dit weinig inzicht, daarom geen 2 punten.

opdracht 3: game-over

De leerling moet aangeleverde code toevoegen aan het juiste script. Deze code laat Cat tegen de randen van het speelveld afketsen en bij stuurt hem bij contact met een wolk terug naar het begin.

punten:	**resultaat:**	**criteria (één van deze):**
NB:	**Als het spel niet optimaal functioneert door fouten in eerdere opdrachten en/of doordat het aangeleverde script gewijzigd is (bijvoorbeeld door abusievelijk klikken/slepen) kan toch aan de criteria voor 2 of 3 punten worden voldaan.**	
3	Het spel functioneert optimaal.	• script is toegevoegd aan groene vlagscript van Cat
2	Het spel functioneert optimaal.	• script is toegevoegd aan extra groene vlaghoed (door de leerling zelf aan scriptveld toegevoegd).*
1	Nadat Cat de eerste keer contact heeft gemaakt met de regenboog, functioneert het spel zoals bedoeld.	• script is toegevoegd aan hoed *wanneer ik signaal bericht1 ontvang*
0	Het spel functioneert niet zoals bedoeld.	• aan geen van bovenstaande criteria is voldaan
*	Deze oplossing leidt weliswaar tot optimaal functionerend spel, maar omdat deel van de code redundant is worden er slechts 2 punten voor toegekend.	

Illustraties bij programma 3

optimale oplossing opdracht 1 (3 punten):

optimale oplossing opdracht 2 (3 punten):

optimale oplossing opdracht 3 (3 punten):

Appendix D

Decomposition test: writing schedule/decomposition frame

Schrijfschema (decompositie) Wat moet er allemaal in je brief komen te staan? Schrijf alle onderdelen in een grijs vak. Als er te weinig grijze vakken zijn, mag je een vak in tweeën delen. Geef in de buitenste vakken voorbeelden van zinnen die je kan gebruiken bij het schrijven van je brief.				
		Schrijfopdracht: Schrijf een brief die je meestuurt met je punten en wikkels. Vertel waarom je geen 10 punten kunt opsturen. Overtuig de firma SMIKKEL ervan dat jij er niets aan kunt doen dat je geen 10 punten hebt, maar dat je toch graag de DVD wil krijgen. Zorg dat ze de DVD toch naar je opsturen!		
Tips voor het invullen van het schrijfschema: • Wat wil je vragen? • Wat moet je uitleggen? • Wat zijn je argumenten (redenen dat jij gelijk hebt) • Wat hoort er allemaal in een nette brief? (bijvoorbeeld: aanhef) • Welke informatie moet je verder nog geven? • Wat zijn goede zinnen die je kan gebruiken? (buitenste vakken)				

Appendix E

Decomposition test: evaluation framework

Transfertest Beoordelingskader

Beschrijving test

De leerlingen maken een decompositie van een brief die ze gaan schrijven a.d.h.v. een schrijfopdracht. De schrijfopdracht is sterk ingekaderd; zowel het opgetreden probleem als de voorgestelde oplossing zijn compleet beschreven in de opdracht.
De leerlingen werken de decompositie uit op een werkblad, waarbij in het middelste vak de schrijfopdracht staat. In vakken om de schrijfopdracht heen geven de leerlingen aan welke onderdelen in de brief moeten komen. In de buitenste vakken formuleren ze de zinnen waaruit ze de brief later kunnen opbouwen.
Een complete decompositie is samengesteld uit vier structuurelementen van een conventionele brief (A-D), de drie componenten die het opgetreden probleem beschrijven (E-G) en de vier componenten van de voorgestelde oplossing) (H-K).

Algemene aanwijzingen

Bij het beoordelen is alleen de aanwezigheid of afwezigheid van de opgenomen elementen van belang. Er vindt geen beoordeling plaats van de kwaliteit van de formulering op het gebied van woordkeus, zinsbouw of spelling. Het aantal aanwezige elementen bepaalt de score voor de decompositie, waarbij minimaal 0 en maximaal 11 punten behaald kunnen worden. Indien een beoordeling niet mogelijk is (bijvoorbeeld door onleesbaar handschrift of een technisch probleem) voorziet de beoordelaar het werk van code P.

element:	**aanwezig (+/-)**
A. bovenschrift; (plaats en) datum Aanwezig indien tenminste een datum weergegeven is waarop de brief geschreven zou kunnen zijn, dus vóór 1 maart 2017. Plaatsnamen en data in ander verband (bijv. in de probleembeschrijving) moeten hier genegeerd worden.	
B. aanhef Aanwezig indien er een openingsgroet genoemd/geformuleerd is, gericht aan de ontvanger van de brief. Tot wie de leerling zich daarbij precies wendt, is niet van belang. Zowel formele als minder formele formuleringen zijn toegestaan (vgl. geachte, beste).	
C. slotformule, afscheidsgroet Aanwezig indien een afsluitende formulering aanwezig is, die volgens de conventies gevolgd kan worden met de naam van de briefschrijver en/of ondertekening. Zowel formele als minder formele formuleringen zijn toegestaan (vgl. hoogachtend, groetjes).	
D. aanleiding Aanwezig indien een reden is aangegeven om de brief te schrijven, hoe summier ook. Niet aanwezig als de aanleiding tot het schrijven van de brief slechts blijkt uit de overige onderdelen. Geldige voorbeelden: • *Ik schrijf deze brief omdat ik een probleem heb.* • *Ik wil graag meedoen aan de spaaractie.*	
E. probleembeschrijving: 2 spaarpunten te weinig Aanwezig indien een duidelijk wordt gemaakt dat de briefschrijver 2 spaarpunten tekort komt, dat kan ook door te vermelden dat er slechts 8 spaarpunten verzameld zijn.	
F. probleembeschrijving: geen spaarpunten meer op verpakkingen Aanwezig indien vermeld is dat er (in de winkel) geen spaarpunten meer op de verpakkingen/repen zitten.	

G. probleembeschrijving: actieperiode is nog niet voorbij Aanwezig indien wordt vermeld dat de actie(periode) nog niet voorbij is of dat de einddatum van de actie (1 maart) nog niet is bereikt.	
H. oplossing: alternatief bewijs van de aankoop van 10 repen Aanwezig indien op de een of andere manier expliciet wordt gemaakt dat er geen 10 spaarpunten zijn meegestuurd, maar wel meegestuurd bewijs wordt geleverd van de aankoop van 10 repen. Dit kan ook door in dit kader te verwijzen naar de inhoud van de envelop (i.v.m. opdrachtbeschrijving die uitgaat van 8 meegestuurde punten en 2 wikkels).	
I. verzoek: toesturen DVD Aanwezig indien gevraagd wordt om een DVD of de wens is geformuleerd een DVD te ontvangen.	
J. verzoek: vermelden gewenste DVD Aanwezig indien een specifieke DVD wordt genoemd (Freek Vonk óf Kinderen voor kinderen).	
K. verzoek: adres voor bezorging DVD Aanwezig indien minimaal een straat en huisnummer worden genoemd waar de DVD heen gestuurd kan worden. Ook aanwezig indien alleen het begrip "adres" in de decompositie is vermeld.	
Totaal aanwezige elementen (0-11):	