Informatics in Education, 2014, Vol. 13, No. 2, 293–305

DOI: http://dx.doi.org/10.15388/infedu.2014.09

The Digital Woodlouse – Scaffolding in Science-Related Scratch Projects

Michael WEIGEND
University of Münster, Institut für Didaktik der Mathematik und der Informatik
Fliednerstr. 21, 48149 Münster, Germany
e-mail: michael.weigend@uni-muenster.de

Received: January 2014

Abstract. Scientific issues like the behavior of wild and domesticated animals can serve as a motivation to learn programming concepts. Instead of following a systematic introduction, the students directly dive into programming and start immediately with their projects. In this constructionist approach the educational challenge for the teacher is to provide suitable scaffolds like step-by-step instructions, architectural spike solutions, discovery questions, puzzles and role plays, which support individual and self-directed learning.

Keywords: Scratch, programming, scaffold, role play, tutorial.

1. Introduction

The Scratch programming environment has been designed with the intention "to nurture the development of a new generation of creative, systematic thinkers who are comfortable using programming to express their ideas" (Resnick *et al.*, 2009, p. 60). One way to learn Scratch – supported by the Scratch online platform – is to import a project and remix it to create something new. A basic problem of this approach seems to be that it is often difficult to understand a Scratch program, which has been created by someone else. According to Kafai *et al.* (2009), the complexity of programs made by young people in Computer Club Houses in their leisure time usually is rather low. Just a small percentage of the Scratch projects contain variables and control structures. The Scratch environment inspires young people to program but it seems that additional scaffolding is needed when it comes to developing more sophisticated projects.

Scaffolding is a Vygotskyan concept. The idea is to provide supportive assistance within the parameters of the learner's zone of proximal development (Wood *et al.*, 1976). Scaffolding involves limiting the complexity of activity that is required for solving a task to a degree that is suitable to the learner. The Scratch environment embodies scaffolding, since it reduces the complexity of programming. For example, dealing with

syntax errors is not necessary. Additional support is provided by the Scratch context help system: When you right-click a block and select the help-option, you see a little examples how to use this element. Furthermore, the Scratch website includes step-by-step tutorials and prototype projects ("starter projects") for different themes.

This contribution presents two programming projects that have been embedded into science projects in a natural science course in grade 7 (age 12 to 13) at a German comprehensive school. This class was extracurricular, the lessons were in the afternoon and the students did not get grades on their school reports for it. There were two groups with 15 students each, taught by two teachers in parallel. The primary object of this course was science not informatics. So the time for learning programming was very limited. The students were supposed to work on their own most of the time, using different types of scaffolds, which were specially tailored for this class and these projects.

The material included mini-tutorials, prototype projects, discovery questions, puzzles and role plays.

2. The Digital Woodlouse

The first project started with a "discovery activity". The students searched for woodlice on the school campus, and mapped the places, where they found some. They collected a few woodlice and measured their speed (while respecting the rules of animal protection) using cell phones (clock and camera) and rulers.

One of the observations made in this discovery phase was that woodlice avoid broad day light and try to hide in the dark beyond dead wood and leaves. This was the motive the first programming project with Scratch. The students got a four-pages tutorial consisting of four parts: Step-by-step instructions, architectural spike solution, discovery questions and a programming puzzle.

Fig. 1. Hands on activities before programming: Where do woodlice live?
What is the maximal speed of a woodlouse?

2.1. *Step-by-Step Instructions*

The tutorial explained step by step how to program a simple woodlouse simulation. The object representing the woodlouse has two costumes (like the Scratch cat) depicting the animal with two different leg positions. By switching between these costumes the illusion of moving legs can be created. The one and only script contains an endless loop causing the woodlouse sprite to move on a random path across the screen. The step-by-step instructions do not only cover the programming part but the complete development process including using the graphical editor, storing a file etc. Programming textbooks are mostly manuals explaining how to do things. The reception of step-by-step instructions is a way to practice basic literacy. You read an instruction, figure out the meaning and then immediately verify your interpretation by executing it using mouse and keyboard. If things do not work as expected, you rethink your text understanding and try again. Step-by-step instructions are more than just a one-to-one description of a procedure. They are more abstract; since they omit details the reader can figure out herself resp. himself. Additionally, when explaining what to do, the instructions inevitably use technical terms referring to programming and the user interface.

2.2. *Architectural Spike Solution*

The tutorial explained how to program a woodlouse that crawls over the screen. This was a minimal project with just one sprite, two costumes and one script (consisting of nine instructions). This was supposed to be just the starting point of a development. In Extreme Programming such prototype clarifying the basic structure of the whole projects is called “architectural spike solution” (Beck, 2003) (Fig. 2).

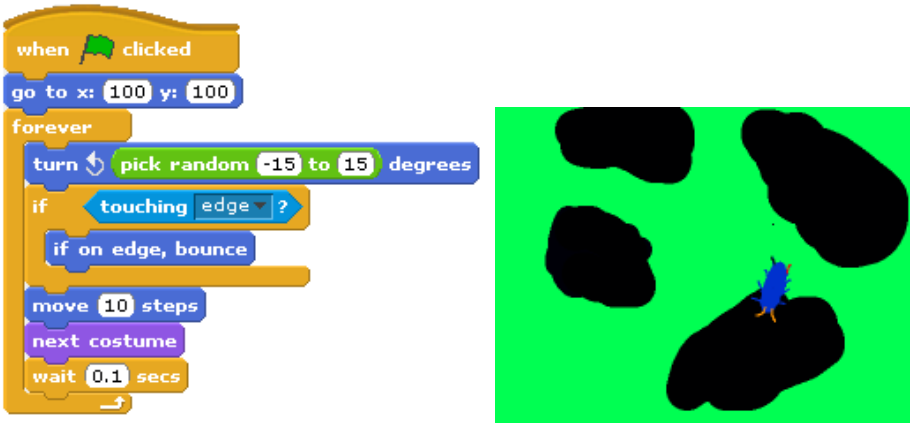

Fig. 2. Minimal script controlling a digital woodlouse and screenshot.

2.3. *Discovery Questions*

When somebody gets a small program and wants to extend it, he or she must understand its semantic first. The wording of the Scratch instructions already gives some idea about the meaning. Additionally the programmer may observe the behavior of the object and find the relation between behavior and program code. The manual contained some questions, which were meant to stimulate analyzing the Scratch program:

- What happens, when the woodlouse touches the edge of the stage?
- Which instruction causes the zigzag movement of the woodlouse?
- Which instruction causes the movement of the legs?
- What is to change to make the woodlouse move quicker? (There are two methods.)

2.4. *Puzzle 1: Make the Woodlouse Searching Dark Spots*

The tutorial also included a programming puzzle. A puzzle is a task with very limited degrees of freedom. It can be solved without additional information. This puzzle consisted of the actual task and three scaffolds (slightly abridged):

Task: Change the script of the woodlouse-object so that it tends to stay in the dark like a real woodlouse. It may move all the time but it must be most of the time at dark (black) places.

Hint 1: Real woodlice have eyes and sensors for perceiving their environment. The woodlouse object in your Scratch project has sensors, painted in a special color – say orange. You can use a sensing block like in the image (Fig. 3).

The condition `color orange is touching black?` is true, if the orange sensors are touching a black area on the background.

Hint 2: Use three of the blocks in the image (Fig. 3), modify them (if necessary) and put them into your script in an appropriate way.

Fig. 3. Instructions that might be of use.

Hint 3: Consider these situations (Fig. 4)! How does the woodlouse behave, when entering or leaving a dark area? Where does it move quicker? In the dark or in the daylight?

The solution is extremely simple but rather difficult to understand. The programmer has to insert a few blocks into the program (Fig. 5 inside the oval). They implement two effects:

- When the woodlouse is in the dark it moves slower (additional wait-statement).
- When it is not in the dark it turns to a higher degree (additional turn-statement) and tends to reverse the direction of movement.

Fig. 4. A digital woodlouse leaving and entering a dark area.

Fig. 5. A script simulating an elementary woodlouse.

2.5. *Puzzle 2: A woodlouse in a Maze*

A second woodlouse task (for quick students) was the implementation of a maze. The woodlouse had to find its way to the food (Fig. 6). The students got a description of the task, an almost finished script for the digital woodlouse and three blocks that had to be inserted (Fig. 7). This is another example of a puzzle with low degrees of freedom. 5 students reported that they solved this puzzle.

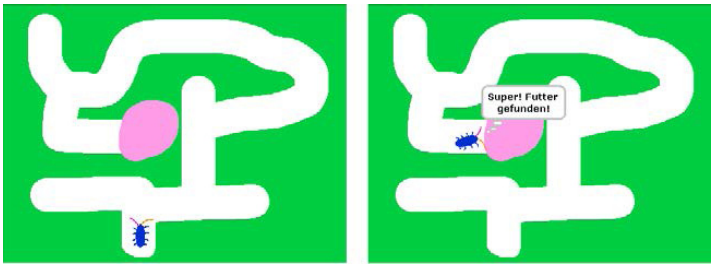

Fig. 6. Two screenshots from the woodlouse maze.

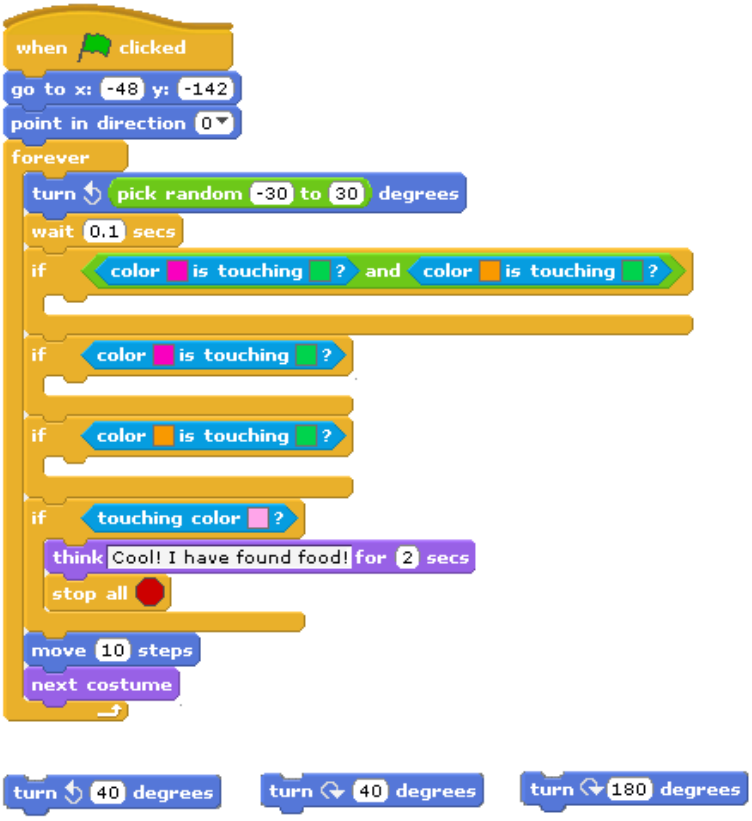

Fig. 7. The script – when completed – implements a Monte Carlo algorithm for finding food in a maze.

3. Animals in the Forest and on a Farm

The second project was about wild and domesticated animals. It was performed with the same students as the "digital woodlouse" project (grade 7). We started with a field trip to a small zoo, which is located close to the school. Each team (two or three students) had to choose a species (e.g. wild pigs or goats) that could be found in the zoo. The children watched the animals, recorded typical behavior and took photos and videos (Fig. 8).

In the second part the students designed an interactive animation about "their" animal. They had almost complete "artistic freedom" but were asked to follow these rules:

- Use your own photos for backgrounds and costumes.
- There has to be some interactivity.
- The animation must have a title which is displayed on a background image. The animation must have at least two different background images.
- The story may be fantasy but it must contain a few real facts about the animal.

This project is more complex than the first: there are several objects, not just one. The objects are not visible all the time and they interact using messages. The stage is an active entity which is able to change the background. The children developed the program in three iterations and got different types of scaffolds.

In iteration 1 the students were asked to develop a non-interactive animation, which showed a story similar to this. Having clicked the green flag you see a trail in a forest and the title of the animation "What is the wild pig looking for?" After two seconds the title vanishes and a wild boar appears at the right hand side of the image, glides to the middle of the scene and thinks "Maybe I can find some acorns here." (Fig. 9)

Fig. 8. Students observing wild and domesticated animals in a zoo.

Fig. 9. Screenshots from the architectural spike solution.

They got a manual explaining step-by-step how to implement this example. But – of course – they were encouraged to use their own pictures and create a different story with the animal they have chosen in the zoo.

3.1. *Roleplay 1: Interaction via Messages*

Mayer and Land (2003) use the term *threshold concepts* for those items, that are difficult to comprehend but which are "opening up a new and previously inaccessible way of thinking about something." A possible threshold concept in the domain of Scratch programming is the interaction of objects through messages. Before the students started programming they had to perform a role play that was supposed to visualize this idea. There is no function calling in Scratch. Object A makes object B doing something by sending a message m. Object B needs a script, which starts with the block `when I receive m`. This is a rather "natural" concept, which is used in everyday life. For the role play each student got a card with an informal script similar to a Scratch script, for example:

```
When I receive "Good morning",
  stand up
  wait until you hear a ring
  send a wave to everyone
  sit down again
```

There were four different scripts of this kind, causing a strange chain of events, which was triggered by a "good morning!"-message sent by the teacher. Students were standing up, sitting down, clapping their hands and waving according to a pattern that was induced by the scripts. The play was performed twice in each group.

In iteration 2 the students had to implement some interactivity. They got a tutorial explaining how to use the relevant blocks. In the example a new object depicting an acorn and a new background were created. The acorn was invisible at the beginning and appeared, when the pig was thinking about searching something to eat. When the user clicked on the acorn, the stage switched to the new background (scene 2, see Fig. 10, right hand side).

In iteration 3 the students were supposed to design the second scene, which should contain a dialog between the animal and another entity. In the example a fantasy dialog between the wild boar and a windflower was suggested. (The pig tells the wind flower that it likes its roots.)

Fig. 10. Screenshot from an animation in iteration 2.

3.2. *Roleplay 2: A Scratch Project as a Theatre Play*

This time – instead of a tutorial – the students got a complete listing of the example project with the wild boar (Fig. 11). The complete listing depicts a box for each object and the stage. Inside each box there are all scripts and thumbnails of all costumes. Additionally there are pointers between message-sending and corresponding message-receiving instructions. The complete listing corresponds to a screenplay. An actor needs to know the complete text of the play – not just his own – to understand his part. A Scratch programmer has a "big picture" in his or her mind but on the screen just the scripts of one object are visible. The others have to be memorized. The complete script reduces this cognitive load (Sweller, 2003). The complete listing is a mild abstraction of the Scratch project. The names of the costumes and the view on the stage are missing. Instead of colored blocks it just depicts the program text of the scripts.

The complete listing was used for a role play with five actors: the stage, three sprites and the user. The rest of the class was watching and checking whether or not the actors were behaving according to the scripts. At the whiteboard there were three simple drawings representing the three background images of the stage. The actor playing the stage part had to point at the background image that was just shown. Again, the play was performed twice. The first time there were a lot of interruptions and discussions about what to do in each situation.

The students developed their projects with individual speed. Only a few were able to implement stories with two interacting entities as it was suggested in iteration 3. However, everybody could manage to create *some* interactive animation. Some products were rather strange, even when they contained the required "true facts" and self-made photos. For example one boy made an animation depicting a goat with sunglasses living on the surface of the moon. But this goat loved to eat carrots just like the one in the zoo.

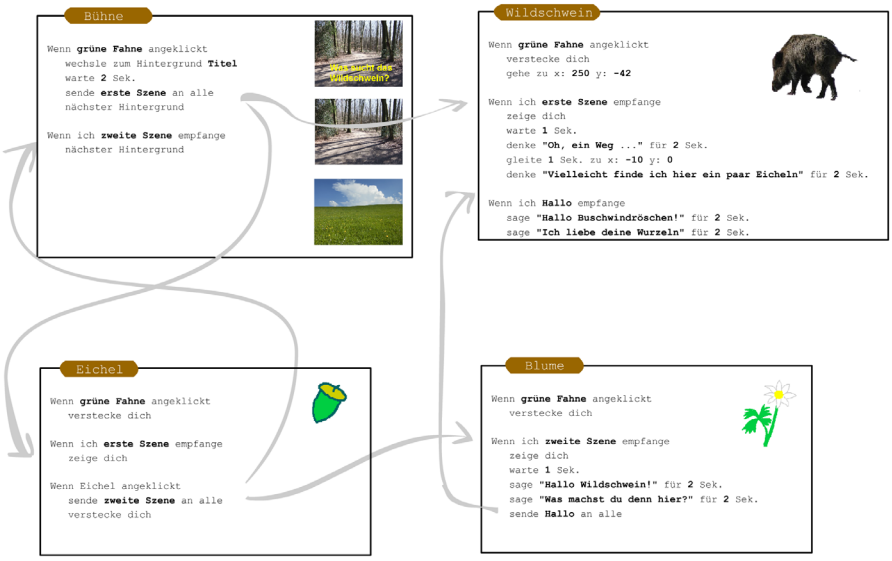

Fig. 11. A complete listing of a Scratch project.

4. Evaluation

26 students (17 boys, 9 girls) answered a questionnaire at the end of the projects. They were asked to check statements about their attitudes toward the development process and their usage of scaffolds. In this section some findings are summarized.

There were children who did not read step-by-step instructions but preferred just to try out or to ask other persons to explain how things work. 58% of the students reported that they followed the written instructions. 38% stated that they read some parts of the tutorials even several times.

The appropriate way to use a tutorial is to read just one instruction, try to do the required thing until it works and then go on with the next instruction. It seems that many students were just not familiar with this procedure. 31% reported that they did not read the step-by-step instructions at the beginning but found out later that these were rather helpful. According to the numbers in Table 1 it seems that the readers of the step-by-step instructions had an advantage. They were more successful, were more satisfied with their products and more of them wanted to go on with their projects.

When you regard the group of success-related statements 1, 2 and 6, the differences between readers and non-readers are significant (2-tailed exact Fisher's test yields $p = 0.00013$). The same is true for the combination of the attitude-related statements 3, 4, 7, 8 ($p = 0.012$) and the wish to spend more time on the projects (statements 5 and 9, $p = 0.034$).

The numbers displayed in Table 2 suggest that some students did not profit much from the role plays. 8 resp. 10 out of 26 students considered them to be a waste of time. This does not necessarily mean that role plays are principally useless for these students. One must take into account that interaction via messages is a difficult concept and many students failed to implement it in their project. On the other hand, 11 resp. 8 students stated that they got a better understanding of programming concepts through role plays.

Table 1

Frequencies of statements checked by students that read step-by-step instructions (readers) and those who did not (non-readers)

Statement		Frequency	
		Readers (n=15)	Non-readers (n=11)
1	I have created a woodlouse that crawls across the screen.	14 (93%)	6 (55%)
2	I have created a woodlouse that searches dark areas.	8 (53%)	2 (18%)
3	I was satisfied with the result of my work.	8 (53%)	5 (45%)
4	I was proud of the result of my work.	3 (20%)	0 (0%)
5	I would like to spend more time on this project.	4 (27%)	0 (0%)
6	I have created an interactive animation with an animal.	15 (100%)	5 (45%)
7	I was satisfied with the result of my work.	9 (60%)	4 (36%)
8	I was proud of the result of my work.	7 (47%)	0 (0%)
9	I would like to spend more time on this project.	5 (33%)	1 (9%)

Table 2
Relevance of role plays

Statement	Frequency (n=26)	
	Role Play 1	Role Play 2
Through this role play I got an idea how message passing works (play 1) resp. how the Scratch program works (play 2)	11	8
I thought of this role play later, when I was working with Scratch.	5	3
I told other people about this role play later.	2	2
This role play was a waste of time. One could have skipped it.	8	10

5. Conclusion

This contribution has presented Scratch projects that were performed in a science class. The primary motivation was not to learn and practice programming concepts but to visualize scientific ideas. Programming was just one of several methods to elaborate scientific content. In such a setting, scaffolds are needed that are focused on the primary task (e.g. "model a woodlouse"). Such include role plays and tutorials with prototype examples ("spike solutions"), discovery questions, puzzles and step-by-step instructions. Students seem to profit from tutorials, but they need to know how to use them. Many do not. They need some kind of introduction to reading and interpreting step-by-step-instructions.

Let me conclude with a few remarks on the prototype projects, which are explained in tutorials and which serve as a starting point for individual developments. They are essential, since they can define the whole technical idea (architecture) of a project. This is a computer science related aspect. But there is a further point. A prototype project might be a minimal representation of a scientific concept.

Consider the example of "dynamic equilibrium", one of the basic concepts in chemistry. Most chemical reactions (like ester synthesis and hydrolysis) work "in both directions". Molecules of type A change to molecules of type B and vice versa. This leads to a dynamic equilibrium. The amount of molecules A and B remain constant although there is permanent change. This principle can be visualized by an agent-based model.

Fig. 12 depicts a minimal prototype with just seven instances of just one type of sprite. Each sprite has two costumes and switches, when it hits an edge. The one and only script consists of not more than nine lines of code. This is very simple. The model is chemistry but the implementation is computer science.

The selection, design and documentation of useful prototype projects in a way that is suitable for children of different age groups is one of the tasks computer science teaching has to take care of.

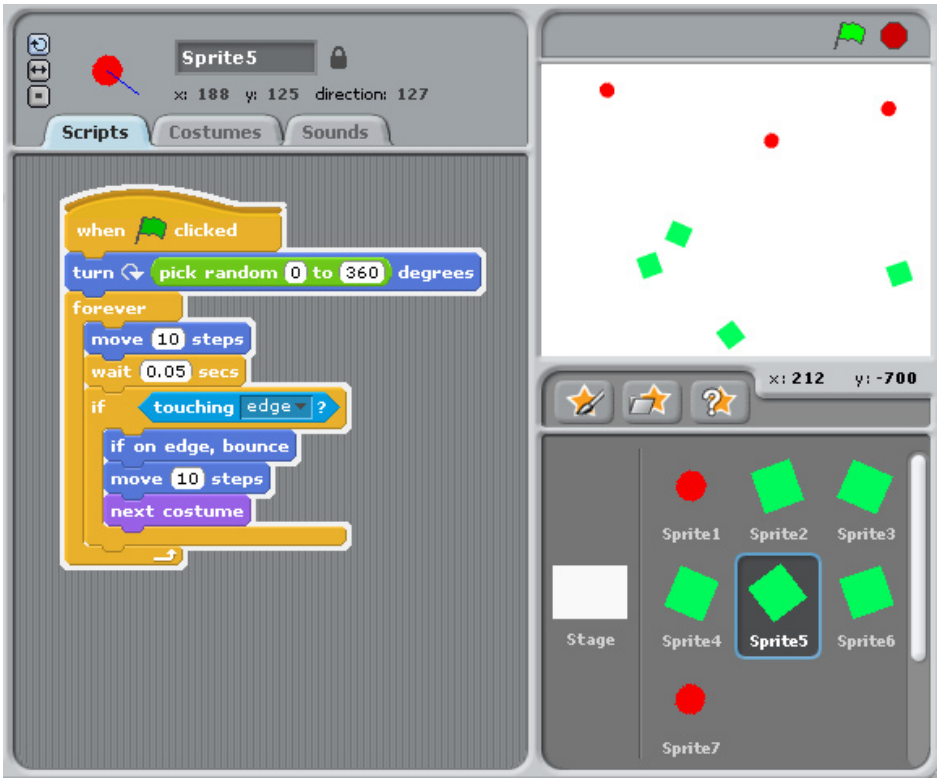

Fig. 12. Agent based simulation of a dynamic equilibrium with nine instructions.

References

Beck, K. (2003). *Extreme Programming Explained*. Addison Wesley.

Kafai, Y.B., Peppler, K.A , Chapman, R.N. (2009). *Computer Clubhouse: Constructionism and Creativity in Youth Communities*. Teachers College Press, New York.

Meyer, J.H.F , Land, R. (2003). Threshold concepts and troublesome knowledge: linkages to ways of thinking and practicing. In: Rust, C. (Ed.), *Improving Student Learning – Ten Years On*. OCSLD, Oxford, 412–424.
https://www.dkit.ie/ga/system/files/Threshold_Concepts__and_Troublesome_Knowledge_by_Professor_Ray_Land.pdf

Resnick, M., Maloney, J., Hernández, A.M., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E. , Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60–67.

Sweller, J. (2002). Visualisation and instructional design. In: Ploetzner, R. (Ed.), *Proceedings of the International Workshop on Dynamic Visualizations and Learning*. Tübingen, 1501–1510.

Wood, D., Bruner, J.S., Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry and Allied Disciplines*, 17(2), 89–100.

M. Weigend studied Computer Science, Chemistry and Education at the University of Bochum and at the University of Hagen and received a PhD in Computer Science from the University of Potsdam. He is a teacher at a secondary school in Witten, Germany and he has taught Computer Science Education at the University of Hagen for almost 20 years. He has published several books on computer programming, web development and visual modelling.

Pedagogo parama – pagrindas „Scratch“ projektuose

Michael WEIGEND

Mokslinės problemos, pavyzdžiui, laukinių ir naminių gyvūnų elgesys, gali būti motyvacijos šaltinis siekiant suvokti programavimo sąvokas. Vietoje nuoseklaus sisteminio įvado, mokiniai tiesiogiai pasineria į programavimą ir įsitraukia į pačių sukurtus projektus. Žvelgiant iš šios konstrukcionistinės perspektyvos, mokytojui keliamas iššūkis suteikti tinkamą paramą, pavyzdžiui, pažingsninę instrukciją, architektūrinius sprendimus, atradimų klausimus, dėliones, vaidmenų žaidimus, kuriais būtų skatinamas individualus ir savarankiškai organizuojamas mokymasis.