ISSN 2676-9425

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 3

Number 2

2021

Eötvös Loránd University
Faculty of Informatics
Budapest, Hungary

Contents

Possibilities of Simulating Robot Generations in Public Education

ABONYI-TÓTH Andor

Abstract. An important part of teaching robotics is describing different robot generations, which represent the characteristic stages of technological development. These generations are also identifiable in case of educational robots. The categories refer to the tasks and problems which can be solved with a given robot. However, the development of robot generations can also be presented to students with the help of algorithmization and coding tasks which simulate the behaviour of robots in a programming environment. These activities can prepare students to work with real educational robots or complement the activities they do with them. This article summarizes the opportunities and advantages of the simulation of robot generations, and shows a concrete example of their implementation.

Keywords: Robotics, Robot Generations, Simulation, Scratch, RoboMind**,** VEXcode VR, Virtual Robotics Toolkit, MakeCode for LEGO MINDSTORMS Education EV3

1. Introduction

When introducing students to robotics, it is worth assessing the existing knowledge children have about the topic. For that, first we have to define the equipment which children perceive as robots. It can be done in a brainstorming session when we collect the robots they know of and mention some tasks those robots can perform. [1]

In these sessions, students often mention everyday objects (e.g. robot vacuum cleaners, robot lawn mowers, humanoid toy robots, industrial robots in an assembly plant, etc.) and fictional robots existing only in films and books (e.g. shape-shifting robots, doctor robots, combat robots). Of course, it is worth clarifying which robots actually exist, and which aspects of fictional robots are rooted in reality.

The robots collected by students can then be grouped into a number of categories, such as location of use, complexity of activity, size, nature (e.g., humanoid), and so on. We can come up with a diverse classification by simply focusing on the activities robots can perform. The groups may include robots performing simple or more complex tasks. At this point we can tell students that robots can be classified into different generations based on their technological level. [2]

2. Robot generations

The primary characteristics of first-generation robots is that they perform their actions according to a specific program and are unable to sense their environment. In this category are, for example, some industrial robots with pre-programmed path of movement and activities. In this case, if a workpiece is positioned incorrectly, the robot will not be able to perform its task under the changed conditions.

Second-generation robots are more advanced. They have a variety of sensors, so they can modify their movements, activities and operation using the information gained of their environment. For example, they can avoid an obstacle in their path.

Third-generation robots have even more advanced artificial intelligence, they can recognize shapes and situations, and they can solve certain tasks through machine learning. They can solve certain

tasks independently in an unfamiliar terrain; Mars rovers, for example, even find their way on an alien planet.

3. Robot generations in education

Educational robots have several generations too.

3.1 First-generation robots

First-generation robots include, for example, floor robots whose path of movement can be programmed by the physical buttons on them (forward, backward, turn right or left, wait), or by an external application. These robots travel along the specified tracks according to the pre-programmed code. They can be used with the original paths supplied with the robot or new paths programmed by the students. Various tasks and problems can be associated with the paths. Floor robots are ideal for developing students' algorithmic thinking in the lower grades of elementary school, and they can help in the playful acquisition of directions, numbers, shapes, colours, and so on. [3]

Figure 1: Blue-bot floor robot[1]

Figure 2: An example floor mat[2]

Certain floor robots also have simulation software which can be used to design and simulate floor robot activities on an interactive whiteboard. In the "Focus on Bee-Bot 3[3]" application available for Bee-Bot robots, the robot can move on different maps and tracks, with its movement programmable with the buttons appearing on the screen. The actions of the robot can even be performed step by step, or children can follow the traversal of the path from the robot's point of view, as if a camera were attached to the robot. The software includes a number of challenges which children can solve on their own or in groups.

[1] https://bit.ly/3avl4qr

[2] https://bit.ly/2x09w0A

[3] https://www.focuseducational.com/category/item/6

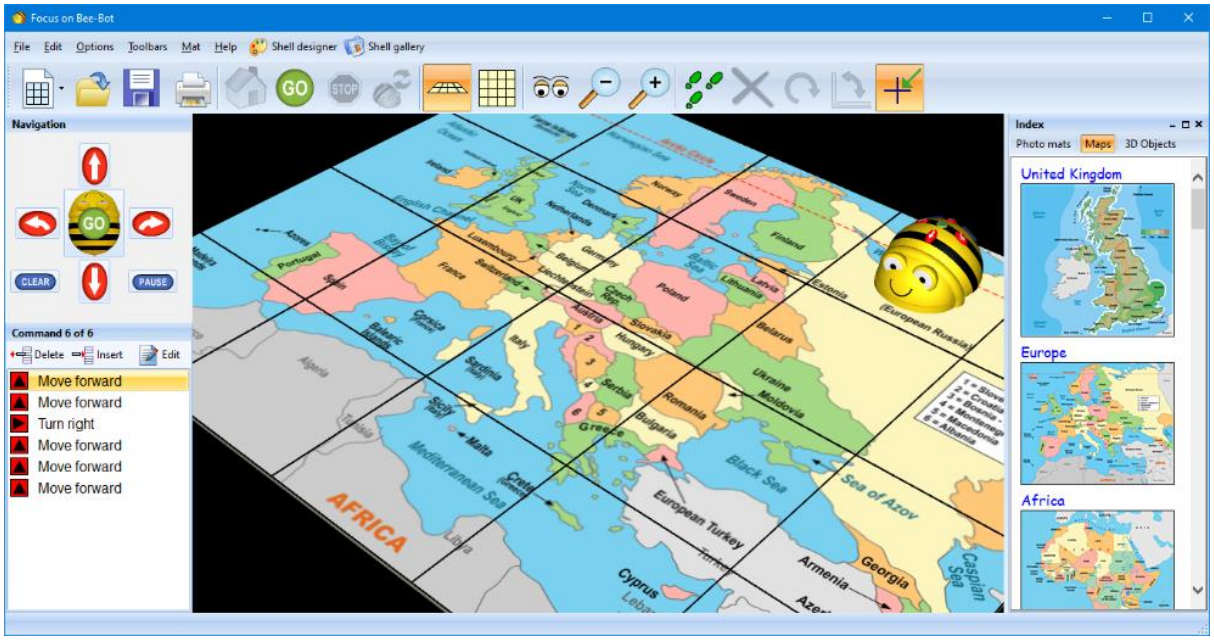

Figure 3: Focus on Bee-Bot 3 software with virtual floor robot and map

3.2 Second generation robots

Second generation robots are robots and kits which can process the data acquired by their sensors for completing the tasks. There is a wide variety of these educational robots. Which kit to choose may depend on the age group, the type and the features of the programming language, the quantity and quality of the related teaching materials, the available financial resources, and the specifics of the teaching task. [4]

If students have already become familiar with programming the deservedly popular micro:bit microcontroller during their studies, it may be worthwhile to obtain kits based on this chip. With those, students can use the familiar device and programming interface to solve robotics-related tasks.

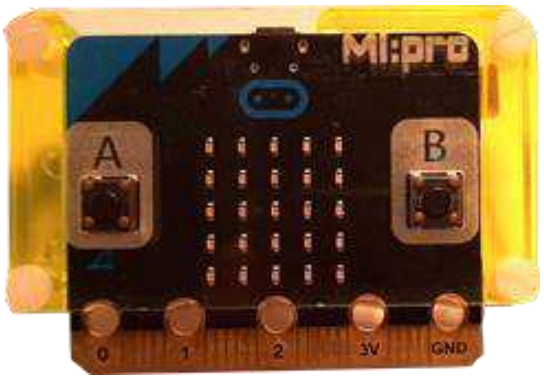

Figure 4: BBC micro:bit microcontroller

The kits differ in the range of sensors included in their basic package and available as optional accessories.

The following sensors are most commonly found in the kits: [5]

- Ultrasonic sensor: it measures the distance between objects or detect the ultrasound emitted by another source or reflected by a surface. It is great for measuring the distance between two robots (vehicles), making automatic doors, building a parking radar, etc. It is also suitable for detecting ultrasound from other sources. For example, a robot may respond to a nearby robot emitting ultrasound.
- Colour sensor: it recognizes various colours, which allows for the creation of a robot which can sort objects of different colours, follow a line or a more advanced one which solves the Rubik's cube.
- Light sensor: measures the intensity of the reflected light or ambient light. It can be used, for example, to enable the robot to stop at the edge of the table when it no longer detects the light reflected from the tabletop.
- Touch sensor: detects when a button is pressed or released. For example, we can make a robotic hand which closes when something touches it.
- Gyro sensor: The gyroscopic sensor is used to detect the angle of rotation of the robot. It can be used, for example, for building a balancing robot.

Figure 5: EV3 Touch sensor, Colour sensor, Ultrasonic sensor4

Robotic vehicles equipped with the required sensors may also be suitable for becoming familiar with second-generation robots. For example, a Bit:bot vehicle for use with a micro:bit chip includes two drive motors with rubber-coated wheels, two line tracking sensors, an analog light sensor, and an ultrasonic distance sensor at the front of the vehicle. Thus, the tool can be used to complete both line tracking and obstacle avoidance tasks.

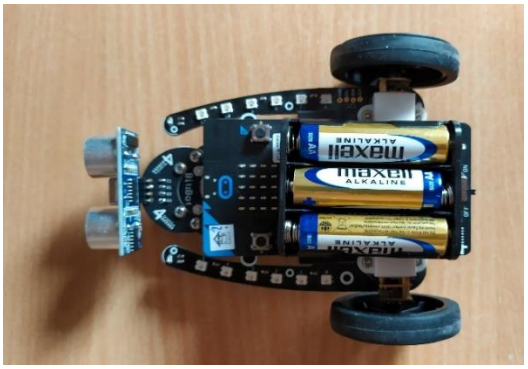

Figure 6: micro:bit controlled Bit:bot vehicle[5]

[4] http://www.legoengineering.com/ev3-sensors/

[5] Photo by Bence Gaál

3.3 Third-generation robots

Third-generation robots play a role in research and development related to artificial intelligence and therefore require high-level programming skills. They have high computing power and allow the use of sensors with advanced functionality.

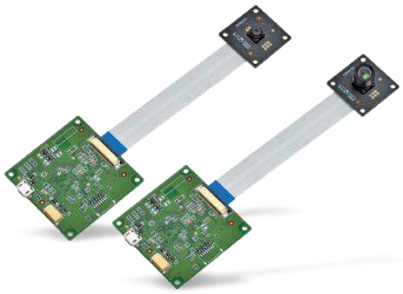

Figure 7: Image Sensor Module[6]

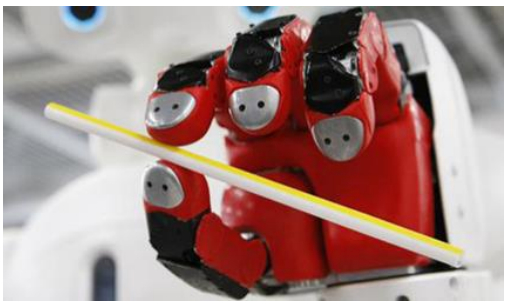

Figure 8: Tactile Sensor[7]

The development of self-driving vehicles requires good quality distance measurement, object recognition, visual and proximity sensors. The acoustic sensors allow the robot to receive voice commands or process sound effects from the environment. Third generation robots typically have a camera mounted on them, whose image must be processed. The processing tasks may include shape and emotion recognition, which can be very important in the case of robots doing social work.

These robots are typically used in institutions above the secondary school level, in tertiary and adult education. For this reason, we do not address this topic in more detail in our article. The third generation robots can also be simulated in a complex environment such as the Webots simulator.[8] The simulator can be used for modelling and programming, and the simulation of the robot's behaviour. [6]

Figure 9: Insect-shaped robot[9]

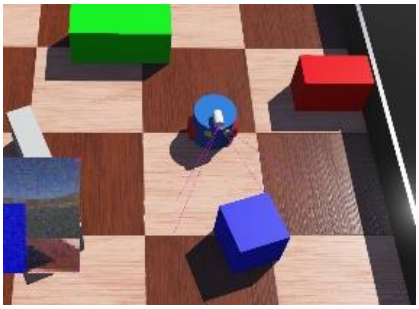

Figure 10: MyBot uses a camera to detect coloured objects[10]

[6] https://hu.mouser.com/new/omronelectronics/omron-b5t-hvc-p2-sensors/

[7] https://pressureprofile.com/nl/robotics

[8] Webots. Commercial Mobile Robot Simulation Software. https://cyberbotics.com/

[9] https://cyberbotics.com/doc/guide/samples-demos

[10] https://cyberbotics.com/doc/guide/samples-devices

4. The potential benefits of simulating robot generations in public education

The development of robot generations can even be presented to children through creating algorithms and writing code by which we can simulate the behaviour of a robot. Such activities can be good complements of working with real educational robots. Some of the activities can even be performed in an 'unplugged' way.

4.1 Unplugged activities

„*Teaching Unplugged is the term used for a teaching method and philosophy which has three primary aims: teaching through conversation, taking out external input such as course book and technology and letting the lesson content be driven by the students rather than being pre-planned by the teacher.*" [11]

Let's see an example of this method. The following task can be read in the Digital Culture textbook used in the 5th grade of Hungarian public education system: [7]

"*Go to a place with a large free area. There, mark a starting point and an end point. Form groups of 4 to 5 students.*

Each group should have a student whose eyes are blindfolded. Create an algorithm that you think will guide your partner from the starting point to the destination. Give verbal instructions to your partner according to the algorithm. What are the experiences? Did your partner actually get to the destination? Now we issue the instructions so that our partner travels exactly the way back. Did your partner get back to the starting point?

Discuss what important lessons you learned about control during the game. How could the instructions have been made clearer? Think of similarities and differences in the control of the real robots compared to this game."

By solving the above task, children can gain many experiences. It is possible that

- the 'robot' did not perform the expected activity because the instructions were not clear;
- the 'robot' could not interpret the issued instruction (e.g. did not understand it due to noise);
- the 'robot' executed the instructions correctly, but the algorithm contained errors;
- although the 'robot' successfully reached its destination and made its way back, it did not reach the starting point;

The experience gained during the 'unplugged' activities can be used in the programming of robots simulated in real or computer environments. It may happen that the real robot does not perform the expected activities (e.g. due to incorrect code and/or algorithm), does not execute the instruction (e.g. a voice-controlled robot does not interpret the command due to noise, or the command contains a syntax error). The robot's movement may also be influenced by external factors. For this reason, the robot does not necessarily return to its starting position, even if the program for traversing the whole path is correct. This can be caused by a variety of factors, such as an uneven or slippery surface, discharging batteries, etc.

The largest collection of unplugged activities for use in computer science education can be found on the CS Unplugged website. "*CS Unplugged is a collection of free teaching material that teaches Computer Science through engaging games and puzzles that use cards, string, crayons and lots of running around.*"[12]

[11] https://www.teachingenglish.org.uk/article/teaching-unplugged

[12] https://csunplugged.org/en/

The site also contains session descriptions for getting acquainted with robotics. An example is "Harold the robot". [8] The session is designed to program the "robot" to build a tower from the building blocks placed on the table, by executing elementary instructions. The role of the robot can be played by the teacher or a student. Students need to figure out what elementary instructions the robot responds to and how they can be used to build the tower. After completing the task, students can gather together which commands the robot has responded to and which it has not, and discuss how more complex tasks can be accomplished using a series of elementary steps.

Of course, the sessions can also be organized so that students are familiar with the basic instructions the robot can execute. The students' task is to place the instructions in the correct order to solve the given problem. The task can be done with the instructions written on pieces of paper, which can also be attached to a white board with magnets. The simulated activity can be, for example, controlling a robotic arm. [9]

Also popular are classes in which students create algorithms with which a robot can build different patterns by stacking plastic cups on top of each other (My Robotic Friends) [10]. During the session, the students take turns creating the algorithm and playing the role of a robot for the execution of the algorithm defined by symbols. In this way, they learn the relationship between the operation and the symbols, the difference between the algorithm and the program, and also gain proficiency in debugging the code.

4.2 Activities in a simulated environment

Activities in a simulated environment allow students to gain sufficient knowledge and experience in the field of programming, so that they could later program physically existing, more complex robots later.

If the school has fewer robots than would be necessary for the sessions, then while one group is programming a real robot, the other can work in the simulated environment and then they can switch.

If the school does not yet have the robots, the basic concepts of robotics can be introduced in the simulated environment. There are several initiatives available for Hungarian schools (e.g. 'wandering robots' [11], 'wandering micro:bits' [12], in which schools can borrow various devices (e.g. floor robots, micro:bits) free of charge. More advanced robotics kits are also expected to be available for borrowing in the future. As these devices are only available in a school for a limited time, students should be taught their programming in advance in simulated environments, so that they could use the kits efficiently to solve advanced tasks when they have access to them.

Simulations allow us to set homework assignments which students could not otherwise do or try out in the absence of the necessary equipment.

It is much easier and faster to define paths for a particular type of problem in a simulated environment than in a real one. The paths can be drawn by the students, so besides completing a programming task, they use their image editing skills too. They can also customise the shape of the robot if the program has that option. The students can even use an animated figure as a robot, whose creation requires creativity and a better knowledge of the equipment.

Moreover, we can define a task in which the robot's program is given, and create a path individually or in a group which meets the conditions set in the task (e.g., the robot must collect the fallen fruit, but avoid poisonous mushrooms.)

Another advantage is that we can try different algorithms and problem-solving strategies without risking causing damage to real, sometimes very expensive devices. While collisions make no difference in a simulation, they can significantly shorten the lifespan of real robots. The methods and solutions tested in the simulator can then be used for the programming of real robots.

The route travelled by the robots can be easily illustrated by the robot drawing a line after itself. Of course, this can be done in a real environment by attaching a felt-tip pen to the robot, and the mat can be made of an easily erasable material. The advantage of this method is that the routes traversed can be easily compared, illustrating that a given problem can be solved in several ways.

If the students have already gained experience in using floor robots and we want to introduce them to block programming environments, we can add introductory tasks which rely on their knowledge gained in the robotics sessions. This way we can teach them the basics of programming and using various control structures.

The simulator allows teachers to better present certain phenomena (e.g. operating model of line follower robots, traversal algorithms, obstacle avoidance algorithms) to the whole class.

If students do not have a background in robotics but have already used block programming environments (e.g. Scratch), we can set novel, playful problem-solving tasks from the field of robotics.

We can also teach functions which are not supported by the available real robots, such as picking up/laying down objects on a track, programming loading and rearrangement tasks.

Students can also be introduced to programming languages which go beyond block programming (e.g. LOGO, Robo).

Keep in mind, however, that programming physically existing and moving robots is a highly motivating activity for children, so simulations should not be used instead of that, but as complementary method. It is very important that students have hands-on experience with real robots in their studies and, if possible, solve real-life, open problems in project work. [13]

4.2.1. Simulation in Scratch 3.0 environment

The Scratch environment can be used as block programming[13] environment to simulate the activities of robots. A simulation environment should be designed from a top view of the robot and the course. The shape of the robot drawn must clearly indicate what direction it is facing.

In the Scratch environment, students can draw the shape of a robot and create animated shapes. The path that the robot must travel along can be easily drawn using either built-in or external drawing tools. The tracks and the robot images can be easily replaced.

The robots can initially be controlled with keystrokes (e.g. cursor keys) and then we can switch to writing programs which allow the creation of more advanced control structures.

The distance unit parameter must be set for each track to allow the robot to move a certain distance or turn when necessary. Parameters are crucial in the programming of real-life robots, so students should learn about them in the simulated environment as well.

Using the pen plug-in, the robot can draw a line after itself, so the users can keep track of its route.

[13] https://scratch.mit.edu/

Sensors can also be simulated. The robot can be programmed to stop if it encounters an obstacle. For that is advisable to draw the obstacle/wall with a specific colour. In the Scratch environment we can ask if our robot has encountered a line with a given colour during its movement. It allows us to simulate the movement of a second generation robot which moves forward until it detects an obstacle. A common task in teaching robotics is when the robot must be programmed to follow a line. It can be simulated in the Scratch environment, which is described in a later chapter in more detail.

With the Video sensing plugin available in Scratch, we can present creating applications with the help of a webcam image, in which we can control participants with our movement or detect the colours appearing in the video image. This could be a transition to simulating third-generation robots.

4.2.2. Simulation in the RoboMind environment

ROBO is an imperative/procedural programming language. With the help of this language, you will gain an insight into areas such as robotics and artificial intelligence. The language consist of basic instructions to control the robot, repetition loops, conditional if ... then ... else statements, the possibility to define instructions yourself by creating procedures.

In Robo, we can program in a development environment called RoboMind.

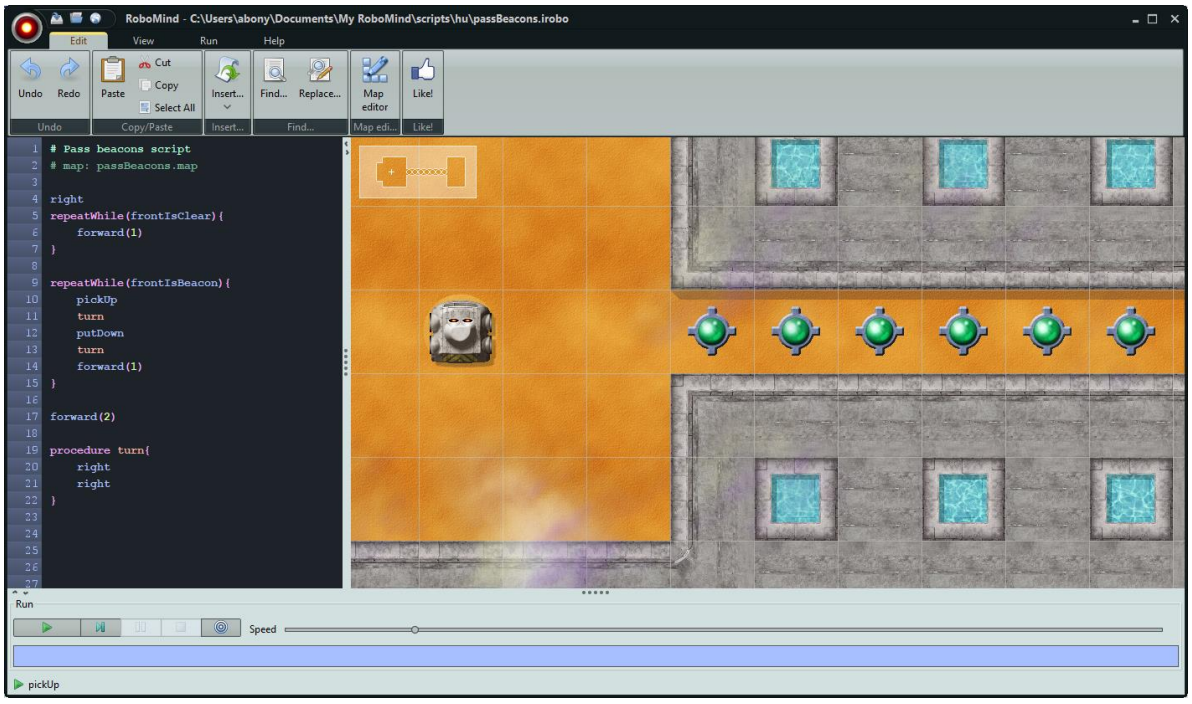

Figure 11: The RoboMind development environment

The virtual robot can navigate in a square grid environment by moving forward, backward, and turning left and right by 90 degrees. The directions can also be set according to the four cardinal points.

We cannot draw the robot in this environment, but we can choose from several existing shapes. The course can be easily created with the help of the built-in map editor. On the course we can

place a wall, various landmarks (crates, plants, water surface), a beacon, and routes marked in white or black.

The advantage of the environment is that the simulated robot can also be controlled with a remote control displayed on the screen (forward, backward, right, left), so that tasks that may be important for younger children can be solved initially, without programming.

The robot can mark the traversed route in white or black. In addition, it can pick up / put down objects (beacons) on the track or "eat" them, which means that it deletes the object after picking it up.

The robot can detect whether there is an obstacle to the left, right or in front of it, whether the path is clear, whether there is a beacon, and whether the road is painted white or black.

The environment is suitable for setting up tasks typical of first-generation robots. Examples are route and area exploration tasks (knowledge of the course, pre-programmed), rearrangement tasks (rearrangement objects of known location and number). The behaviour of second-generation robots equipped with sensors can also be simulated through various tasks. Examples are path finding tasks (the path to be followed is unknown), getting out of a maze, search tasks (finding and collecting objects on an open course or one containing few obstacles), rearrangement tasks (with the position and/or number of objects previously unknown), traversal tasks (comparison of random and systematic traverses). [14]

4.2.3. Simulation in the VEXcode VR environment

VEX Robotics released the VEXcode VR simulation environment in April 2020. It can be used in a browser, so there is no need to install special software. The website allows virtual use of the VEXcode environment used to program VEX 123, GO, IQ and V5 robots. The interface supports both block programming and programming in Python, so it is versatile for use in public education.

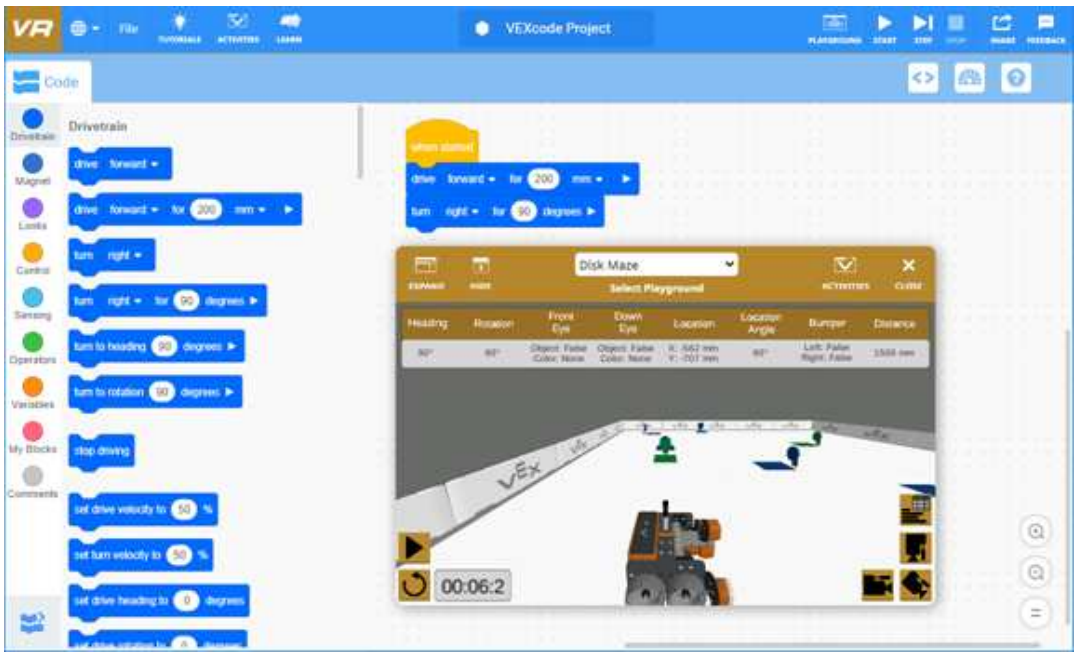

Figure 12: VEXcode VR environment

The virtual robot is placed in a playground where it executes the specified code. There are several types of playgrounds: a grid, a maze, a castle floor plan, a coral reef, etc., which are associated with different types of problem-solving tasks. The tasks are also accessible on the website. The track and the robot can be displayed in 2D (top view) or in 3D, with several camera views are available for the latter.

The environment can be used to simulate the activities of both first and second generation robots. Sensors include a rotation sensor, left and right impact sensors, a distance sensor, front and bottom colour sensors/brightness sensors, and a location sensor.

4.2.4. Simulation in the Virtual Robotics Toolkit environment

The Virtual Robotics Toolkit is a development environment for LEGO® MINDSTORMS® robots that can be installed on Windows/Mac operating systems. It can be used with a subscription.

The virtual robot can be programmed in the LEGO® MINDSTORMS® EV3 environment. The physical parameters can be modified in the simulation, so it is possible to simulate how the robot would operate in a state of weightlessness. 3D models can also be imported in the environment.

The following sensors can be used in the simulated environment: MINDSTORMS EV3 – Ultrasonic sensor, Color sensor, Touch sensor, IR sensor, Gyro Sensor, HiTechnic – Infrared sensor, Compass sensor.

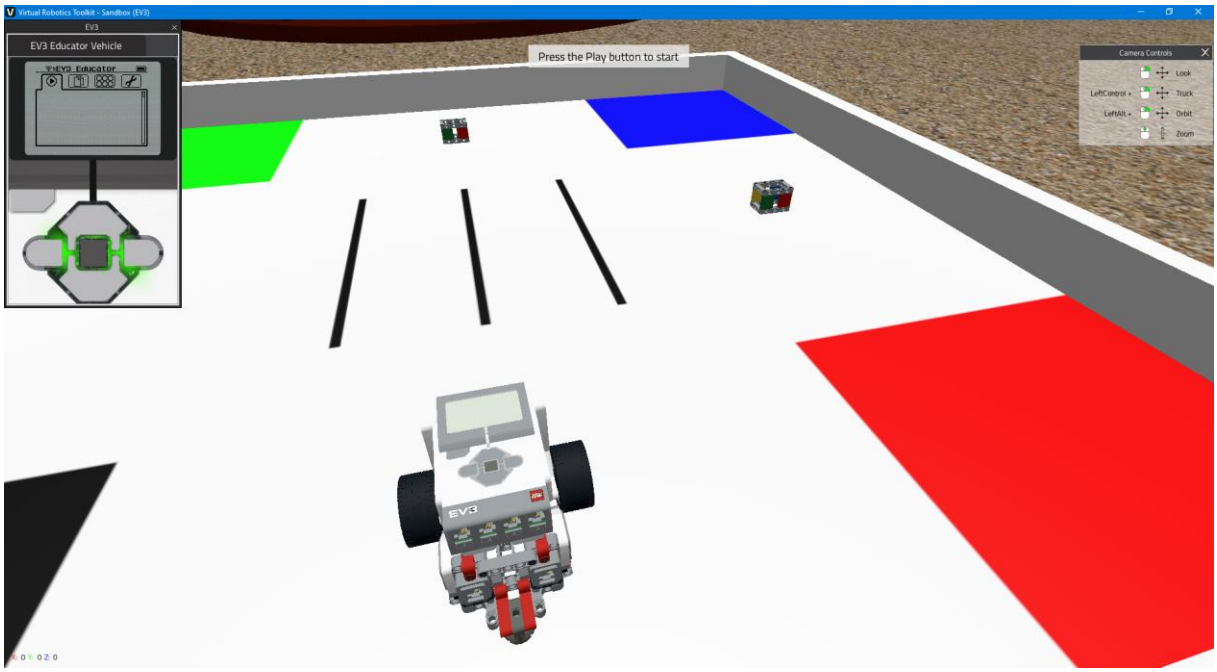

Figure 13: Virtual Robotics Toolkit environment

4.2.5. Simulation in MakeCode for LEGO® MINDSTORMS® Education EV3 environment

LEGO® MINDSTORMS® EV3 robots can be programmed using an online, free environment supporting[14] both block programming and JavaScript programming.

The interface has a simulation option, but this is different from the ones seen before. It is not a robot moving in a virtual space, but an EV4 brick (the brain of the device) to which different components can be connected, e.g. touch sensors, colour sensors, ultrasonic distance sensors, gyro sensors, infrared sensors, as well as large and medium-sized motors.

In the simulator, we can set the values measured by each sensor, so we can test whether our program is working properly (in principle), and we can also see when a motor is switched on/off, and check its rotation direction and power.

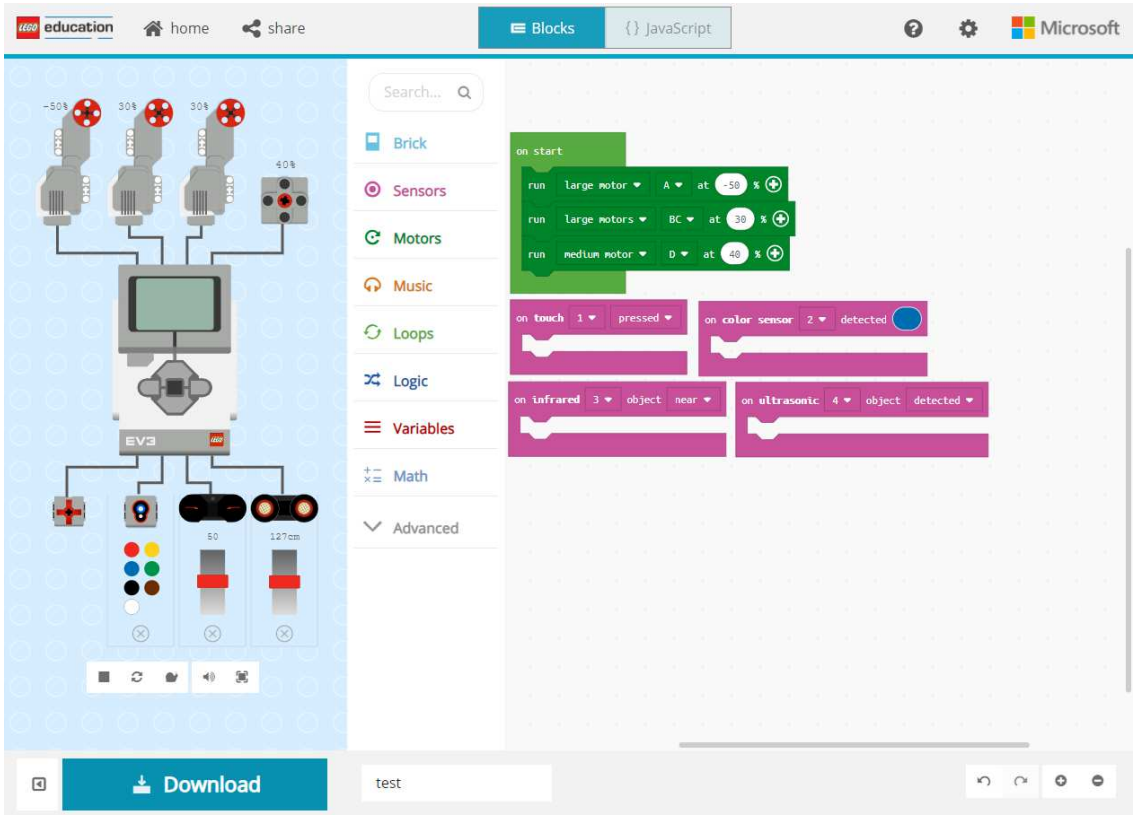

Figure 14: MakeCode for LEGO® MINDSTORMS® Education EV3 environment

This environment is therefore recommended for the development of a program for the operation of more advanced, individually assembled robots.

[14] https://makecode.mindstorms.com/

4.3 Summary of the possibilities of the environments

The following table summarizes the functions provided by each simulation environment.

	Scratch 3.0	**RoboMind**	**VEXcode VR**	**Virtual Robotics Toolkit**	**MakeCode for LEGO® MINDSTORMS® Education EV3**
Is it free to use?	☑	☑ v. 6.0 ☒ v. 7.0	☑	☒	☑
Can it be used in a browser?	☑	☒	☑	☒	☑
Supported operating system (offline version)	macOS 10.13 or later Windows 10 or later	v 6.0: Windows v7.0: Linux, macOS, Windows	-	Windows, macOS	Windows, macOS
Does it support block programming?	☑	☒	☑	☑	☑
Supported programming languages	☒	ROBO	Python	☒	JavaScript
What type of robot can we simulate?	General	General	VEX 123, GO, IQ and V5	LEGO MINDSTORMS NXT/EV3	LEGO MINDSTORMS EV3
Simulation environment (2D/3D)	2D	2D	3D	3D	2D
Suitable for simulating the movement of a robot vehicle	☑	☑	☑	☑	☒
Does it support creating custom tracks?	☑	☑	☒	☑	☒

It is visible that there are significant differences between the individual environments based on the examined aspects.

If you are looking for a browser-based environment which supports block-level programming, and it is not very important to see the robot's operation in 3D, then the Scratch environment may be the right choice for the simulations.

If you want to simulate the operation of a robot vehicle in 3D and are satisfied with the tracks offered by the manufacturer, we recommend the VEXcode VR environment. If the environment or the tasks need to be customised, or if your students go on to work with LEGO MINDSTORMS NXT/EV3 robots later, the Virtual Robotics Toolkit may be the right choice.

If you want to introduce students to the basics of robotics with an easy-to-use, imperative/procedural programming language, the RoboMind environment is the perfect choice. The easy modification of the tracks and the variety of basic instructions allow you to design a wide range of activities.

In the MakeCode for LEGO® MINDSTORMS® Education EV3 environment, the simulation is less spectacular, but it provides an opportunity to simulate the operation of a custom-built EV3 robot. Therefore, we recommend using this environment if it is also part of the task for students to build a robot on their own to solve a problem.

5. Case study

Let us examine a line following task in detail. In real life, the route is usually marked by a duct tape glued to the floor or a table, which allows for its simple modification. One solution may be to mount sensors on the front, left and right sides of the line tracking robot which detects either the intensity of the reflected light or its colour. The goal is to keep the centre of the robot on the line as it moves. If the left sensor detects the line, it means that the line is turning left, so the robot must also turn left by a certain degree. The same applies to turning right.

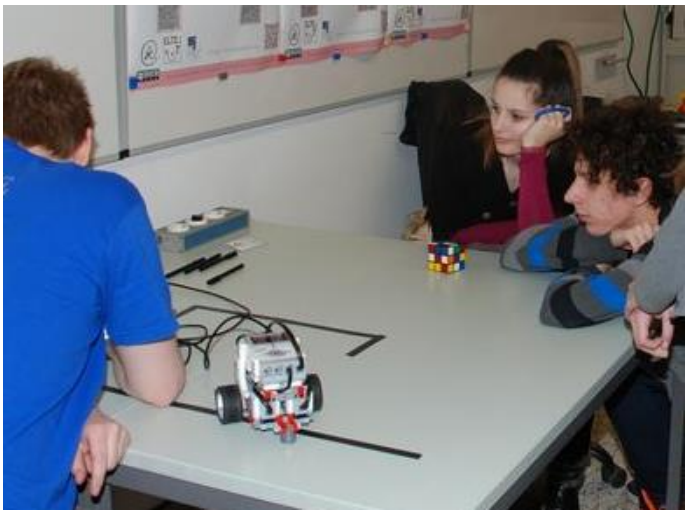

Figure 15: Robotics demonstration with a line-following robot in ELTE University's T@T Kuckó (T@T hub)[15]

Another solution may be a colour or light sensor mounted on the bottom centre of the robot. The simplest algorithm in this case is that when the vehicle moves away from the left edge of the strip,

[15] http://tet.inf.elte.hu/tetkucko/galeria/

it has to turn right until it finds its way back to the strip and then turn left afterwards. That is, the vehicle will perform a zigzag movement. In this implementation the strip must be relatively wide, as the vehicle can easily pass over a thin line.

Let us see an example how a line following task can be simulated in different simulation environments.

5.1 Line following simulation in RoboMind environment

Below is the track drawn in the RoboMind environment and a possible solution of the task. In the task, a white line on a square grid defines the path the robot should follow. The target where the robot should stop is marked at the end of the white line. The path should be such that the white line is not in the immediate vicinity of the robot, but falls into its path if it starts to move in the set direction. Let us see how this problem can be solved in the two environments.

The solution is that we advance 1 step until the field in front of us is white. We then go through the line with a loop which checks whether the target is in front of us (the execution of the loop stops), whether there is a white field in front of us (we move forward), or whether there is a white route on the left or right, as then we have to turn in the right direction.

Figure 16: The path drawn in the map editor

```
repeatWhile(not frontIsWhite){
     forward(1)
}
repeat
{
 if(frontIsBeacon){end}
elseif(frontIsWhite){forward(1)}
     else if(rightIsWhite){right}
     else if(leftIsWhite){left}
}
```

5.2 Line following simulation in Scratch environment

The solution is a little more complex in the Scratch environment, but that is an advantage in this case, as it shows in more detail how the task would be implemented in the real world. There are several types of sensors which can be used in the Scratch environment. As we mentioned, real robots often complete the task by colour perception, so we can try to do that in a similar way in Scratch.

A good way to do it is to draw a robot shape which has two antennas on the left and the right side, with ends marked with different colours. Those colours should also be different from the colour

of the path. If the colour of the sensor on the right antenna touches the colour of the route, the robot should turn right, and go on. The same applies for turning left.

In this case, a large black circle marks the target on the course. The left antenna of the robot is blue and the right is red.

The implementation is similar to the solution created in Robo, but in this case it must be ensured that if the left or right sensor detects the colour of the route, the robot should not turn immediately, but take so many steps forward that its middle be approximately above the centre of the line. That distance depends on the shape of the robot as well as the unit used on the route. The other difference compared to the previous solution is that the two colour sensors can be used to better illustrate the implementation with real robots, thus making better use of the knowledge gained for the construction of real robots.

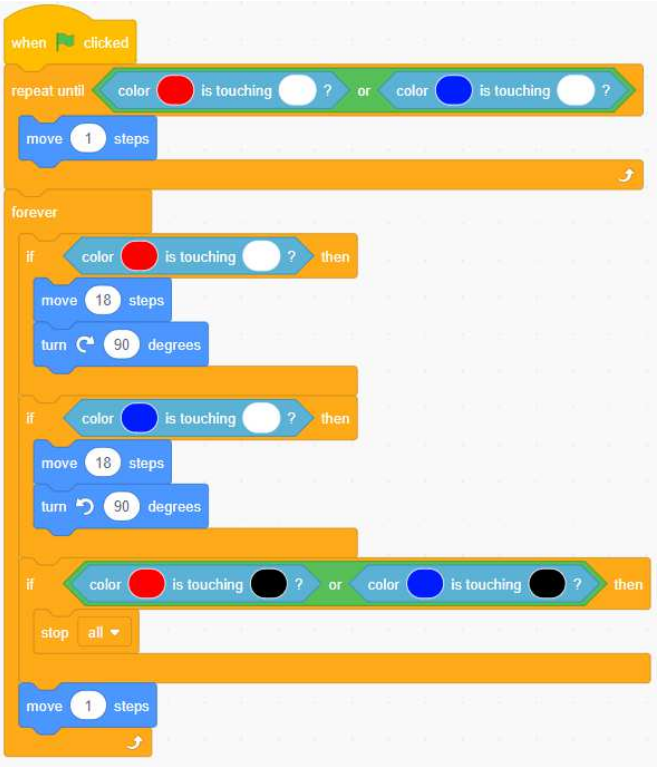

Figure 17: The Scratch code

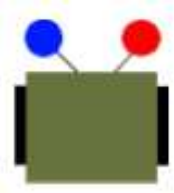

Figure 18: The drawn robot

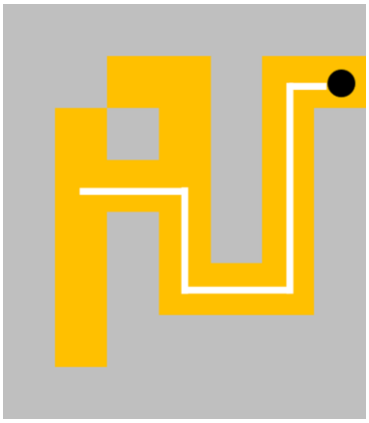

Figure 19: The path

5.3 Line following simulation in VEXcode VR environment

Unfortunately, we have not been able to solve the task in this environment. The playground in which the robot travels is pre-programmed and it cannot be modified (at least at the moment), so it is difficult to draw the track containing the line. The robot can draw with a pen, but it should get back to the starting point with the pen raised. However, even though we drew the line, the robot's sensor did not detect it. The colour recognition function only works for objects placed on the track. We hope that the possibilities of the VR environment will be expanded in the future, so that more advanced tasks can be solved in the environment.

5.4 Line following simulation in Virtual Robotics Toolkit environment

In this simulation environment we found a pre-designed track (Maze (EV3)) in which we could solve the line tracking problem.

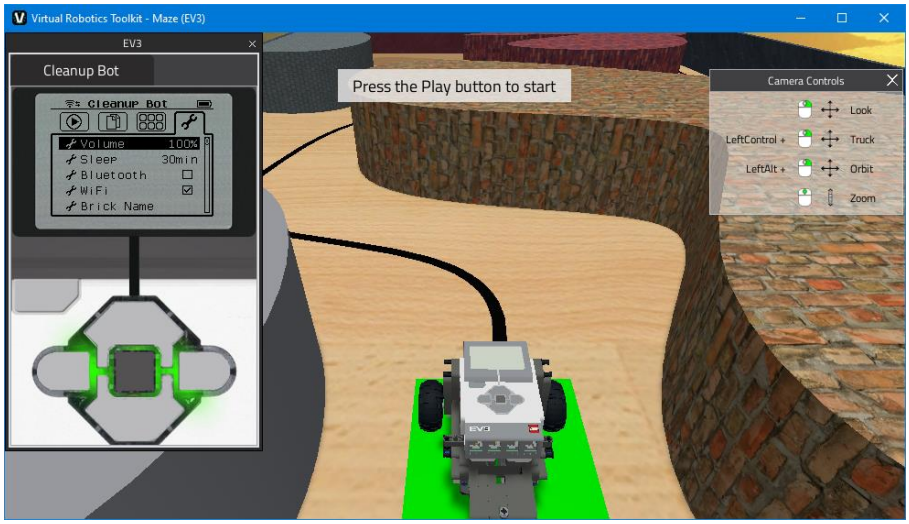

Figure 20: A built-in track for line tracking tasks

When the track is loaded, the robot's starting position is in a green area, while its colour sensor detects the black path. So, a program must be generated which follows the black path. You can see that the route does not turn at right angles as in previous cases, so the solution will be different.

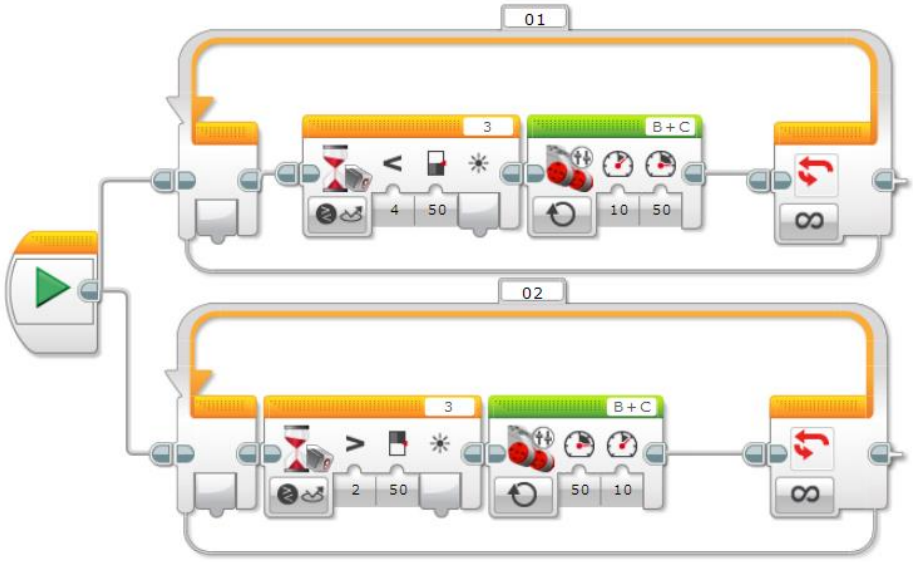

Figure 21: The completed program

In the above program, we examine whether the intensity of the reflected light is less than or greater than a threshold (in this case 50), and depending on this, we turn either left or right with the robot, so that it follows the edge of the line. Turning is done by changing the power of the left and right engine, respectively. When the power of the left engine is 10 and that of the right engine is 50, it means that the vehicle is turning left. The greater the difference in power between the two engines, the smaller the curve.

6. Conclusion

This article outlines the possibilities and benefits of simulating robot generations in public education, highlighting the fact that these activities can well complement (but not completely replace) the work with real robots.

We have demonstrated that students can gain important experience in controlling and programming robots, even by unplugged activities, which they can later use during practical activities.

Several computer environments are available to simulate the activities of robots. Some of these simulate the operation of specific robot types, while in others we have the ability to simulate general (or fictitious) robots. Important aspect when choosing an environment may be the programming languages in supports, the software's ease of use, the customizability of the environment and the tracks, the option to simulate the movement of robotic vehicles, and whether it uses a 2D or a 3D visual representation. The main features of the simulation environments are also summarized in a table, which helps in selecting the appropriate environment.

In the case study, we showed how a line tracking task can be solved in different simulation environments. During the implementation of the specific task, we found that there are very different options in each environment for setting and customizing the properties of the environment, which greatly affects the extent to which we can set our own, unique tasks for students.

Bibliography

1. Pluhár Zsuzsa: Robotikáról tanároknak. [About robotics for teachers.] 2019. [on-line] ⟨https://bit.ly/3aAW697⟩
2. Dr. Kodácsy János, Dr. Pintér József (2011): Szerszámgépek és gyártórendszerek. [Power tools and production systems] [on-line] ⟨https://bit.ly/2zfRRml⟩
3. Lénárd András (ed.) Az algoritmikus gondolkodás fejlesztése padlórobotok segítségével. [Development of algorithmic thinking with the help of floor robots.] 2018. Budapest, Hungary: Stiefel Kft.
4. Gaál Bence: Comparative analysis of sets used in robotics education. In: Proceeding of Didmattech 2020 Conference. [on-line] ⟨http://didmattech.inf.elte.hu/proceedings-2020/⟩
5. John Burfoot: EV3 Sensors – LEGO Engineering, 2018. [on-line] ⟨http://www.legoengineering.com/ev3-sensors/⟩
6. Michel, Olivier: WebotsTM: Professional Mobile Robot Simulation. International Journal of Advanced Robotic Systems. 1. DOI: 10.5772/5618. 2004. [on-line] ⟨https://bit.ly/2xRP96i⟩
7. Pintér Gergely (ed.) Digitális kultúra 5. tankönyv [Digital culture 5. school book] 2020-Budapest, Hungary. Oktatási Hivatal. ⟨https://www.tankonyvkatalogus.hu/pdf/OH-DIG05TA__teljes.pdf⟩
8. Richard Nelson, Jason Clutterbuck, Sebastian Höhna, Stefan Marks and Wilson Siringoringo: Harold the Robot. [on-line] ⟨https://classic.csunplugged.org/harold-the-robot-2⟩

9. Miller, Blanca & Kirn, Adam & Anderson, Mercedes & Major, Justin & Feil-Seifer, David & Jurkiewicz, Melissa. (2018). Unplugged Robotics to Increase K-12 Students' Engineering Interest and Attitudes. 1-5. DOI: 10.1109/FIE.2018.8658959.

10. CS Fundamentals Unplugged | Code.org [on-line] ⟨https://code.org/curriculum/unplugged⟩

11. Péter Fehér, Dóra Orsolya Aknai: Wandering Robots in Hungarian Primary Schools: a Case Study. ECEr 2019 Conference. [on-line] ⟨https://bit.ly/2XXOZEV ⟩

12. Abonyi-Tóth Andor, Pluhár Zsuzsa: Wandering microbits in the public education of Hungary- LECTURE NOTES IN COMPUTER SCIENCE 11913 pp. 189-199., 11 p. 2019.

13. Jonassen, D. H.: Toward a design theory of problem solving. Education Technology Research and Development, 48 (4), 63--85. (2000). DOI: 10.1007/BF02300500.

14. Bernát Péter: Robotika az általános iskolában és a RoboMind programozási környezet Paper:3 [Robotics in primary school and the RoboMind programming environment Paper:3] In: Péter Szlávi; Zsakó, László (ed.) INFODIDACT 2015, Budapest, Hungary: Webdidaktika Alapítvány, (2015)

Authors

ABONYI-TÓTH Andor

Eötvös Loránd University, Faculty of Informatics, Department of Media and Educational Informatics, Hungary,
e-mail: abonyita@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 3, Number 2. 2021

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.3.2.1469

License

Curricula and Methods on Teaching Different Aspects of Agile Software Development

ILYÉS Enikő

Abstract. Agile methodologies are the most commonly used software development methodologies nowadays. For this reason, education of software engineering should include this topic at universities. However, the educational method of this topic has still open questions. Higher education lacks traditional curricula and methods in teaching agile software development.

Agile methodologies are based on collaboration and interaction. For this reason, high level teaching of agile methodologies requires not only teaching of theory and practice, but also development of soft skills and of an appropriate set of values. We need to find teaching methods that can develop students in all four aspects: theory, practice, skills, values. This article presents methods for developing all four aspects of agile methodologies. In addition, by describing five different course curricula, it proves these methods can be effectively incorporated into classic university courses.

Keywords: education, agile, Scrum, skills, values

1. Introduction

Agile methodologies are very popular nowadays in the software development industry. As a result, university programs in the field of software development have begun to consider teaching agile methodologies necessary.

Application and teaching of agile methodologies can be analyzed from several aspects. One aspect is the theory of these methodologies, such as what events and roles are defined within them. The practical aspect is to experience how the methodology works. Soft skills can be considered as a separate, third aspect: agile methodologies are very communication and collaboration based, so high-level applications of agile methodologies require high-level use of soft skills. The fourth aspect is the agile values, which must be continuously manifested during application of the methodology. Only this way operation of agile methodologies will be credible and effective. In case of Scrum, these values are courage, respect, openness, commitment, focus.

A common solution to teaching agile methodologies is to implement a full semester software development project in teams, most often adapting Scrum. Another approach of instructors is to teach Scrum through a simulation project.

While articles on the topic mostly discuss how the teaching or application of agile methodologies have been realized in a single course, this paper discusses education of agile methodologies in different courses of an entire software engineering educational program, which is the main contribution. At different levels of the program, in mandatory, optional, or elective courses, students can get to know agile software development. From undergraduate to graduate, 5 different courses offer an understanding of agility from 5 different perspectives. Lectures, seminars, practices, and research groups included in this package can be understood as modules, and their combination allows students to acquire in-depth knowledge about agility.

This article presents the details, methods used, aspects developed in the 5 different modules of this package. It seeks to answer the following research questions:

1. Can different aspects of agile software development (theory, practice, skills, values) be taught at university?
2. What kinds of methods can be used to teach the aspects of agile software development?
3. In what kind of curriculum can these methods be integrated?

We set as acceptance criteria that we should define at least 3 different methods for the development of each aspect. These should be incorporated into at least 3 curricula. In the case of the implemented curricula, it must be demonstrated that the four different aspects have been developed by the courses in an amount of minimum 70%. The experiment should involve at least 130 students.

Therefore, following a brief clarification of key concepts (Chapter 2), the reader can find out what teaching methods we defined to be used to teach all four aspects of agility (theory, practice, skills, values) at university (Chapter 3). In addition, he/she will see examples of 5 different course topics and implementations that successfully integrated these teaching methods (Chapter 4). The article concludes with discussion and related work (Chapter 5). All this serves as inspiration for other universities or educational institutes for teaching agile software development in a comprehensive way.

2. About agile software development and Scrum

This chapter clarifies the fundamental concepts referred to in this article.

Agile methodologies are software development methodologies that implement the principles expressed in the Manifesto for Agile Software Development: emphasizing individuals and interactions, customer collaboration, responding to change, working software over comprehensive documentation. [1]

Scrum [2] is the most commonly used agile methodology [3]. It defines 3 Scrum roles, 5 Scrum events and 3 Scrum artifacts. The Product Owner represents the interests of the customer. The Scrum Master is responsible for ensuring that workflows are implemented efficiently and along Scrum values. The development team is a self-organizing team who has all the skills necessary to implement the product. The workflow consists of sprints. The sprints are fixed work periods of 1-4 weeks during which a new increment must be created. Each sprint begins with planning, during which the Scrum team commits to a sprint goal. Then they work together and discuss their progress over a short 15-minute meeting each day - this is called the daily Scrum. At the end of the sprints sprint review takes place, ideally in the presence of the customer, where they give feedback on the result of the sprint. This meeting is also suitable for brief discussion of additional needs of the product. The sprints are closed by the sprint retrospective, where the Scrum team analyzes its operation and optimizes it. During Scrum events the different actors of Scrum need to use their soft skills a lot, for example: communication, collaboration, adaptation, critical thinking. Their manifestations should be based on the Scrum values, which are as follows: respect, courage, openness, commitment, focus.

Prioritized product requirements are stored in the product backlog, managed by the Product Owner. Some of the elements of the list are selected by the team during the sprint planning in cooperation with the Product Owner, and a sprint goal is determined. The selected items are placed in the form of tasks in the sprint backlog, which the team manages during the sprint. An increment is created when a product backlog element is implemented and matches the definition of done.

The definition of done is fixed by the team. It is evolving during the project with the intention of creating the most stable and reliable increments possible.

3. Methods for teaching agile software development

This chapter defines methods for teaching the theory, practice, skills, and attitude aspects of agility in four separate subsections. For each method, it gives a general description and then helps a deeper understanding through an example.

3.1. Methods for teaching the theory

Lecture: The instructor introduces a topic to a larger group of students, orally, sometimes with the help of tools (ex. keywords, important definitions, figures listed on slides/whiteboard). Communication in this case is mostly one-way: the instructor speaks, the students listen.

Example: The target is to teach the Scrum roles. First the instructor highlights the main idea behind each Scrum role. Then he/she explains in detail each role's responsibilities and the qualities beneficial to have for that specific role. The oral presentation is supported by the following slide set: a slide with each role's main target; two more slides about each role, summarizing which responsibilities and qualities should the people in these roles have.

Reading literature: The instructor selects a literature, and the student has to read it carefully.

Example: The target is to make students understand Scrum in detail. The instructor selects The Scrum Guide [2] as obligatory literature - this should be read by the students. During the examination, the students have to know details that were not mentioned during classes but are presented in this document.

Literature review: The students - independently or in teams - have to categorize, compare, summarize the basic ideas of the literature related to a given topic.

Example: The aim is to give insight into agile transformation. Students should create a list of causes and workflows for agile transformation based on a number of literatures.

3.2. Methods for teaching the practice

Semester-long software development project - from the developer's perspective: A group of students work on a software development project during an entire semester. Each student is solely a member of the development team (not a leader).

Example: The aim is to provide a possibility for students to experience the workflow of Scrum. Students should develop in teams of 3 a game software (for example: Adventure park) adapting the Scrum methodology. The role of the Product Owner is fulfilled by the instructor, and the Scrum Master is a senior student. The semester consists of 4 sprints, 3 weeks each.

Semester-long software development project - from the Scrum Master's perspective: A student supports a development team during an entire semester long, taking on the role of the Scrum Master.

Example: The target is to provide a possibility for students to experience the servant leadership of Scrum. A senior student as a Scrum Master supports a team of 3 undergraduate students who need to develop a game software in one semester. The Scrum Master student holds the weekly Scrum and at the end of the 3-week sprints he/she prepares the team for the review, coordinates the retrospective and the planning.

Semester-long software development project - from different perspectives: A group of students work on a software development project for a full semester. Each student takes turns in the group as a developer and Scrum Master.

Example: The target is to provide students the possibility to experience both the developer and the Scrum Master roles. Students develop a small administration tool in teams of 3. The project is implemented in 3 sprints using an adaptation of Scrum methodology. During each sprint different students from the teams take on the roles of Scrum Master, Product Owner, and developer.

Monitoring of several software development teams - from the perspective of an agile coach: A student monitors the operation of several development teams and helps them with reflections, advice.

Example: The target is to give multiple insights about how Scrum teams work. Students in teams of 3 use Scrum to develop a software product. At the end of each 3 week-long sprint a senior student visits all teams one by one. He/she coordinates a Retrospective, where they reflect on their strengths and weaknesses. He/she supports them with reflections and advice addressing their impediments.

Situational exercises: Students simulate a situation as part of the lesson, inspired by the industrial use of the methodology, as part of a lesson. The goal is to understand and then elicit the correct operation of the methodology in a given situation.

Example: The target is to give the possibility to experience the course of a daily Scrum. We assign roles to 5 students from the class. Among the characters is a Scrum Master and 4 developers. The characters have different motivations and characteristics: The Scrum Master wants to validate Scrum values; developer 1 explains a lot and in detail; developer 2 is defocused because of a personal issue; developer 3 is ashamed to admit that he is stuck with his task; developer 4 likes to chat. The task is for students to simulate the daily Scrum of this team.

3.3. Methods for developing agile skills

Reflection: When a student looks back at a practical situation of which he or she was an observer or participant and analyzes it - alone or with the help of others. (Analysis can also be supported by sharing some criteria first.) He/she examines what was appropriate, what needs to be improved about the situation, and develops specific improvement ideas.

Example: The target is to learn how to communicate during a daily Scrum. After playing a situational exercise (see previous point), students look back at how they manifested themselves. If this is not considered appropriate, they get back into the situation and this time try to behave in an other, more appropriate way.

Schemas and examples: Presentation of manifestations that illustrate the proper functioning level of soft skills. If possible, sharing schemes that help to reach a similar level.

Example: The students work in a team of 3. One of the students does not appear at the meetings for more than a week, nor does he respond to messages. The team is insecure and frustrated. They

don't know how to solve this difficult situation correctly. The target is to learn how to communicate in these kind of difficult team situations. The instructor introduces the schematic operation of the Nonviolent Communication [4] (observations, feelings, needs, requests) in the class and then illustrates it with an example. It encourages students to use the scheme to write a message to the missing student.

Consultation: When the student, together with the instructor, analyzes a situation in order to learn how to behave in it. The student receives personalized recommendations from the instructor for self-improvement.

Example: A Scrum Master student reports that members of his team communicate timidly. The target is to teach the Scrum Master student how to support his team in gaining more courage. Advice listed: play icebreaker games, lay down common rules, give positive feedback, set the example of brave communication, reflect on what you see to the team (e.g. "I see you are timid"), etc.

3.4. Methods for teaching the values

Discussion: Students have to argue about what would be the appropriate behavior in a given hypothetical situation.

Example: The target is to help students understand the value of self-management. The instructor describes the following hypothetical situation to the students: A part of a team indicates to the Scrum Master that the division of tasks between them is not satisfying and one team member barely works. The Scrum Master takes control over the situation, distributes tasks proportionately among team members. The question is: Did the Scrum Master do the right thing? The instructor collects the pro and contra arguments and highlights the values behind them.

Reflection: When a student looks back at a practical situation of which he or she was an observer or participant and analyzes it - alone or with the help of others. (Analysis can also be supported by first sharing some criteria.) He/she examines what values have been manifested in that situation, what needs to be changed to manifest the agile values even more.

Example: Target is to teach Scrum values and detect their presence on Scrum event. Students work in teams of 3 on a semester-long project using Scrum. Following the first sprint, the instructor asks students to analyze their Scrum Retrospective in terms of how much the Scrum values (respect, focus, commitment, openness, courage) were present.

Consultation: When the student, together with the instructor, analyzes a situation in order to see how agile values were present during it. The student receives personalized recommendations from the instructor for improving the manifestation of agile values.

Example: Target is to teach that the presence or absence of Scrum values deeply effects teamwork. Students work in teams of 3 on a semester-long project using Scrum. The Scrum Master reports that one team member rarely communicates with other team members. He reports that this is despite the fact that he has created new communication channels. The instructor draws the attention of the Scrum Master to the fact that it is possible that the lack of communication is just a symptom. Maybe he has to deal with commitment, openness, courage issues. He should explore the deeper causes of missing communication through questions, listening, reflection.

4. Curricula for teaching agile software development

This chapter shows how the teaching methods discussed above can be incorporated and combined in 5 different courses. In each case, the course details, the weekly schedule, the way of implementing the teaching methods in the course and experiences related to the implementation, and students' feedback on the course are presented.

4.1. Software technology (practice)

Course details

Dedicated to: undergraduate students, fourth semester

Duration: 90 minutes / week

Type: mandatory

Goals: The aim of the course is to provide students with an experience of the entire project lifecycle and collaboration on developing a larger software project.

Developed aspects of agile: practice, skills

Weekly schedule:

Week	Action during class
1	Preparation
2	Weekly Scrum
3	Weekly Scrum
4	Review
5	Weekly Scrum
6	Weekly Scrum
7	Review
8	Weekly Scrum
9	Weekly Scrum
10	Review
11	Weekly Scrum
12	Weekly Scrum
13	Final Review

Teaching methods used:

Semester-long software development project - from the developer's perspective: Students must develop a strategic game software in teams of 3 in any programming language. At the beginning of the semester, they receive a short (15-minute) presentation on how the Scrum methodology works. Then they have to use an adaptation of Scrum. The semester is divided into four 3-week sprints. The role of the Product Owner is played by the instructor, the Scrum Master is a senior student, or also the instructor (in case we do not have sufficient senior student taking this role). Every 3 weeks the teams present the results of their sprint to the class. In the intermediate

weeks the teams do a weekly Scrum. During the sprints, students should work together in a self-organizing way. They need to distribute the tasks, share their impediments and work together to find solutions. After the Sprint Reviews, the teams have to hold a Retrospective at an extra meeting (not during class) with their Scrum Master, an Agile Coach, or on their own - with the help of a scheme. At the end of the semester, they can learn about the Scrum methodology in detail in a 90 minutes lecture and compare it with their experiences during teamwork.

Experiences: Experience has shown that students enjoyed teamwork. Most of the teams were enthusiastic all along, many definitely liked to create a game software. In the beginning it was harder for them to manage the tasks, nor did they understand the essence of their meetings. They kept reporting to the Scrum Master during daily Scrums, instead of cooperating. As the semester progressed, these problems disappeared in most teams. The most stubborn mistake was procrastination. Few teams managed to outgrow this.

Reflection: There was room for reflection during retrospectives. Retrospectives on the other hand, are not part of the class, they have to take place during separate, extra meetings.

Experiences: Reflection was taken more seriously in the teams, which had a Scrum Master student or an agile coach student. In this case improvement ideas were invented, some of them were proven to be useful. The other teams could use reflection questions in the form of a questionnaire, but we do not know if this was effective. This meeting gives opportunity to find ways to improve their soft skills and their understanding of Scrum values.

Students' feedback on the course: During 2021, 146 students participated in the course, of which 99 completed a feedback questionnaire at the end of the semester. 72% of the students considered the course absolutely useful, 20% mostly useful. Students rated different aspects of their team work on a scale of 1 to 5, where 5 means "absolutely". For the question "How successful have you been in realizing your own agile role: active member of development team?" the average of the scores was 4.31. Other averages: 4.65 - self-management of the team, 4.36 - motivation of the team, 4.64 - communication of the team, 4.74 - performance of the team, 4.68 - well-being in the team.

Students rated how much the course improved the four different aspects of agile software development. The average values were as follows: 4.35 - development of the theoretical aspect, 4.29 - development of the practical aspect, 4.92 - development of the skills, 4.29 - development of the attitude - all of which are considered high. It is striking that the degree of theoretic and attitudinal development is highly rated, although in this curriculum we did not stress this particularly. This may be caused by the fact that most of the students first encountered agility and Scrum at this point in their lives.

Some textual feedback were: "It was a great experience to work in a team.", "It was useful that I got acquainted with GitLab and CI tools.", "The most profitable was that I was introduced to agile software development.", "I gained insight into how software development works in an industrial environment."

4.2. Project management in IT (lecture)

Course details:

Dedicated to: graduate students, first semester

Duration: 90 minutes / week

Type: mandatory

Goals: The aim of the course is to make students understand the operation of their workplace environment, to work in project teams or lead them.

Developed aspects of agile: theory, values

Weekly schedule:

Week	Topic of the lecture
1	Basic concepts: project, project diamond
2	Corporate culture
3	Organizational models
4	Project life cycle 1.
5	Project life cycle 2.
6	Scrum methodology 1.
7	Scrum methodology 2.
8	Personality models
9	Customer reception at the highest level
10	Effective team
11	Leadership models
12	Change management
13	Burnout prevention

Teaching methods used:

Lecture: The Scrum methodology is presented on weeks 6 and 7. The following subtopics are taught with the help of a slide show: Waterfall model; Manifesto for Agile Software Development [1]; roles in Scrum, their responsibilities, advantageous skills for taking them, in the case of taking them; Scrum events, their typical course, typical mistakes; Scrum artifacts, their goals, properties, examples; Agile metrics; summary picture of Scrum operation; Scrum values; Agile leadership.

Experiences: Experience has shown that students have understood the basics of the Scrum methodology. On their exam they had to answer the simpler questions of a basic Scrum Master certification - everyone passed the exam.

Discussion: At certain points of the presentation of the Scrum theory some situations are thrown in to be discussed. These are: "We use the Waterfall model. What could be the disadvantage of this?"; "We use planning poker on our sprint planning. What could be the benefit of this?"; "Scrum values are strongly present in our team. How could this be manifested during our Scrum events?"; Students can share their opinions, arguments.

Experiences: Approx. 30% of the students got involved in the discussions and asked more questions. The discussions broke the monotony of the predominantly one-way communication, sparking attention. On the exams the questions concerning Scrum values were answered well by 70% of the students.

Students' feedback on the course:

During 2020, 77 students participated on the course, of which 33 completed the university's course feedback questionnaire. 91% of respondents gave maximal score on the structure of the course. 88% reported it was maximally helpful for their professional development. 91% stated the classes

were as interactive as possible. This can be considered a particularly high rating for a lecture, probably due to the discussions integrated into the classes. Some textual feedback were "Thank you for this course! It was a very good experience!", "It was the most enjoyable lecture of my studies so far."

4.3. Scrum Master training (seminar)

Course details:

Dedicated to: undergraduate and master students, any semester

Duration: 90 minutes / week

Type: elective

Goals: The aim of the course is to assure appropriate knowledge for taking the Scrum Master role and to develop the necessary skills and attitude.

Developed aspects of agile: theory, practice, skills, values

Weekly schedule:

Week	Project	Lesson	Other methods
1	Preparation	Introduction to Scrum	
2	Weekly Scrum	Scrum Roles; daily Scrum	Situation - A Daily Scrum, Reflection
3	Weekly Scrum	Scrum events; Retrospective techniques	Situation - Daily Scrum with Product Owner, Reflection, Schema – NVC [4]
4	Review, Retrospective, Planning		Reflection - First sprint
5	Weekly Scrum	Retrospective techniques, software tools	Reflection - First Retrospective, Schema - Retro
6	Weekly Scrum	Agile metrics	Situation - Scapegoating in the team, Reflection
7	Weekly Scrum	Scrum artifacts	Situation - A team split in two, Reflection
8	Review, Retrospective, Planning		Reflection - Second sprint
9	Weekly Scrum	Empiricism	Reflection - Second Reviews, Scheme - Review, Situation - Senior developer against agile, Reflection
10	Weekly Scrum	Scrum values, agile leadership	Situation - New Scrum Master, Reflection
11	Weekly Scrum	Scrum Master test	
12	Final Review, Final Retrospective		Reflection - Third sprint
13		Evaluation	Consultation - Final evaluation

Teaching methods used:

Lecture: Most of the classes include an approx. 20-minute lecture.

Experiences: With this solution, students learned the basics of Scrum during the semester, while time remained to apply other teaching methods as well.

Reading literature: It is obligatory to read The Scrum Guide [2].

Experiences: The end-of-semester test showed that students have known its content but knew better the theoretical parts that were mentioned during classes.

Semester-long software development project - from different perspectives: During the semester students work in teams of 3 on an administration software. The role of the Product Owner is performed by a senior student or the instructor. All students try out both the Scrum Master and developer roles. There is a total of 3 sprints during the semester. On the closing class of the sprints, students have 30 minutes per team for a review, retrospective, and planning.

Experiences: Experience has shown that during the semester students became more and more aware of the course and purpose of each Scrum event and implemented them more and more appropriately. According to their feedback they failed to truly gain experience in the role of the Scrum Master, which they practiced only for a sprint. This may also have been caused by the fact that the teams were small (3 people, including the Scrum Master). The 30 minutes allocated for a team's review-retrospective-planning triple usually proved to be too short. They would have liked to try out more retrospective techniques. By the end of the semester the products were successfully completed, but some students stated that it would not have been necessary to have such a large amount of software-development to understand how Scrum works. Moreover, perhaps a non-software development simulation project would have been enough.

Situational exercises: There are 6 situational exercises during the semester. To prepare and play them, approx. 10 minutes are needed. Students receive their own role descriptions and details of the situation via email. They don't know each other's role descriptions. Situational exercises become more and more difficult, requiring more and more theoretical knowledge, advanced skills, developed attitude. Each time someone else plays the role of the Scrum Master, and those who were not given an active role receive observational aspects. The daily Scrum situational exercise simulates an average daily Scrum of an average team that has talkative, quiet, defocused, etc. members. The second situational exercise is also a daily Scrum, but with the presence of the Product Owner. The role of the Product Owner is to distract the team by telling stories from the management. In the third exercise the point is that several members of the team blame a single team member for the failure of the sprint. The question is how the Scrum Master handles this. In the case of the "A team split in two" exercise, one issue has to be decided. Half of the team would like to choose option A, the other half option B. The Scrum Master has to deal with this situation. In the fifth exercise the topic is the resistance of the team's most experienced senior developer against teamwork. The challenge is to find a way to deal with his resistance. In the last exercise the group simulates the first meeting of a new Scrum Master and his team. There are also skeptical and optimistic members of the team. The task of the Scrum Master is to start a good cooperation with his new team.

Experiences: The experience was that students got involved in their roles - although they had to implement it online because of Covid-19. As the semester progressed, the Scrum Masters were more able to handle different characters (e.g. quiet, talkative, etc.). The students liked the situational exercises, found them useful, even more of them would have been welcome.

Reflection: Reflection is present several times during the course. Primarily, situational exercises are always followed by reflection. Students who were given observer roles are the first who reflect on the situations. After that each actor is able to tell what they experienced.

Experiences: The time allocated for reflection was at least 10 minutes, but it always turned out to be too short. Soft skills and Scrum values were in the focus of the reflections and students formulated ideas to develop this aspect of Scrum.

Secondly, the Scrum retrospectives are a reflection on each sprint. The teams look back, analyze the sprint along aspects chosen by the Scrum Masters and search for ways to optimize their processes.

Experiences: The development in operation of the teams was observable during the semester.

After closing the sprints, there is a reflection on the quality of the reviews and retrospectives. The goal is to learn how this can be lead on a high quality.

Experiences: During the semester, sprint ceremonies got better and better - more focused, more concise, smooth.

Schemas and examples: After seeing that the first sprint reviews and retrospectives were not satisfactorily effective, schemas of a retrospective and a review were presented.

Experiences: After the presentation of these schemas, these Scrum events have become smoother and more professional.

Consultation:

Experiences: A Scrum Master turned to advice with the problem that one of his team members does not work, neither responds to messages. The team did not know if they can count on him, which makes the team insecure. The Scrum Master was introduced to the method of NVC [4] and was given an example of how he could communicate with the problematic team member. The Scrum Master tried out NVC through posting on message board and the missing student responded. The problematic student officially and peacefully left the team, which helped the team to deal with the situation. At the end of semester, students received feedback one by one on their strengths and weaknesses in order to have the possibility to become good Scrum Masters. The students were very grateful for the personal feedback.

Students' feedback on the course: During 2021, 9 students participated in the course, of which 7 completed a feedback questionnaire at the end of the semester. 28% of the students (2 students) were completely satisfied with the training, 72% (5 students) were mostly satisfied. 42% (3 students) found the situation exercises absolutely useful, 58% (4 students) mostly useful. According to their votes, situational exercises contributed mostly understand Scrum roles and their responsibilities (6 votes), team dynamics (6 votes), handling problems (5 votes) and Scrum values (5 votes each). In textual feedback, several students indicated that they really liked that the exercises were based on real situations. Situational exercises were mentioned as the best part of the course by 72% (5 students). However, project work was not popular within this course. 58% (4 students) indicated that the work was too much, 42% (3 students) stated that change is needed regarding project works in future semesters. On the question "How useful did you find the project work?", the following scores were received: 28% (2 students) - a little, 14% (1 student) - fairly, 42% (3 students) - mostly, 14% (1 student) - absolutely.

Students rated how much the course improved the four different aspects of agility on a scale of 1 to 5, where 5 means "absolutely". The average values were as follows: 4.57 - development of the theoretical aspect, 4.71 - development of the practical aspect, 4.28 - development of the skills, 4.57 - development of the attitude - all of which are considered high. A textual feedback: "I really enjoyed the course. It is good that we can acquire such up-to-date knowledge at university. Keep it up! Lots more! :)"

4.4. Software development in practice (practice)

Course details:

Dedicated to: master students, second semester

Duration: 90 minutes / week

Type: optional

Goals: The aim of the course is to simulate an industrial software development situation. In 2021, each student played the role of the Scrum Master in a semester-long software development project, realized by a team of 3 undergraduate students.

Develops the following aspects of agile: theory, practice, skills, values

Weekly schedule:

Week	Action during class	Action during the week
1	Theoretical preparation	Meeting the Product Owner
2	Theoretical preparation	Team building, Planning
3	Consultation - Subgroup A	Weekly Scrum, preparation to Review
4	Consultation - Subgroup B	Review, Retrospective, Planning
5	Consultation - Subgroup A	Weekly Scrum
6	Consultation - Subgroup B	Weekly Scrum, preparation to Review
7	Consultation - Subgroup A	Review, Retrospective, Planning
8	Consultation - Subgroup B	Weekly Scrum
9	Consultation - Subgroup A	Weekly Scrum, preparation to Review
10	Consultation - Subgroup B	Review, Retrospective, Planning
11	Consultation - Subgroup A	Weekly Scrum
12	Consultation - Subgroup B	Weekly Scrum, preparation to Review
13	Closing	Final Review, Final Retrospective

Teaching methods used:

Semester-long software development project - from the Scrum Master's perspective: Every student of the Software development in practice course is the Scrum Master of a team from the Software technology course. During the semester, they have to support the team, participate in their daily Scrums, Reviews, coordinate Retrospectives and Plannings a total of 4 times.

Experiences: The students liked the Scrum Master role and got more and more involved. Because Retrospective and Planning did not fit in the timeframe of the classes (there was no time left after the Reviews), it was often difficult to schedule a separate meeting for them. There were about 4-5 Scrum Master students who had greater difficulties with their team, e.g. lazy team member, split team, misunderstandings in the team. Typically, the Scrum Masters hoped the situation would resolve on its own and found it difficult to admit that there is something wrong. At the same time about 20 Scrum teams worked well, the Scrum Masters reported better and better communication, cooperation, focused meetings and successes. About 3 Scrum Masters have tried out delegation by the end of semester and held creative retrospectives.

Lecture: The semester begins with two 90-minute lectures, which briefly presents Scrum events, roles, products, Scrum values, Agile leadership. During the presentation of the theory, we also discuss exactly how to implement them in the particular case of the course.

Experiences: After the two preparatory classes, the students all passed a theory test and were able to begin their work with their teams. Later, only a few questions arose regarding Scrum's theory.

Reading literature: The Scrum Guide [2] is a must read at the beginning of the semester. There are questions in the test that can be answered only by reading the document.

Experiences: The students were less likely to answer the questions, but still everyone passed the test.

Discussion: The two preparatory lectures at the beginning of the semester are enriched by small discussion questions. For example: "Our team complains that they do not understand what the job is. We, as Scrum Masters, contact the Product Owner, ask him what the job is and then explain it to the team. Is this a good approach?", "A team member is missing from a meeting. We are not worried, she will definitely come next week. Is this a good attitude?" etc.

Experiences: The students got involved in the discussions, made several arguments, and the solution always tipped in the right direction.

Schemas and examples: During the theoretical preparation, students receive schemas regarding situational leadership model, how to ask questions, how to give feedback, Scrum values as compass in decisions.

Experiences: Later, during the discussions, it was observable that the students thought along the Scrum values. Some Scrum Masters reported that they used the feedback giving scheme and found that it was a success.

Reflection: After the theoretical preparation, during each class a subgroup of the students reports on what they experience in their teams. Sometimes they do it based on reflection aspects, such as "What is the biggest weakness of your team?", "How has your team developed till now?" etc.

Experiences: Students thought over the aspects and sometimes formulated things that were new to themselves. Often, they articulated aloud what they would do differently next, and later reported that they had tried and succeeded.

Consultation: During classes, the instructor can provide personalized advice to the student.

Experiences: For example, one Scrum Master reported that his team is very enthusiastic but very dependent on him, always waiting for him to decide, organize etc. He was advised to try to delegate a daily Scrum. The student later reported that he had tried it and it was a good experience for the whole team that they were capable of self-management. In another case, a Scrum Master repeatedly reported that he is frustrated because his team is quiet, restrained. He was advised to try to find out what is behind this, e.g. lack of courage, openness or commitment, or it is simply the dynamics of his team. Later, the Scrum Master reported that he realized Scrum teams can be different, even quiet, restrained, and that does not necessarily mean there is something wrong. Overall, it was obvious that students are happy to receive personalized advice and find them helpful. Moreover, those who only witness the consultation, also learn from the other Scrum Master's situation.

Students' feedback on the course: During 2021, 25 students participated in the course, of which 24 completed a feedback questionnaire at the end of the semester. 66% of the students considered the course absolutely useful, 25% mostly useful. Students rated how much they could be helpful in their teams on a scale of 1 to 5, where 5 means "absolutely". The average values were as follows: 3.91 - helpful in general, 3.58 - helpful in planning, 3.70 - helpful in preparing reviews, 4.29 - helpful

on retrospectives, 4.62 - helpful in weekly Scrums, 4.45 - helpful in clearing impediments, 4.08 - helpful in constantly developing team processes. For the question "How successful have you been in realizing your own agile role: a leader who supports and serves a self-organizing team?" the average of the scores was 4.29.

Students rated how much the course improved the four different aspects of agile software development. The average values were as follows: 4.66 - development of the theoretical aspect, 4.12 - development of the practical aspect, 3.92 - development of the skills, 4.62 - development of the attitude - all of which are considered high. Some textual feedback were: "The course gave a good chance to try the theory in practice.", "Now I see the Scrum Master's point of view", "I learned the Scrum Master does play an important role.", "I learned it is not easy to be a Scrum Master."

4.5. Agile Research Team (research project)

Course details:

Dedicated to: master students, all semesters

Duration: 90 minutes consultation / week

Type: elective

Goals: The aim of the Agile Research Team is to study research results on the topic of agile methodologies, practices, principles and carry out new experiments.

Developed aspects of agile: theory, practice, skills, values

Research topics:

Topic	Students	Since
Developing methods for supporting the Scrum Master training	2	1, 5 year
Developing methods for supporting effective Scrum Retrospectives in education	1	1, 5 year
Analyzing a new hybrid agile methodology	1	1, 5 year
Analyzing the impact of agile transformation on software developers	1	1 year

Teaching methods used:

Reading literature: When someone joins the research team, he/she definitely has to read some important literature. These include the Manifesto for Agile Software Development [1] and the Scrum guide [2].

Experiences: Experiences had shown that this has built a common understanding of basic concepts and later everyone could join the discussions.

Consultation: Each student chooses a research goal after enrollment. Our weekly meetings are actually joint consultations in which everyone reports on what they have achieved since the last meeting, what impediments they have met, what plans they have until the next meeting. The instructor and also other students give advice.

Experiences: With this method the team has reached to publish 4 articles in conference proceedings and 5 dissertations are going to be presented this semester.

Discussion: Sometimes, there are spontaneous discussions on meetings.

Experiences: Students argue with each other about what questions or answers are relevant to a particular Scrum-related questionnaire that some of them creates, for example. Based on this, more nuanced questionnaires were created than those originally compiled by a single student.

Literature review: Some of the research topics are literature focused.

Experiences: The student with the topic "Analyzing the impact of agile transformation on software developers" firstly explored the causes, course etc. of the agile transformation from literature. Based on this, he was able to create a detailed, exciting questionnaire for his research. The student dealing with topic "Analyzing a new hybrid agile methodology" also started with the collection of literature on hybrid methodologies, and then was able to describe a completely new hybrid agile methodology he invented and used before.

Monitoring of several software development teams - from the perspective of an agile coach: The student with the topic "Developing methods for supporting effective Scrum Retrospectives in education" visited a few Software technology teams at the end of each sprint.

Experiences: She tried out different retrospective techniques with them. She gave tips on correcting their weaknesses, based on the literature and her own experiences – which grow from sprint to sprint. She was able to build an entire collection of tips on handling various typical team weaknesses, which she also incorporated into one application. This app aims to automate Scrum Retrospectives. It uses a form to assess the teams situation and then gives tips based on it.

Reflection: Two students with the topic "Developing methods for supporting the Scrum Master training" attended the training as Product Owners. In addition, they have witnessed all the situational exercises. After the lessons, we always reflected separately on what they had seen and learned from the situation, how it could be done differently.

Experiences: The two students came to deeper and deeper understanding of Scrum and were able to perform the role of the Product Owner quite well.

Students' feedback on the course: Students reported on how they benefited from being a member of the Agile Research Team: "I was able to study closely the topic of agile values, which basically lacks literature.", "I saw agile software development from many perspectives by following the research of my colleagues.", "We gave each other good ideas. We were a small creative community, and we were catalyzing each other's work."

5. Discussion

As the curricula and experiences described above demonstrate, the four aspects of agile software development can be taught at university.

Teaching of the theory of agile methodologies could be incorporated in project management related lectures, as in the case of our Project Management in IT course. Such lectures make possible to compare agile methodologies with other methodologies, which leads to a better understanding of them. Teaching of theory can be well complemented with teaching of values by starting discussions during lecture. Discussions help deepen values as students come up with supporting arguments. The discussions also break the monotony of the lectures.

The practical aspect of an agile methodology can be taught on a practice course, by using it for managing software engineering projects of teams of students. There are several examples of this teaching approach in the literature, according to [5] this is the most common form of teaching agile methodologies. Cases differ mostly in assigning the roles. Product Owner role is commonly taken by an instructor or other expert, such as in cases [6,7,8,9,10], but there are also examples of Product Owner students [12,13]. In the case of the Scrum Master role, the opposite is more typical, so mostly students fill the Scrum Master role [6,7,9,11,12]. However, there are also examples where the instructor fills the Scrum Master role [8,13]. If an instructor fills a Scrum role, it is usually advantageous, because he/she fills it more authentically due to his/her experience. If a student fills a Scrum role, experience is not guaranteed, but in return we give the student the opportunity to gain experience.

In our case, assigning the Product Owner role to the instructor (in case of each course presented in this paper) seemed to be a good solution, as the evaluation of the products at the end of the semester was also his task. This way he could authentically represent the product requirements. On the Software Technology course, students played only the role of developers, and the Scrum Master role was played by senior students from the Software engineering in practice course. We consider this a good solution because – being their first teamwork at the university – undergraduate students needed help in managing teamwork. Assigning the Scrum Master role to senior students was an efficient solution, because in this way the teams have got more attention and senior students gained more experience in Scrum. Student teams who had a senior student as their Scrum Master assigned an average value of 4.65 on how much the Scrum Master student helped their work. Senior students who were able to try themselves in the role of the Scrum Master gave an average rate of 4.60 on how useful this opportunity was for them. In addition, they felt that they could help their team in an average of 3.91. The three high values further validate the success of connecting the Software technology and the Software development in practice course. However, we had to be aware that some Scrum Master students tend to wait too long for their teams to solve problems among themselves, others may intervene too soon and not support self-sufficiency enough, as mentioned in [12]. We supported the operation of student Scrum Masters with providing weekly consultations with an instructor, in line with recommendation form [5]: "one of the instructors should still play the supervisor Scrum Master for the whole process". We considered the connection of the two courses very successful because we have very few human resources at our university and several practical courses. [12] presents a successful method of teaching the practice of Scrum that they claim need a large staff (up to 11 supervisors). We managed to achieve something similarly effective by connecting the two courses, and even saved significant resources in terms of university staff.

According to the literature, another frequent way to teach the practice of agile methodologies is to play educational games. Some of these are presented in [14,15,16,17]. There has been no example of this at our university yet, but in the case of the Scrum Master course, we should seriously consider this option. The feedback was the students were the least convinced by the efficiency of the semester-long project among all the methods used on this course. They considered the effort needed too high compared to the lessons learned. [5,18] point out that Scrum games are suitable for demonstrating the practical operation of Scrum in short time. Due to little time required, we could consider making a short Scrum simulation even on the Project Management in IT lecture in some creative way, e.g. paper city construction [19]. This game is cheap compared to Lego based Scrum games [14].

The basic theory and practice of Scrum can and should be taught on mandatory courses because of their popularity. However, we should not forget about teaching agile skills and values too. Our

experience has shown that if we want to teach in depth and strengthen all four aspects then optional or elective courses are more suitable.

The importance of teaching agile skills and values are discussed in multiple articles of Martin Kropp and Andreas Meier [7,18,19]. In their Agile Competence Pyramid model, Agile Values are placed at the highest level of being agile [7]. They defined three principles for teaching agile comprehensively: a.) make personal experience by working in an agile production environment, b.) cooperate in an agile group with a substantial amount of social exchange and discourse taking place, c.) develop and discuss agile values and attitudes [18]. We believe keeping these three principles in mind while designing a course is in line with developing all four aspects of agile methodologies.

In our case, the Scrum Master training and the Software engineering in practice integrated all three principles through semester-long project works, reflection, discussions. Besides, it developed all four aspects of agile methodologies. On both courses, the same theoretical material was presented, and the same literature was obligatory to read. The difference is that in case of the Software development in practice course, the knowledge was passed on in form of two 90-minute lectures, while the Scrum Master training was divided into short, 20-minute lectures. Both solutions proved to offer a very good understanding of Scrum theory (averages higher than 4.50). As part of the Scrum Master training, 6 situational exercises and projects of 3 ensured that students become familiar with the practice of Scrum. Students found the situational exercises very useful, but the project-work less so. They said the teams were too small and no real lessons were drawn. The time invested into projects was also too much. In comparison, we received a lot of positive feedback on the method where students played only Scrum Master roles of project teams of another course – like in case of the Software development in practice course. In addition, there were many opportunities for consultation, reflection and discussion. Students found this solution very useful. In the case of Scrum Master course, students could observe 6 typical industrial Scrum situations closely through situational exercises. In the case of the Software development in practice course, students could observe only one team's functioning closely - but this was more real, not a simulation. Both agile practice experience and soft skill development were rated to be more noticeable in the case of the Scrum Master course (4.71 - practice, 4.28 - skills), and they received high ratings in the case of the Software development in practice course as well (4.12 - practice, 3.92 - skills). We think the difference is caused by the use of situational exercises in the Scrum Master course, which have put students in difficult real-life situation, thus developing their knowledge of real-life Scrum effectively. Agile values development was rated in a similar way: 4.62 in the case of the Software development in practice course, 4.57 in the case of the Scrum Master training course. The advantage in favor of the Software development in practice course could be due to more time allocated for discussions and consultations.

Based on the similarly high rates, we consider both the Scrum Master training and the Software development in practice course adequate for teaching all four aspects of agile software development. There were two students who participated on both courses. They agreed the two courses were good one by one, and they strengthened each other as well. A textual feedback was: "I think it is worth completing several courses, especially Scrum Master training and Software engineering in practice, because together they give a much more comprehensive picture of Scrum."

Thus, we consider agile methodologies could be taught most widely on elective and optional courses, the research team is more suitable for teaching in depth. In case of Agile Research Team, each student has his own research topic. We can choose different methods of helping them based on their topic, e.g. reviewing, assigning the role of an agile coach or Product Owner. Every student can be supported by consultation, which also supports all other team members in learning about agile at the same time.

Courses teaching various aspects are best placed in such a way that, proceeding from undergraduate studies to graduate studies, previously taught material is always supplemented in depth or variety. In our case students meet agile software development first in the Software technology mandatory course. Here they gain basic knowledge about Scrum roles, events, artifacts, and experience being a developer in a Scrum team. Secondly, they deepen their theoretical knowledge on agile software development in the Project management in IT course, which is a mandatory course for first year graduate students. This course is enriched with discussions to give insight into agile values. If students get interested in Scrum by these obligatory courses, we provide them optional or elective courses to expand their knowledge. The Scrum Master training and the Software engineering in practice courses both focus on developing skills and values of agile software development. Besides that, they highlight the perspective of the Scrum Master. If a student wants to gain even more knowledge on agile software development, he/she can join the Agile Research Team. This allows students to thoroughly study what the most inspiring part of agile software development is for them.

As for some of the curricula and conclusions of this article, it is important to note that we can consider them validated only in a modest way due to low number of feedback. For example, in case of the Scrum master training only 7 people filled out our feedback survey.

We considered heavy focus on developing skills and attitudes the essence of Scrum Master training, which is most effective on small courses. However, because of the small number of participants, it is very difficult to collect an amount of data that can be considered sufficient to validate the curriculum. The current version of the Scrum Master training presented in this article was attended by 9 people and 7 people provided feedback on the course. In terms of validation, however, it may be interesting that a previous version of this course had been attended by a total of 2x9 students in previous years, and the feedback pointed in a similar direction. The course differed in that there was only one situational exercise throughout the semester, and slightly more theoretical material was delivered in the form of a lecture. 16 students provided feedback on the previous version of the course. The averages of the evaluations were as follows (notation: aspect {c = current course average, o = old course average}) satisfaction with the course {c = 4.28; o = 4.12}; usefulness of project work {c = 3.42; o = 3.50}; usefulness of situational exercises {c = 4.58; o = 4.60}. In the text-based feedback, 42% (3 students) mentioned that they would change the project work of the current course, and 37.5% (6 people) in the case of the old one. Thus, it can be seen that although we have a small number of measurements, they point in one the direction, pointing out the weak (project work) and strong points (situational exercises) of the curricula.

Another interesting observation is that the satisfaction related to the courses may have been influenced by students' prior knowledge of Scrum and agility.

There seems to be a tendency for students who had practical experience on Scrum to vote lower values for several aspects of the courses. For example, in the case of the Scrum master course (current (2021) and old one (2020) as well), we asked students what experience they had with Scrum before the course. We call beginners those who knew something about Scrum theory at most and advanced ones who have had some practical experience with Scrum before. Roughly half of students were beginner, half were advanced. The averages were as follows (notation: aspect {b = average of beginners, ad = average of advanced}) satisfaction with the course {b = 4.63; ad = 4.00}; usefulness of project work {b = 3.91; ad = 3.16}; development of the theoretical aspect {b = 4.58; ad = 4.26}; development of the practical aspect {b = 4.65; ad = 4.19}; development of the skills {b = 4.58; ad = 3.87}; development of the attitude {b = 4.20; ad = 3.66}. The biggest difference is in the assessment of the usefulness of the project work (0.75), but there is also a big difference in the overall satisfaction with the course (0.65). In the case of Software technology

course, we can also see that development along different aspects was rated by the students very high (at least 4.29 in all cases), although this course did not involve many different methods. Thus, it emerges that the curricula presented in this article are mostly successful for beginners, and the methods defined in the article also develop beginners the most.

Since the experiments took place in 2020 and 2021, we cannot miss the question of how the Covid 19 might have influenced it - although this is not the focus of this article. At our university, from mid-March 2020 to present, the education took place only online (because of the Covid 19 restrictions). We used MS Teams and realized live online teaching on all courses presented.

We can give a measurement-based estimate on the effect of Covid 19 only on the courses whose structure have not changed since 2019. This holds only for the Project management in IT lecture. In 2020, for the first time, the lecture took place online, in the form of an MS Teams meeting. If we look at the data, we can see that students rated the online course a bit higher than its offline version: (notation: aspect {on = online course average; of = offline course average}) structure of the course {on = 4.88; of = 4.50}; professional development by the course {on =4.81; of = 4.15}; interactivity on the course {on = 4.81; of = 4.55}. The author's own conclusion was that in an online environment, students appreciated even more if one lecture was interactive. Some students preferred to write in a joint chat rather than comment on the discussions in a video voice call. Slightly fewer people took part in the discussions, but they were still involved. Students also said they were able to learn more through a course because they could replay the lesson recordings. For the other courses, we can only rely on the verbal feedback of the students or the information provided on the Sprint Reviews. Some students, especially in the case of teamwork, missed face-to-face meetings. At our call, several students organized weekly 1-3 hour long online coding sessions together and they admitted it helped them a lot. Short, daily meeting - like weekly Scrums - were considered by many to be logistically easier to organize online.

Thus, the shift to online teaching caused by Covid 19 at first seemed to be a challenge. But after all, our experience was that by finding the right online tools, we were able to achieve the same performance as before on the courses presented.

Overall, we discussed how agility can be taught at university. The uniqueness of this article compared to the others is that it first defines separate methods to teach the four aspects and then shows how they can be incorporated into different curricula or an entire university program. Compared to other articles, which usually present the implementation of a single curriculum, this article presents many methods and their embedding and combinability in different curricula. Based on the measurements related to the implemented curricula, the methods appear to be successful. However, their success is recommended to be confirmed by further measurements.

6. Conclusion

Teaching of the four aspects of agile software development can be realized in classic university courses,

Teaching of theory can be integrated into many types of courses. Discussions can complement teaching theory and thus ensure understanding of agile values. The practice of agile software development can most effectively be taught by semester-long projects and situational exercises, in practices and seminars. Reflections on practical situations and consultations can amplify skills development. Choosing an inspiring topic related to agile software development and working in an agile research team can be beneficial in various ways.

Incorporating teaching of agile software development into multiple types of courses in various ways is a winning strategy because it gives students opportunity to master agile software development.

In the introduction of this article, we defined three questions:

1. Can different aspects of agile software development (theory, practice, skills, values) be taught at university?
2. What kinds of methods can be used to teach the aspects of agile software development?
3. In what kind of curriculum can these methods be integrated?

In response, we defined 14 methods for developing the four different aspects, with at least 3 methods for each aspect. In the field of practice development, these are in line with those prevalent in the literature. The methods were inserted into 5 different curricula, which we implemented. We checked their effectiveness with a total of more than 130 students. The results showed that the students found that they had developed along the four different aspects on the courses. In the case of every aspect at least an average of 70% development were measured. These achievements provide a positive answer to our question that agile methodologies can be taught at university. The number of involved students is moderated, so it is recommended to collect additional responses, possibly not based on self-assessment. However, in the field of teaching agility, measurement methods are a challenge in other research as well.

7. Acknowledgment

The research project was supported by the European Union and co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

Bibliography

1. *Manifesto for Agile Software Development* (2001) https://agilemanifesto.org/
2. K. Schwaber, J. Sutherland: *The Scrum Guide* (2020) https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf
3. *14th Annual State of Agile Report* (2020) https://stateofagile.com
4. M. Rosenberg: *Nonviolent Communication*. Puddle Dancer Press, Encitas (2015)
5. V. Mahnič: *Scrum in software engineering courses: an outline of the literature*. Global Journal of Engineering Education, 17 (2015) 2. World Institute for Engineering and Technology Education (2015) 77-83
6. D. Damian, C. Lassenius, M. Paasivaara, A. Borici, A. Schröter: *Teaching a globally distributed project course using Scrum practices*. CTGDSD 2012, Zurich (2012) 30-34 DOI: 10.1109/CTGDSD.2012.6226947

7. M. Kropp, A. Meier: *Teaching agile software development at university level: values, management, and craftsmanship*. CSEE&T 2013, San Francisco (2013) 179-188 DOI: 10.1109/CSEET.2013.6595249

8. V. Mahnič: *A capstone course on agile software development using Scrum*. IEEE Transactions on Education, 55 (2012) 1. IEEE (2012) 99-106

9. M. Paasivaara, C. Lassenius, D. Damian, P. Raty, A. Schröter: *Teaching students global software engineering skills using distributed Scrum*. ICSE 2013, San Francisco (2013) 1128-1137 DOI: 10.1109/ICSE.2013.6606664

10. S.D. Zorzo, L. De Ponte, D. Lucrédio: *Using scrum to teach software engineering: a case study*. FIE 2013, Oklahoma City (2013) 455-461 DOI: 10.1109/FIE.2013.6684866

11. G. Rodríguez, A. Soria, M. Campo: *Teaching Scrum to software engineering students with virtual reality support*. Lecture Notes in Computer Science 7547 (2012) 140-159

12. A. Scharf, A. Koch: *Scrum in a software engineering course: an in-depth praxis report*. CSEE&T 2013, San Francisco (2013) 159-168 DOI: 10.1109/CSEET.2013.6595247

13. T. Reichlmayr: *Working towards the student Scrum - developing agile Android applications*. ASEE Annual Conference & Exposition 2011, Vancouver (2011) 159-168

14. M. Paasivaara, C. Lassenius, V. Heikkilä, T. Toivola: *Teaching students Scrum using LEGO blocks*. ICSE 2014, Hyderabad (2014) 382-391 DOI: 10.1145/2591062.2591169

15. C.G.Von Wangenheim, A.F. Borgatto: *SCRUMIA - an educational game for teaching SCRUM incomputing courses*. Journal of Systems and Software, 86 (2013) 10. Elsevier (2013) 2675-2687

16. S. Ramingwong, L. Ramingwong: *Plasticine Scrum: an alternative solution for simulating Scrum software development*. Lecture Notes in Electrical Engineering, 339 (2015) 851-858

17. J. M. Fernandes, S.M. Sousa: *PlayScrum - a card game to learn the Scrum agile method*. VS-GAMES 2010, Braga (2010) 52-59 DOI 10.1109/VS-GAMES16805.2010

18. M. Kropp, A. Meier, M. Mateescu, C. Zahn: *Teaching and learning agile collaboration*. CSEE&T 2014, Klagenfurt (2014) 139-148 DOI: 10.1109/CSEET.2014.6816791

19. S. Hof, M. Kropp, M. Landolt: *Use of gamification to teach agile values and collaboration: A multi-week scrum simulation project in an undergraduate software engineering course*. ITiCSE 2017, Bologna (2017) 323-328 DOI: 10.1145/3059009.3059043

Authors

ILYÉS Enikő

Eötvös Loránd University, Faculty of Informatics, Department of Software Technology and Methodology, Hungary, e-mail: ilyese@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 3, Number 2. 2021

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.3.2.2378

License

SWOT Assessment Usage in School Talent Management

SARMASÁGI Pál

Abstract. Talent management is important part of human resource management in business world. There are two frequently used tools that supports talent management. DISC assessment is the leading personality assessment tool in the business world that helps improve work efficiency, team work, and workplace communication. SWOT analysis is a popular marketing tool that often use as personal develop tool. These can be particularly useful in talent development at secondary school. This research presents a possible application of the SWOT tools in secondary education. Beside the talent management SWOT can also use in formative assessment.

Keywords: SWOT analysis, formative assessment, talent management, digital competences

1. Introduction

The talent management in the world of work is more productive than in education, so the application of the methods used in this field in industry can be especially useful at schools. Those methods can be used in public education that are commonly used, tried, and useful in business and industry.

The Human Resource department of bigger companies regularly select the talented employees and train them as a potential manager. After the selection the employees' skills are evaluated by the common used assessments. The results of these assessments point to what kind of skills should be developed. The most frequently applied tests are DISC and SWOT assessments in the business world. While DISC investigates the personality of people, SWOT focuses on the skills and potential of the tested persons. DISC assessment helps to find the proper position while SWOT highlights the strengths and weaknesses of the tested employee. Based on the results of these tests the specialists prepare a training and self-training plan to develop the potential manager.

School talent management requires a similar approach so it was examined how teachers can use these assessments in public education. This paper presents some sample usages of SWOT and the experiences of these assessments' usage at secondary school.

2. Concept and definition of SWOT analysis

SWOT analysis is well-known within the Hungarian public education system as well as in the business world. It mainly used for evaluation of curriculums or schools in education. The name SWOT is a mosaic word composed of the initials of Strengths, Weaknesses, Opportunities and Threats.

There is a debate between historians of science regarding the origin of SWOT. According to one concept, it was developed by George Albert Smith Jr. and C. Roland Christensen at Harvard University in the 1950s. Others state SWOT came from Albert Humphrey who worked at Stanford University since he was the first who published about SWOT. The Harvardians' work was continued by Kenneth Andrews and based on his lectures at the Harvard Business School SWOT became a known marketing tool. [1] [2]

The aim of the analysis is to position a product properly in the market. Strengths and weaknesses are internal factors that refer to the physical and financial characteristics of the product and the human resources behind it. Opportunities and threats from external factors such as economic and

market trends. At the end of the analysis, a proposal and a goal must be formulated, and the strategy necessary to achieve the goal must be found.

SWOT was first used in the 1960s, and the method has since proven much, so it has been used in more and more areas. The employee is the product that needs to be able to sell on the labour-market. So the human resource management staff, the company trainers, coaches and the head-hunters started to apply SWOT to employees as individuals as well. The result is an individual marketing plan that outlines the strategy of a personal development plan. [3][4]

3 Literature review

SWOT has a wide literature in marketing and human resource management, while there are only few papers regarding usage in education. The most of education related SWOT paper focuses on curriculum and institution evaluation. There is an interesting study from Thailand which investigated medical students by SWOT analysis. It searched to answer the question: Which students will become a dedicated well-educated doctor? The primary goal of the research was to explore the threats that result in students not being placed in a medical field after their graduating. Knowing the results, the curriculum was modified so that as many students as possible would take up the medical profession. [4][5]

4 Business tool at school

Pedagogy uses the achievements of psychology in many areas just like the human resource management and marketing in the world of work uses. Psychological tools are particularly useful when teachers work with non-average pupils especially in the case of talents. It is a well-known fact that the talents are usually 'odd', they require special communication and special care. Some are loud and disturb lessons, while others are retreating and quiet and they don't express their knowledge. Teachers' task to explore the hidden skills of pupils and teach them to utilize these skills. The presented assessments can help to implement these expectations, and these are well-known and proven in the world of work and a lot of experiences are available regarding their usage. The next trials show an example, a possible way of how can we use the demonstrated marketing tool within the public education.

4.1 Evaluation at school

Formative evaluation is still a novel method although it has been in the Hungarian literature for 20 years. Nowadays, of course, it is one of the most popular topics in both conferences and papers, despite these it is not used generally in practice.

The need for formative assessment is most often justified by continuous feedback and the importance of properly informing the learner. It allows individual students to better manage their time, their energy for learning, while it helps teachers recognize which topics need to be addressed more or to teach with new metaphors for better understanding. [6]

The most frequently used elements of formative assessment are the homework assignment, presentation, standardized tests, quizzes, oral question, or draft work. The ICT tools are particularly useful in the computer science subject and are being used by more and more teachers to support formative assessment (e.g. Kahoot, Redmenta, Learningapps.com, etc.) [7]

There is another aspect along the lines of formative and diagnostic evaluation. In addition to continuous information, it is important to involve pupils in the assessment. The evaluation builds on comparison and mathematical relations that improves pupils' mathematical competences as well as it also develops the ability to self-assess of students.

The importance of formative assessments is strengthened in the case of school talent management. The venue of talent management is usually study groups as well as specialization classes where the small groups provide a possibility to discuss evaluation aspects between the teacher and the pupils.

4.2 SWOT analysis as a tool for formative assessment

The classic SWOT analysis explores the internal and external factors of the given product. SWOT can be applied on workers in human resource management and career counselling to explore their strengths and weaknesses as they are 'sales products' on the labour market. Generally, internal factors are mainly the relevant skills from the point of view of a given job that are available in the job description. There is an analogy in the case of pupils, because they are also living persons who would like to achieve better results. On the other hand, there are some differences as they are not 'sales products' and their self-knowledge is not as developed as adults'. Pupils require a guidance for evaluating, especially self-assessment. The best solution is if we provide a group of predefined factors and the pupils' task is only responding.

First we need to specify the proper internal and external factors that we can apply on pupils at school. What kind of criteria can be the base of our assessment, what are the internal and external factors of students? After we defined the necessary factors pupils can start to get knowledge about SWOT. It is important to inform them about the concept and definition of SWOT and the main aims of analysis. The teacher has to introduce self-assessment to them, while pupils have to make a decision in the case of each factor, whether it is a strength or a weakness, an opportunity or a threat. This 'driven' SWOT can be repeated during the academic year to strengthen pupils' self-assessment skill and to help them become more familiar with SWOT analysis. As pupils and teacher evaluate these analyses they implement a special case of formative assessment. They can clarify goals, they can track development and this activity develops pupils' self-assessment skills.

4.3 Internal and external factors at school

We have decades of experience in competency measurements, and the areas of competence defined by the EU are a good starting point as internal factors. From the point of view of computer science subject, it is also good to start from the competencies, which are arranged in a hierarchical structure. The 8 key competencies are known to everyone (In some cases there are separated the mathematical and the science and technology competencies): [8]

- Communication in the mother tongue;
- Communication in foreign languages;
- Mathematical competence and basic competences in science and technology;
- Digital competence;
- Learning to learn;
- Social and civic competences;

- Sense of initiative and entrepreneurship;
- Cultural awareness and expression.

Pupils can examine themselves by these competencies. Is the given competence their strength or weakness? They have to make decision; they need to practice their self-assessments by these factors. In terms of IT talent management these areas of general competences at the higher level cannot be neglected. Many IT talents have a hard time expressing themselves. The weaknesses of communication in the mother tongue still can be improved at secondary school. There are some international competitions in computer science where the talented students are also entered, so the knowledge of English language is required. It is a great loss if our best student cannot be named due to lack of language skills. Mathematical, Science and Technical competences are close to digital competence we refer those frequently as STEM (science, technology, engineering and mathematics). The areas of social and civic competences are also important and to be examined in terms of talent development, as well as the areas of competence of initiative, cultural awareness and learning ability.

The next step is a detailed investigation the components of relevant competence. The second level is no longer clear. There are several grouping of components in the case of digital competence. The European Commission has developed the European Digital Competence Framework for Citizens (DigComp) which is divided into five areas: information and data literacy; communication and collaboration; digital content creation; safety; and problem solving. [9] Another grouping is common used in Hungarian public education and it more fits to computer science subject. [8] This grouping divides digital competence to the next ten components:

- algorithmic thinking
- data modelling
- modelling the real world
- problem solving
- communication skills
- application skills
- team work, collaboration, interoperability
- creative skills
- orientation and information skills
- systemic thinking

Similar to first level of internal factors pupils have to make decision again. Is the given component of digital competence their strength or weakness? The result of this level can already provide several feedbacks to teacher. It informs about the interests of pupils. On the other hand, it provides feedback about the understanding of the given curriculum. Both help teachers to apply the proper differentiation in classroom. This is even more true for small group lessons where teacher can customize curriculum by pupil.

The result of this level, the components of digital competence allows us to continue our analysis in different ways based on the pupils' interests. It also helps the talent management as we analyse the pupils' interests and measure their strengths and weaknesses. There are three well defined areas of IT talent at secondary school: application, creative and programming.

The algorithmic thinking is a very strong base of programming so we expound on this component on the next hierarchical level.

- Recognising and Understanding Algorithms (a sequence of activities)
- Implementing Algorithms (a sequence of actions)
- Analysing Algorithms (a sequence of actions)
- Making Algorithms (a sequence of actions)
- Realising Algorithms (a sequence of actions)
- Modifying and Changing Algorithms (a sequence of actions)
- Designing Complex Algorithms (a sequence of actions)

The curriculum at secondary school doesn't allow the coverage all part of this level. Study groups and specialization classes provide possibility to practice programming, that requires the knowledge and applying of algorithms. Focus on implementing algorithmic element we go down to a new level. There are several directions to approach the specifications and levels of implementing algorithmic competence. Adapted to the curriculum of secondary school the following competences are relevant.

- Data types and data structures
- Concept and usage of variable
- Conditions
- Loops
- Patterns of algorithms
- The knowledge at least a programming language
- Value assignment (in the given language)
- File handling (in the given language)
- Functions and procedures (in the given language)
- Implement patterns of algorithms (in the given language)
- Implement recursive algorithms (in the given language)

The number of levels can vary per student and subject, as can the assessment criteria at each level. These assessment aspects themselves also help to formulate goals for both the pupils and the teachers.

The analysis is not complete yet, the previous criteria help to examine the internal factors, strengths and weaknesses. What are the external factors and how can we measure those? The external factors can be closely related to internal factors when analysing a person instead of a product. Opportunities can be related to strengths, while threats can be related to weaknesses. When the given competence is the strength of the pupil it is worth entering the competition, it is an opportunity. In the case of weakness, we have to improve the given competence of pupil to avoid threats.

There are other aspects to analyse external factors regardless competencies. The background and the environment of pupils are a real external factors. The next four categories can cover the most relevant external factors:

- School
- Social background
- Community
- Other external factors

The talent management provides some further interesting criteria that can be used for SWOT analysis. Czeizel extended Renzulli's talent model and defined his 2x4+1 factors model. [10] The main factors of this model are also interesting as an external and internal factors of SWOT.

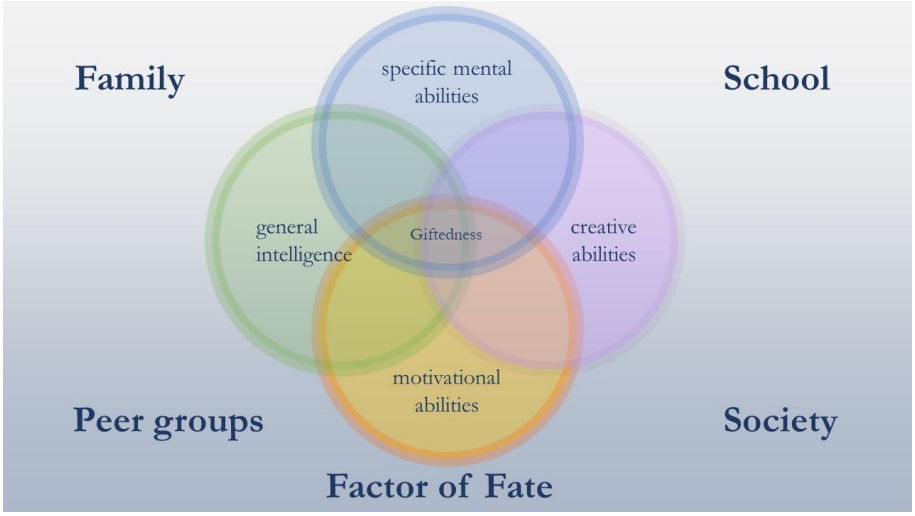

Figure 4: Czeizel's 2x4+1 factors talent model [10]

The internal factors are within the rings while the external factors are outside of rings on the chart. The intersection of internal factors refers to the possible talent. When a pupil is characterized by these presented internal factors we can suppose we found a possible talent. The four external factors can help or block to unfolding the given talent even just the additional fate factor. The teachers' scope is limited, it focuses on school and affects the peer groups as well partly the families. However, this scope can be enough to support the talented children at school. The benefit of SWOT analysis is that it helps detect the critical parts of external and internal factors.

There are well-known strategic models developed during the decades of practice in the literature of SWOT analyses. These models can also be transferred to school evaluation, especially in formative evaluation with a feedback and regulatory function. The four basic strategies come from the letter pairs written in the combined SWOT matrix. The figure 5 summarizes these basis strategies that can also be applied in school talent management.

	Strengths	Weaknesses
Opportunities	SO strategies Leverage strengths to maximize opportunities Attacking strategy	WO strategies Counter weaknesses through exploiting opportunities Build strengths for attacking strategy
Threats	ST strategies Leverage strengths to minimize threats Deffensive strategy	WT strategies Counter weaknesses and threats Build strengths for defensive strategy

Figure 5: SWOT basic strategies

- S-O Strategy: Offensive Strategy - Exploiting opportunities through strengths
- W-O strategy: Crisis-oriented strategy - Overcoming weaknesses by seizing opportunities
- S-T Strategy: Diversified Strategy - Defending Against Threats Using Strengths
- W-T Strategy: Defensive Strategy - Strategy that protects against the threat targeted to weaknesses

Choosing the right strategy together with the pupil is another way to engage students in assessment. It helps develop the students' analytical skills. At the same time, the teacher's pedagogical sense and experience are needed for the chosen strategy to contribute to the pupil's development. When some pupils recognize their weaknesses they cannot make a decision which strategy should be applied to develop themselves. Teacher has to help whether to start defensive strategy or develop as many areas as possible, or to focus on some strengths with offensive strategy. The decision also depends on the personality of the particular pupil. These difficulties affect all pupils regardless of their skills and marks.

The formative nature of the evaluation is strengthened by the repeating of SWOT analysis following the application of a strategy. We can try another strategy or can continue the selected one based on the new results. The pupils should be being involved both parts of SWOT, not only in the evaluative part of the analysis, but also in the selection of strategy, when we precise formulation of the learning goals.

4.4 Classroom examples of using SWOT analysis

SWOT analysis was tested within my classes several times. [7] The pupils of first sample group were graduate students of a small IT specialization group. They filled out a spreadsheet only about their internal factors. The evaluation pointed out which parts of curriculum requires more focus during the education. The pupils were confident on the level of key competencies, but they thought the 'designing complex algorithms' is their weakness. Examining individual pupils more weaknesses were recognized. It has already matched better to the teacher's evaluation. These internal factors were the following: The 'data modelling', the 'modelling the real world' and the 'systemic thinking' between the 'digital competences', while the 'making algorithms', the 'realising algorithms' and the 'designing complex algorithms' on the 'details of algorithmic thinking' level. The group-level results identified the teaching goals, what topics need to be practiced even more.

The individual-level results should be discussed pupil by pupil. The most of talented pupils are unaware with their talents while some other think themselves even more talented. SWOT analysis and the personal discussion about the results helped to clarify the actual level of skills. Teacher and pupil can discuss and mark together the areas that should be developed. The differs are reduced between the evaluation of teacher and pupils during the repetition of analysis, so SWOT helped develop pupils' self-assessment. The pupils from this group achieved a good graduation result and they were admitted to IT faculty of University. [7]

A new extended form was created based on the first experiences that contains more internal factors and it already contains external factors too. The aspects for internal factors compound from the well-known competences. The external factors based on the social background of Czeizel's talent model. [10] The form is available via internet and the results are stored in a database. There were examined two IT specialization groups from 10 and 11 school years.

The first step was the fill out the form. The prepared webpage displayed the aspects of internal and external factors and the pupils had to select the strength or the weakness and the opportunity or the threat row by row.

The examined internal factors are the following:

Internal Factors
Key Competencies - Communication
Speaking skill in native language
Reading, writing and comprehension (traditional)
Reading and comprehension (digital)
Typing
Foreign language skill
Key Competence - Mathematical competence and basic competences in science and technology
Arithmetic skill (fractions, percentages)
Geometry skill
Comparison, estimation
Mathematical rule recognition
Combinatorics, probability and computing
Numeral systems, binary numbers
Mathematical reasoning, writing and proof
Science (physics, chemistry, biology)
Technology, usage and theory
Key Competence - Learning to learn and social and civic competences
Skill of perseverance in learning
Awareness of the learning process, conscious learning
Organizing learning
Self-control
Note-taking
Scheduling
Intrinsic motivation
Skill of self-checking
Critical thinking
Cooperative learning
Integration into the community
Adaption to school

Key Competence - Digital competence
Algorithmic thinking
Data modelling
Modelling the real world
Problem solving
Communication skills
Application skills
Team work, collaboration, interoperability
Creative skills
Orientation and information skills
Systemic thinking
Details of algorithmic thinking
Recognising and Understanding Algorithms (a sequence of activities)
Implementing Algorithms (a sequence of actions)
Analysing Algorithms (a sequence of actions)
Making Algorithms (a sequence of actions)
Realising Algorithms (a sequence of actions)
Modifying and Changing Algorithms (a sequence of actions)
Designing Complex Algorithms (a sequence of actions)
Details of Implementing Algorithms
Data types and data structures
Concept of variable, variables, assign value
Conditional, logical expression
Loops
Patterns of algorithms
Knowledge of at least one programming language
Handling of variables, assignment, input
File handling
Functions and procedures
Implementation of patterns of algorithms
Implementation of recursive of algorithms

Table 10: Internal factors of SWOT

The examined external factors are as follows:

External factors
School
Teachers' knowledge
Teachers' humanity
Consultation with teachers
Study group, tutoring
Competition
Library
Study room
Backrgound
Family
Social background
Socio-economic status
Financial background
Own computer
Community
Classmates
Friends
Teachers
Parents
Relationship (friend, girlfriend)

Other
Freedom close to adulthood
Relax, entertainment
Trend and fashion
External motivation
Health (mental health)
Commute
Sports

Table 11: External factors of SWOT

The evaluation started with the comparison of pupils' marks and their strengths and weaknesses. While on the highest level of competencies there is not any discrepancy between teachers' and pupils' evaluation, there are differs on the deeper levels. These differs are important feedback for teachers. On the one hand, it provides feedback about pupils' self-assessment skills, so it implements one of the aims of formative assessment. On the other hand, it presents the understanding level of each topic from the given subject, so it implements another one aim of formative assessment.

SWOT analysis provides feedback to pupils too. When they evaluate the components of competencies as an internal factor they can recognize what are the most important competences to be acquired. The repetition of SWOT analysis a few months later helps improve pupils' self-assessment skills.

First let's see the aggregated internal factors. Each strength means one and each weakness means minus one. The serious weaknesses are important feedback to teachers, these competencies should be improved. Some of those belongs to computer science subject, so it defines our next task with the given group. Some others weaknesses belong to other subject and other teachers, so these should be discussed with colleagues.

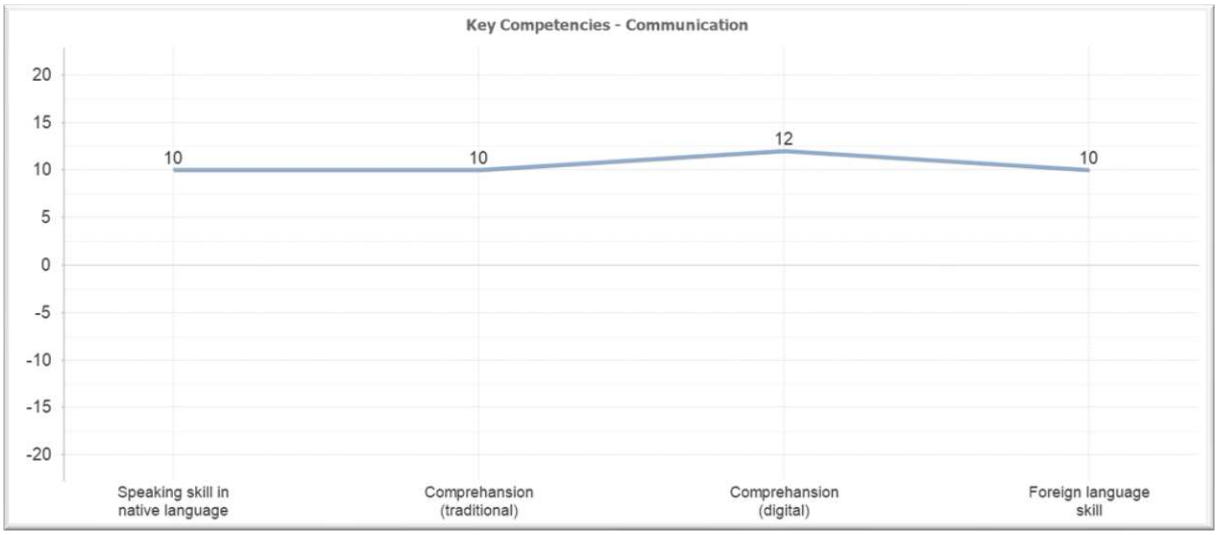

Figure 6: Key Competencies - Communication

The first group of internal factors are related to native and foreign language competences. The reading-writing and comprehension were divided two parts by the format, because there can be differences between paper-based and digital reading and comprehension based on the assessments. [11] In the case of the examined group typing skill was significantly low. It was confirmed by other examined groups so we inserted some typing lessons into the curriculum.

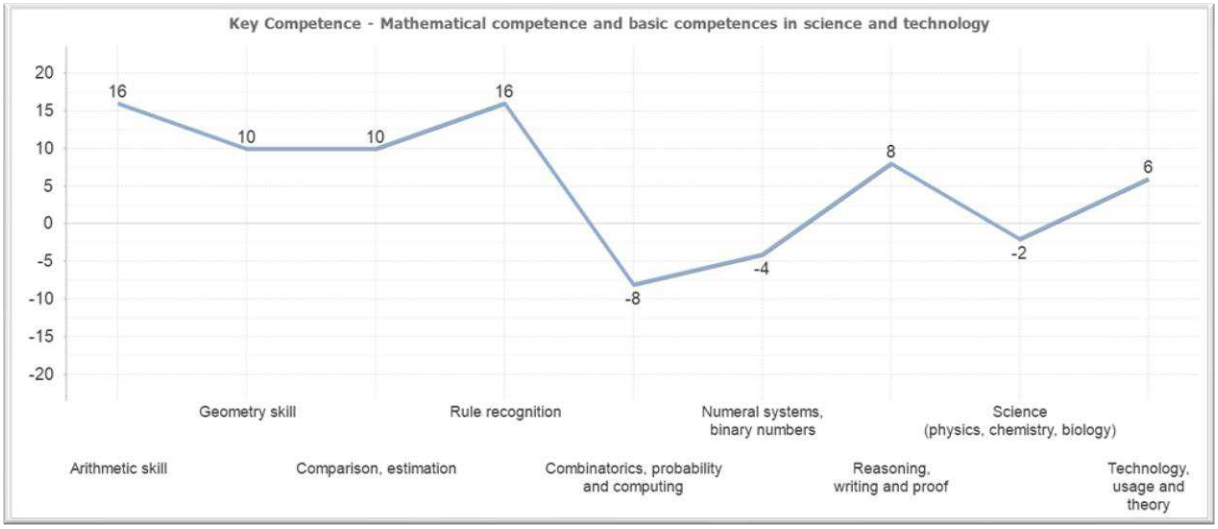

Figure 7: Key Competencies - STEM

The STEM skills were weaker. The combinatorics, probability and computing are new at secondary school, it is reasonable that pupils need more practice. The low level of science skills reflects to curriculum that reduced the number of science lessons in the last couple of years. There is no reasonable explanation for the low level of numeral systems and binary numbers knowledge. It was an important feedback of SWOT analysis.

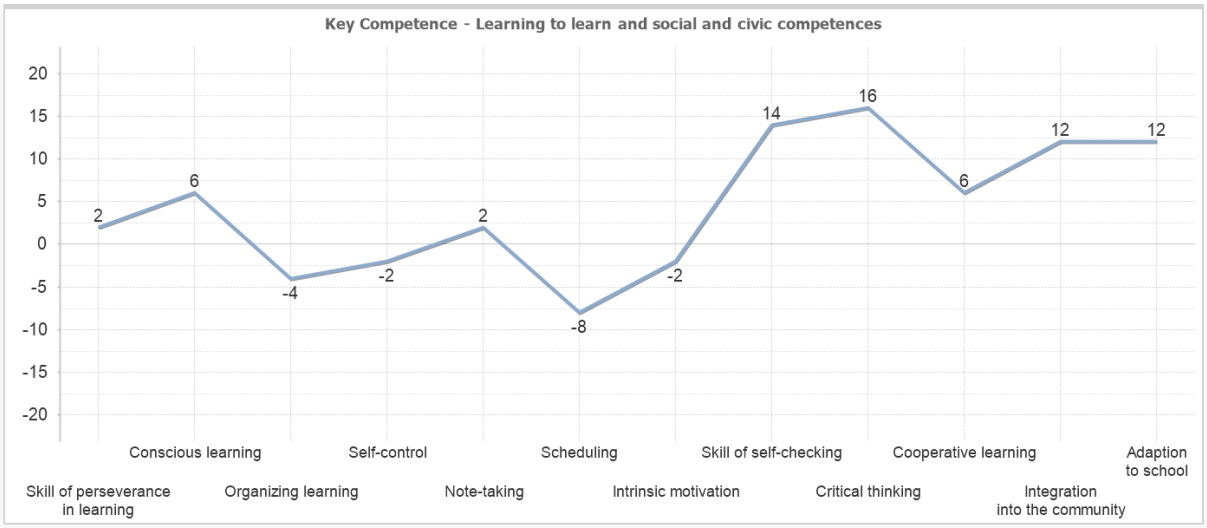

Figure 8: Key Competencies – Learning to learn and social competences

The examination of learning to learn and social competencies are new internal factors that were not investigate last year. There are four factors under zero. The weakness of scheduling, self-control and the organizing learning are age-specific, they have to develop these skills if they would like to go to University. The lack of intrinsic motivation made me think. Would that be burnout? The lessons need to be made much more interesting and the practices require more interesting tasks. Lack of motivation doesn't affect only IT lessons, however it confusing in the case of IT specialization group.

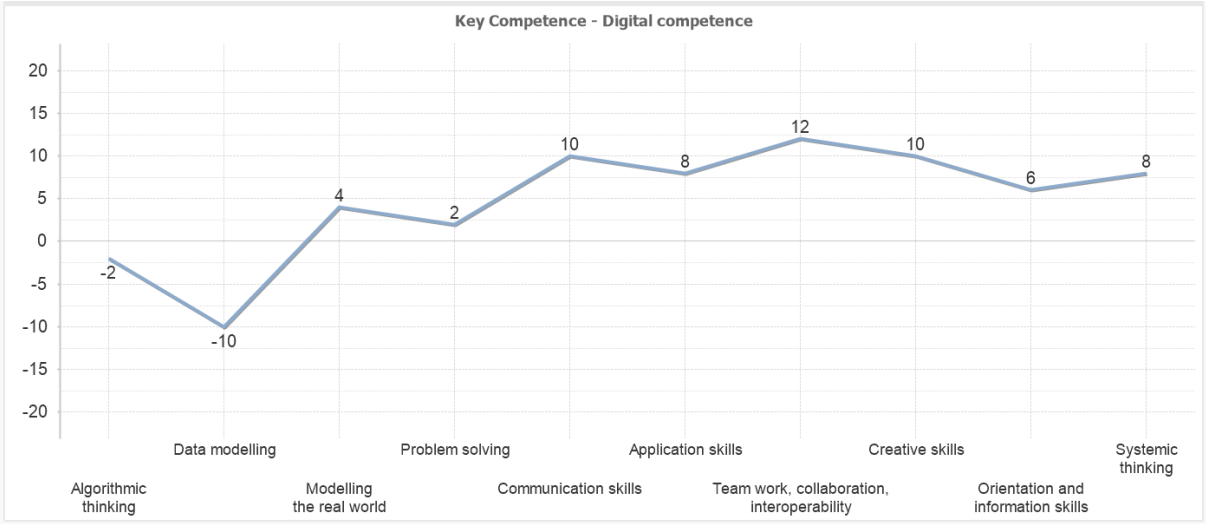

Figure 9: Key Competencies – Digital competences

The components of digital competencies are well-known within the group and the pupils' evaluation were close to teacher's opinion. The differs were at the negative values and these came from the pupils evaluated their skills to weaker than teacher thought. Based on the experience of the task solution the pupils recognized their weaknesses regarding algorithmic thinking and data modelling. In this special case in addition to further practice pupils also need more time to deepen their knowledge.

After the evaluation of general competences, the examination of internal factors focusses on computer science. A selected competence component was examined using the drill-down method as a next step. The 'algorithmic thinking' is an important base of programming that is the aim of this course, therefore, we chose this for deeper analysis.

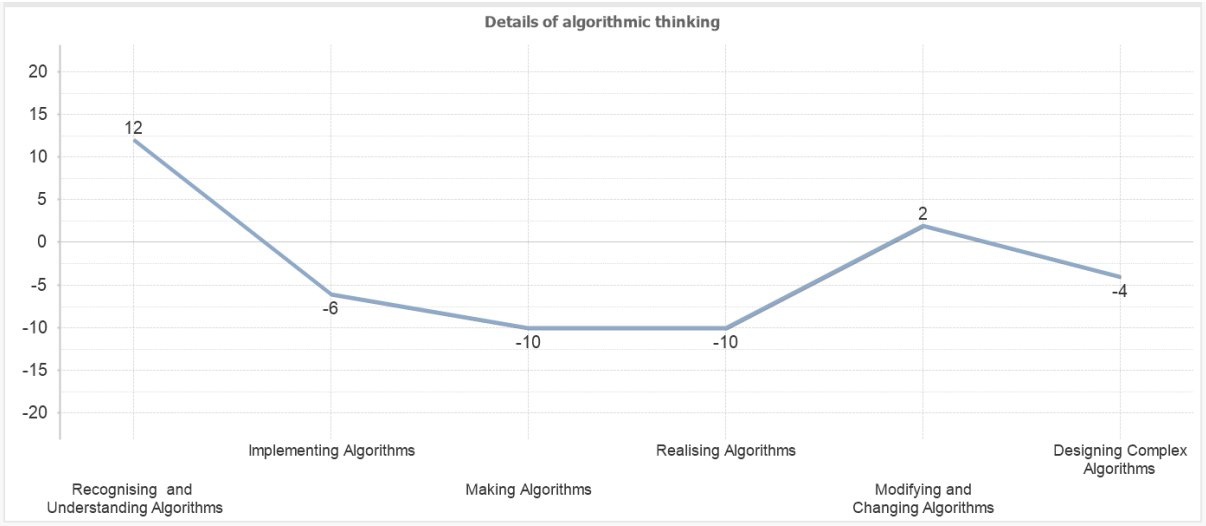

Figure 10: Details of Algorithmic thinking

Similar to previous level the less-known and less-practised topics were evaluated to even neutral and the well-known and well-practised topics were evaluated to weaker. The most frequently practiced part is the implementing algorithms and this component is not a major weakness based on the teacher's evaluation. The making algorithms and realising algorithms are also well-practiced components and the analysis pointed out the importance of developing these parts.

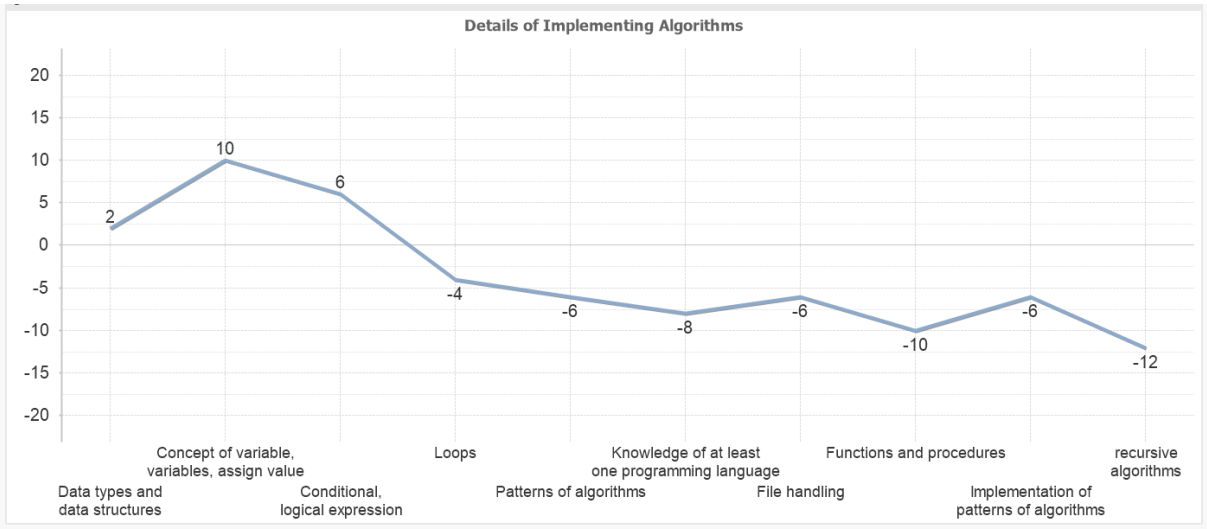

Figure 11: Details of Implementing Algorithms

The details of 'implementing algorithms' is the deepest level that contains elementary simple to knows. The concept of loops is well-known similar to conditional and logical expression. Behind the weaknesses are the differences between the different kind of loops, that are not so clear for the given group. It affects other components, like patterns of algorithms and implementation of patterns of algorithms as the pupils often mix the required loops during implementation of algorithm patterns. It also affects to knowledge of at least one programming language and file handling as these use different kind of loops. This is again a well-defined area that the teacher has to teach better. The question about the recursive algorithm is a control point. We haven't learnt about it yet. It was set as strengths by two pupils who learnt about it at summer camp.

The external factors are not so interesting on the group level except for some questions. The pupils were satisfied with their school background and their family background, these contain opportunities rather than threats. However, owning a computer is a threat rather than an opportunity as the online virtual world is stealing their time. The community around the pupils is also supportive. However, most of the pupils thought the relationship (between a boy and a girl) is a threat, similar to own computer, and the time requirement is the reason for this. Pupils thought they can manage their downtime, entertainment and the increased freedom. But they thought the influence of trends and fashion are threats. The last external factor that provides threat is the commute. The study group classes are usually late in afternoon and it is a threat to commitment.

The previous analysis helps the teacher to improve their teaching process as well as it helps pupils recognize the details of requirements. The personal level of SWOT provides useful support to talent management as it allows to customize curriculum. It cannot be implemented within a standard class, but it can be implemented within study group. The personal discussion is a good possibility to clarify real strengths and weaknesses as well the real opportunities and threats. At the end of this kind of discussion the teacher and pupil can define together the goals for the next period.

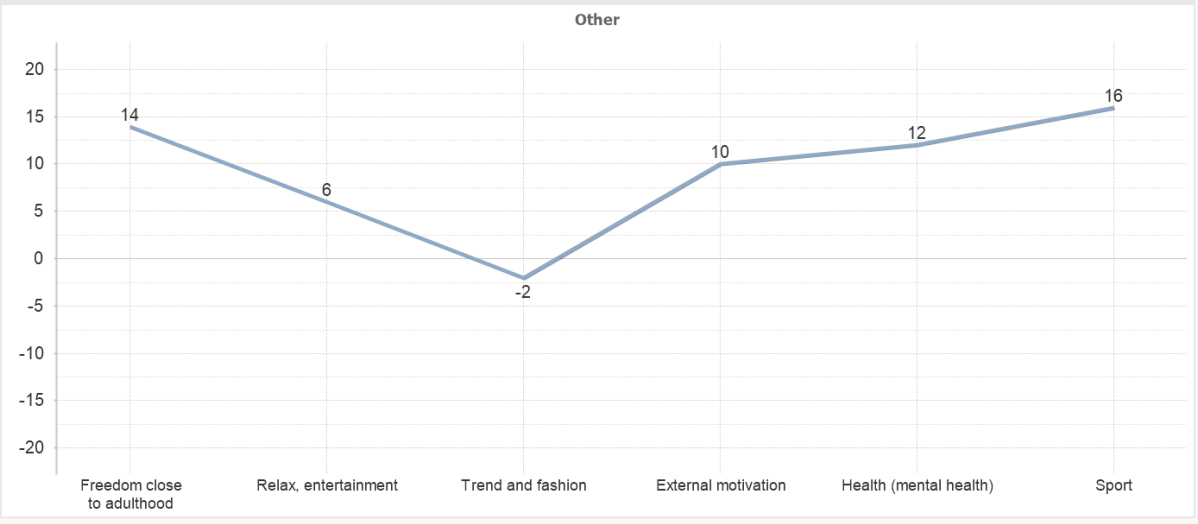

Figure 12: Other external factors

The following table contains a pupil's SWOT:

Internal factors	
Key Competencies – Communication	
Speaking skill in native language	1
Comprehension (traditional)	1
Comprehension (digital)	1
Typing	-1
Foreign language skill	1
Key Competence - Mathematical competence and basic competences in science and technology	
Arithmetic skill, fractions, percentages	1
Geometry skill	1
Comparison, estimation	1
Mathematical rule recognition	1
Combinatorics, probability and computing	-1
Numeral systems, binary numbers	1
Mathematical reasoning, writing and proof	1
Science (physics, chemistry, biology)	-1
Technology, usage and theory	1
Key Competence - Learning to learn and social and civic competences	
Skill of perseverance in learning	1
Awareness of the learning process, conscious learning	1
Organizing learning	1
Self-control	1
Note-taking	1
Scheduling	-1
Intrinsic motivation	1
Skill of self-checking	1
Critical thinking	1
Cooperative learning	1
Integration into the community	1
Adaption to school	1

Table 12: General competencies of a pupil

According to the analysis, the selected pupil has good opportunities. His general education and intelligence are outstanding, he enjoys IT and has a strong basis. The typing skill can be developed by practices and he received a link to a site with online typing lessons. He was informed during the

discussion about the importance of mathematical and science skills and his teachers were also involved to his development. The scheduling is not an easy question, he received some suggestions like he should create weekly and daily agendas that he has to check regularly. The second SWOT helps to check the improvement a few months later.

Key Competence - Digital competence	
Algorithmic thinking	-1
Data modelling	-1
Modelling the real world	1
Problem solving	-1
Communication skills	1
Application skills	1
Team work, collaboration, interoperability	1
Creative skills	1
Orientation and information skills	1
Systemic thinking	1
Details of algorithmic thinking	
Recognising and Understanding Algorithms (a sequence of activities)	1
Implementing Algorithms (a sequence of actions)	1
Analysing Algorithms (a sequence of actions)	-1
Making Algorithms (a sequence of actions)	-1
Realising Algorithms (a sequence of actions)	-1
Modifying and Changing Algorithms (a sequence of actions)	-1
Designing Complex Algorithms (a sequence of actions)	-1
Details of Implementing Algorithms	
Data types and data structures	-1
Concept of variable, variables, assign value	1
Conditional, logical expression	1
Loops	-1
Patterns of algorithms	-1
Knowledge of at least one programming language	-1
Handling of variables, assignment, input	1
File handling	1
Functions and procedures	1
Implementation of patterns of algorithms	1
Implementation of recursive of algorithms	-1

Table 13: Digital competencies of a pupil

The personal discussion between the teacher and pupil helped to clarify the internal factors of digital competences. The data modelling was a real weakness, while algorithmic thinking and problem solving were close to strengths. The given pupil understands the algorithms and he can implement those in C++ environment. The clarified goals are the practicing of analysis and changing of algorithms.

There were some differences between teacher's evaluation and pupil's evaluation on the deepest level that was clarified during personal discussion. The pupil knows the necessary data types and data structures and he handles these in C++ environment. The teacher's suggested a table creation that contains loop types and loop indexing by pattern of algorithm. The fulfilment of defined goals should be checked via repeated SWOT a few months later.

External factors	
School	
Teachers' knowledge	1
Teachers' humanity	1
Consultation with teachers	-1
Study group, tutoring	-1
Competition	1
Library	1
Study room	-1
Backrgound	
Family	1
Social background	1
Socio-economic status	1
Financial background	1
Own computer	1
Community	
Classmates	1
Friends	1
Teachers	1
Parents	1
Relationship (friend, girlfriend)	1
Other	
Freedom close to adulthood	1
Relax, entertainment	1
Trend and fashion	1
External motivation	1
Health (mental health)	1
Commute	-1
Sports	1

Table 14: External factors of a pupil

The external factors contained some threats that should be avoided. It became clear during the personal discussion that there is only one real threat: commute. The pupil has to go home from the school before he can talk with the teacher and he cannot join prepare sessions for competition. As the given pupil is motivated and he has a good chance in competitions the teacher proposed an online preparation for him. It is a good example of applying different strategies. The suggestions contained an S-O strategy as built on the strengths that exploit opportunities, while a W-T strategy as it protected from the threats that target its weaknesses. The focus on the customized personal strategy is what supports talent management at secondary school.

The above presented analysis and the repeating of SWOT analysis is a good method for applying formative assessment. The predefined internal and external factors can be changed depend on the examined pupils. There are two kind of IT study groups at the examined school. The presented internal factors help the analysing the pupils of programmer group. The other study group focuses on creative component of digital competencies, the pupils study vector graphics and raster graphics applications. In this case the drill-down to details of digital components uses other questions to explore internal factors. In this case the survey analyses the creative skills part of digital component on different levels.

This SWOT analysis also was tested at an IT department of University where it examined freshmen during the first semester. The applied internal factors were same as used at programmer study group, while the external factors were specified to freshmen. There are more possible threats in this special age when students start their study at university parallel they enter to adult age. SWOT

helps them recognize several threats and the discussion with a mentor can help to explore opportunities and some relevant strategies.

5. Conclusion

The above presented SWOT analysis with DISC assessment [12] are mutually supportive in school talent management as well education in general. Both methods are very useful tools for both parties at public education. These are deservedly commonly used assessments in human resource development in the business world and these methods works at secondary school too, especially in talent management. While SWOT analysis helps detect strengths, weaknesses, opportunities and threats and assists to find necessary strategy, DISC helps the teacher select the proper communication based on the given pupil's personality.

SWOT analysis provides feedback on the success of teaching that is important from the teachers' point of view. On the other hand, it helps to clarify the curriculum to be learned for pupils and the multiple implemented analyses develop the pupils' self-assessment skill. It should be noted that the SWOT result depend on pupils' decision at a given moment. For the right evaluation require a personal discussion between the teacher and pupil. Personal communication does miracles usually.

In addition, the school environment provides possibility of long term observation of pupils. While psychologists or talent experts examine test results, mark and meet with pupils only few times, the teacher meets with them every week. The classroom work, the regular test, homework and the repetition of SWOT assessments allows the teacher a deeper observation and get detailed and confirmed information about pupils.

Talent management is not new within the education, however it is outdated, while it is more modern in the business world. These assessments are regularly used in human resource management and corporate talent management programs [13][14]. However, it is not really used within public education, meanwhile the school talent management is the exact environment where these assessments could be used efficiently. It has to be noted these assessments support talent management rather than talent search and talent recognition.

SWOT analysis is a good complement of DISC assessment in the case of talent management. Similar to company talent management, the selected pupils have to create a SWOT analysis together with their teacher. The common personal analysis helps to recognize and understand for both side the real strengths and weaknesses of a given pupil. SWOT can also be part of common used portfolio assessment; it is a good complement of its. Based on the SWOT result the teacher can provide a personal coaching to pupil that can reduce weaknesses and improve strengths. The young talents require the customized attention and care. The teacher task regarding the talent management can be supported by tutor and mentor. SWOT analyses as well DISC assessment provide the necessary inputs to the team that develop the talented pupils. This special process is called coaching in the business world and it should be spread at secondary school.

Bibliography

1. Friesner, Tim: *History of swot analysis*; (2011) https://www.researchgate.net/publication/288958760_History_of_swot_analysis

2. Gürel, Emet; Tat, Merba: *SWOT analysis: a theoretical review*; The Journal of International Social Research (2017)

http://www.sosyalarastirmalar.com/cilt10/sayi51_pdf/6iksisat_kamu_isletme/gurel_eme.pdf

3. Addams, Lon; Allred, Anthony; Chertudi, Mike: *The first step in proactively managing students' careers: Teaching Self-SWOT analysis*. In: Academy of Educational Leadership. Vol. 17, No. 4 (2013) p. 43-51.

4 Sarmasági, Pál: *Formatív értékelés SWOT-analízissel*; In: Szlávi, Péter; Zsakó, László (szerk.) InfoDidact2019 Zamárdi, Magyarország. Webdidaktika Alapítvány, (2019) pp. 253-264. https://people.inf.elte.hu/szlavi/InfoDidact19/betolt.html

5. Thira Woratanarat, Patarawan Woratanarat: *Assessment of Prospective Physician Characteristics by SWOT Analysis*; (2012), Malays J Med Science https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3436489/

6. Radnóti Katalin: *Milyen oktatási és értékelési módszereket alkalmaznak a pedagógusok?* In: Kerber Zoltán (Szerk.) Hidak a tantárgyak között. Országos Közoktatási Intézet, Budapest (2005)

7. Szerk: Károly Krisztina, Homonnay Zoltán: *Mérési és értékelési módszerek az oktatásban és a pedagógusképzésben*; ELTE Eötvös Kiadó, Budapest (2017)

8. Zsakó, L.; Szlávi, P.: *ICT Competencies: Algorithmic thinking;* Acta Didactica Napocensia Vol 5 (2012)

9. *Digital competence: the vital 21st-century skill for teachers and students* (2020) https://www.schooleducationgateway.eu/en/pub/resources/tutorials/digital-competence-the-vital-.htm

10. Czeizel, E.; Páskuné Kiss,J.: *A tehetség definíciói, fajtái* Didakt Kiadó, Debrecen (2015)

11. Balázs, I.; Ostorics, L.: *PISA2009 Digitális szövegértés*, Oktatási Hivatal, Budapest (2011)

12 Sarmasági, Pál: *DISC assessment usage in school talent management*; In: Abonyi-Tóth, Andor; Stoffa, Veronika; Zsakó, László (szerk.) New Methods and Technologies in Education, Research and Practice: Proceedings of XXXIII. DidMatTech 2020 Conference Budapest, Magyarország. ELTE Informatikai Kar, (2020) pp. 183-202. http://didmattech.inf.elte.hu/proceedings-2020/

13. Scott, W.B.-Jackson: *HR is Business: Achieving competitive advantage through strategic talent management* Oxford Brookes University Business School, Oxford (2008)

14. Weiming, Gao: *Study on the Application of DISC Behavioral Style in Talent Management in Banking Industry*; In: *Deng Mingran: Proceedings of the 8th International Conference on Innovation & Management* Wuhan University of Technology Press, Wuhan, China (2011) p.760-767.

Authors

SARMASÁGI Pál

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary,
e mail: psarmasagi@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 3, Number 2. 2021

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.3.2.1355

License

The Role of Variable in Programming: Examples and Methodology

TÖRLEY Gábor, ZSAKÓ László

Abstract. One of the hardest notions to define in programming is the variable and the related command of assignment. In our opinion, it is exactly these difficulties that are responsible for the reluctance towards programming. The reason for this, according to us and others [7], is the multifunctional nature of the variable: it can be used for various purposes. Its concept "in our heads" and in the programming languages is markedly different in this respect.

Keywords: programming, variable, programming methodology.

1. Everyday Algorithms

The fact that during our everyday activities we perform several algorithms, from sequences, branches, and repetitions, to non-deterministic and parallel structures, facilitates our conception of the algorithm [2, 3, 6].

There are programming languages designed for beginners (like ELAN), in which the concept of algorithm is included, but not the concept of data [6].

The data world appears in manifold ways: first, the data are some kind of *objects,* which can be grouped into *classes* (and not as variables in the traditional sense). Such a class is the family, where there is a mother, a father, children, etc. The elements (objects) of the class are the specific family. In everyday algorithms, such objects appear in diverse forms; in default cases, their values are constant, only rarely variable.

Sidenote: Typically, beginners are wary of the concepts of class and object-oriented programming. The underlying reason is that in the classical, von Neumann-style execution of programs, the coder sits in the position of the "processor" and performs the program sequentially, contrary to the above mentioned, everyday-world experience of the concept. This parallelism, implicitly or explicitly present in the object-oriented approach, creates confusion. Languages for beginners include so-called seeded objects (like the turtle in Logo or the cat in Scratch) to avoid this problem.

Interestingly, the structures of the classes are complex straight away; the basic types appear only as parts of the class structure. Contrarily, programming education almost always begins with basic types. The primary composition mode is the direct multiplication (the record concept), which can be complemented by special multitudes: a *set* of clothes, *queue* in the supermarket, etc.

Arrays appear in an odd way too; their indexes can be the floors an elevator passes (negative indices are also possible), or the stations of a railway (where the index can be the very name of the station). In this sense, we can call arrays as indexed structures.

2. The Role of Mathematics

Mathematics introduces number types (integer, real; even if mathematics uses natural and rational numbers as well) and sequences (infinite is possible in mathematics) into the data world. The latter can lead to the concept of the array.

Then there are the matrices as well, but we can bump into two-index structures much earlier, for example, during spreadsheet operations; in fact, they can even precede the sequence concept of mathematics.

The peculiarity of spreadsheets is that they are not "embodiments" of what we could traditionally call variables, rather a functional model of solving a problem [4]. There are constants in certain cells, while in others, functions are applied for specific cells or cell groups. Therefore, functions have parameters by definition. In default cases, however, we calculate all these functions together. If the value of some cells changes, the functions are recounted. Instead of the algorithmic activities, which would apply variables, this model features operations (such as sum, maximum, etc.) defined for ranges (table parts) [9].

Note that everyday algorithms contain similar calculations too (for example, if we buy three cappuccinos, for 2 EUR per piece, then we will pay 6 EUR), but not even then will we store the "sum" in a separate "variable."

3. Why is Programming Difficult?

There are many articles about the difficulties of teaching programming. Hofuku et al. [5] illustrate it by a very simple example-pair:

```
int i;
for (i = 0; i < 10; i++) {
  printf("Hello");
}
```

```
int i = 0;
while (i < 10) {
  printf("Hello");
  i++;
}
```

As they write, *"both programs (…) show that the instruction repeats display a string Hello 10 times. To understand these programs, learners should know variables, data type (int), variables initialization, assignment, compare operators, Boolean values, and increment operator. Many concepts are necessary to understand these simple structures."* The problem identified by the authors relates to the concept of the variable.

Nevertheless, a drawing procedure in Logo, drawing a square with :n side length, causes much less trouble. We draw a square with :n side length by repeating making :n steps forward and turning 90 degrees to the right four times.

```
to square :n
  repeat 4 [forward :n right 90]
end
```

Therefore, here there is no need for the classical concept of the variable.

Usually, it is difficult for students to understand how variables work and the role and concept of them [8].

4. Types of Variables

Sajaniemi et al. [7] summarize the roles and types of variables in the following table (table 1):

Role	Informal definition
Fixed value	A data item that does not get a new proper value after its initialization
Stepper	A data item stepping through a systematic, predictable succession of values.
Walker	A data item traversing in a data structure.
Most-recent holder	A data item holding the latest value encountered in going through a succession of unpredictable values, or simply the latest value obtained as input.
Most-wanted holder	A data item holding the best or otherwise most appropriate value encountered so far.
Gatherer	A data item accumulating the effect of individual values.
Follower	A data item that gets its new value always from the old value of some other data item.
One-way flag	A two-valued data item that cannot get its initial value once the value has been changed.
Temporary	A data item holding some value for a very short time only.
Organizer	A data structure storing elements that can be rearranged.
Container	A data structure storing elements that can be added and removed.
Other	Any other variable.

Table 1: Roles of variables [7]

Our categorization shares some of the elements of the above table. Our methodology, however, fits better with constructivist pedagogy, reflecting the path of how learners get in contact with variables during their studies and how they get to understand their concept. It will also become clear that the two main groups of our categories are linked to whether the notion of assignment has already been introduced or not. Accordingly, the first group contains variable types where the assignment is not yet present [10].

The most classical form of dealing with a "variable" is when it comes to the constants of other subjects (e, pi, g, etc.), which are essentially *numerical constants with names*. They are not variables in the sense that practically they replace a character sequence. (As a consequence, programming languages typically assign the given value to such constants while translating the code; that is, they are not truly treated as variables.) These constants generally do not even appear in the description of tasks (or their names may come up, but it is their value that needs to be used).

Constants in task descriptions play a similar role. For example, from the description "let us read the first 100 numbers", it comes that just like pi, we could name 100 as maxn, for instance. Such constants (with or without names) can appear in task descriptions or input conditions. Nevertheless, they will also be added to the code during translation.

There are constants (so-called *quasi constants*) which appear as constants for people (for example, we know the height of a given "n" person), but in the program, they will turn up as special variables that will get value at some point (as the data are retrieved), and then we will use them only for calculating. In several classical programming tasks, input variables will not change after being retrieved.

```
for i := 1 to n do
  read(x[i])
end for
calculate(n, x)
```

In many cases, it might occur that the retrieved values can be considered constants, but we use a value sequence for the same variable, and we even process it right away. Such constants can be called *quasi constants for multiple purposes.*

```
for i := 1 to n do
  read(y)
  calculate(y)
end for
```

Perhaps the first notion that can be truly interpreted as a variable is the *state component,* such as the position, the direction, etc. of the turtle in the Logo language. We can get the values of the state component, and we can also change them with the help of the dedicated functions or operations. Nevertheless, assignments in the classical sense should be avoided in the case of state components. Most often, state components have their name, so there is no need for a new definition.

The notion of *parameter* is brought in by the procedure (function). In the functional approach (and, for example, in the turtle graphics of Logo, too) parameter is essentially the name of the value, which will be copied in the place of the parameter during function call (with lazy evaluation a bit later). In this case, every parameter is strictly an input parameter only; thus, value is assigned only once, which does not change. The result of the function is a value (perhaps complex), which might be part of an arbitrary expression.

The first data appearing really as variables come up when we need to calculate formulas. These are called *variables calculated from others* (x := f(y)). Their role lies in their name, so essentially these variables are the names of function values stored in variables in classical programming languages.

```
Read(y)
x := f(y)
Write(x)
```

We can build the above algorithm easily on a functional approach: let us calculate something with the input data and write out its result:

```
Write(f(Read))
```

This functional thinking can facilitate and prepare the introduction of variables (which is why it is useful to apply programming languages containing functional programming elements, like Logo or spreadsheets built on functional bases).

Variables, calculated from others, *containing only value* (y := f(x); z := y * (y + 1)) are used solely for the purpose of efficiency, shortening, etc. In the following algorithm excerpt, aimed to calculate the second-based time difference between two dates, TA and TB are such variables.

```
TA := A.hour * 3600 + A.min * 60 + A.sec
TB := B.hour * 3600 + B.min * 60 + B.sec
difference := TB - TA
```

When processing data multitudes (like sequences, sets, etc.), we might need to handle each of their elements, for which *variables applied for a given range* can be used. Such are direct or indirect loop variables.

```
for i := 1 to n do …

for x in H do …
```

```
x.first; while not x.end do … x.next
```

Certain loop types may allow for the partial traversal of a structure as well.

The above data (may they be constants or variables) do not logically require the related but more complicated notion of *assignment*. The classical notion of assignment is the generalization of the memory cell of von Neumann-type computers. We can load data into this named storage; we can get and change them, among others. Consequently, such a notion of the variable does not need to be present in all computing models.

State variables can have a special role. In simple cases, they are logical values (like processed, unprocessed) or in classical graph traversal, they are the colors denoting the state of the points (white, grey, or black). State variables are similar to the state components of the turtle or robots, but here assignment commands make sense, traditional variables are involved.

We often use state variables for ending loops.

```
needed := true
while needed do
  …
  if … then needed := false
  …
end while
```

Programming based on the theory of finite automata frequently applies such state variables.

There are several operations with which we can add or distract elements in data multitudes. These are the *variables changing the number of elements* (such as stack, queue, etc.). They lack classical assignment, but they have *push, pop,* etc. operations. The elements we add or distract, however, are classical variables, which can undergo any kind of operation.

The assignment is probably the hardest to understand with *variables cumulating a specific value*. The main difficulty with them is that in many languages, the description of assignment highly resembles the description of equality check in mathematics (for example, C++: x = f(x, y)), which is mathematical nonsense. Such variables collect the values of a data structure with some cumulative operation.

```
x := …
for y in H do x := f(x, y)
```

A classical version is summation:

```
x := 0
for y in H do x := x + y
```

or maximum selection:

```
x := minval
for y in H do x := max(x, y)
```

The variable of classical counting loops can be such cumulating-type variables:

```
i := 1
while i ≤ N do
  …
```

```
  i := i + 1
end while
```

where the changes of the i variable are often marked with special commands (for example, i++, inc(i)).

Variables calculated recursively from their own previous value are similar to cumulative variables. They frequently appear in mathematical tasks. While the factorial can be described with a simple cumulative variable:

$$n! = \begin{cases} 1 & if \quad n = 0 \\ n * (n-1)! & if \quad n = 1 \end{cases}$$

```
fact := 1
for i :=1 to n do fact := fact * i
```

with Fibonacci numbers, it is easier to calculate the next element of the sequence from the previous values (which not surprisingly is simpler to understand than the cumulative variable of the factorial).

$$Fib(n) = \begin{cases} 0 & if \quad n = 0 \\ 1 & if \quad n = 1 \\ Fib(n-1) + Fib(n-2) & if \quad n > 1 \end{cases}$$

```
fib[0] := 0
fib[1] := 1
for i := 2 to n do fib[i] := fib[i - 1] + fib[i - 2]
```

Easier example is the arithmetic sequence for this role:

```
seq[1] := 2
d := 3
for i := 2 to n do seq[i] := seq[i - 1] + d
```

Frequently, we need to perform operations on data or data multitudes that are trivial to do parallelly, but for sequential execution, we need *temporary storage space,* for example, swapping two variables:

Parallelly	**Sequentially**

`swap(A, B):`
`  A, B := B, A`
`end swap`
	`swap(A, B):`
	`  s := A; A := B; B := s`
	`end swap`

or the cyclic rotating of a sequence:

```
rotate(A):
  c := A[1]
  for i := 2 to n do A[i - 1] := A[i]
  A[n] := c
end rotate
```

Similar to state variables are *variables storing some other variable's value.* Typically, we need them when we want to process a multitude by completely traversing it, but as a result, we expect just one element. For example, such a variable is the one containing the index of the maximum value element in maximum selection:

```
maximum(A):
  max := A[1]
  for i := 2 to n do
    if A[i] > max then max := A[i]; index := i
  end for
  maximum := index
end maximum
```

If the formation time and the processing time of the data are different (in time or order), we need variables to temporarily store the data in a given order (stack, queue, dequeue, priority queue, etc.). These are the *variables defining processing order*.

Another group of the variables (and with this one, we are getting far from the first types) constitutes *variables describing relations*. They are typically used in hierarchical and network data structures (trees, graphs, etc.), but structures linked from a certain aspect are similar too.

The above categorization is summarized below:

Group name	Informal definition
Numerical constants with names	Names to substitute character sequences, behind which there are clearly defined values, like e, pi, g, etc.
Constants in task descriptions	Constants which can appear for example in input conditions.
Quasi constants	They appear as constants for people, but in the program, they will turn up as special variables that will get value only when retrieved.
State components	We can get or change their values, but assignments in the classical sense should be avoided.
Parameters	Name of the value which will be copied in the place of the parameter upon function call (with lazy evaluation a bit later).
Variables calculated from others	Names of function values which are stored in variables in classical programming languages.
Variables applied for a given range	Such are direct or indirect loop variables.
State variables	Compared to state components, the notion of assignment commands is included.
Variables changing the number of elements	The operations for adding or distracting elements in data multitudes, they do not involve classical assignment.
Variables cumulating a specific value	They collect the values of a data structure with some cumulative operation.
Variables calculated recursively, from their own previous value	Variables which are calculated from their own previous value
Temporary storage space	Variables that store data for only a very short time.
Variables storing some other variable's value	When we want to process a multitude by completely traversing it, but as a result, we expect just one element.
Variables defining processing order	Variables which temporarily store the data in a given order.
Variables describing relations	Variables that describe the relations of hierarchical and network data structures.
All other variables	Every variable that does not belong in any of the above is categorized into this group.

Table 2: Categorization of variables

5. Comparative Analysis

Comparing Sajaniemi's [7] categories with ours, thus we can conclude:

- Both categorizations follow constructivist principles.
- Their definition of "fixed value" matches our categories "numerical constants with names" and "constants in task descriptions."
- Their "stepper" group fits our "variables applied for a given range" category.
- The variables in their "most-recent holder," "most-wanted holder," and "gatherer" groups correspond to our "variables cumulating a specific value." In our view, it is always

a cumulative function that gives value to a variable, and it depends on the programming theorem behind the function whether the variable belongs to the "most-recent holder," the "most-wanted holder," or the "gatherer" group.

- A part of the variables grouped as "gatherer" falls into our "variables calculated recursively, from their own previous value" category. In our opinion, it is important to make a distinction here because wherever the concept of recursion comes up, it is a different level of cognition compared to cumulative variables.
- Our "quasi constants used for multiple purposes" is the part of their "most-recent holder" category.
- Their "one-way flag" category matches our "state variables."
- Their "temporary" group corresponds to our "temporary storage space."
- Their "container" group is similar to our "variables describing relations" and "Variables defining processing order". We have separated these two groups because data structures in our former group are linear, and the latter group contains non-linear data structures usually.
  - The rest of the groups cannot be matched directly. It is worth noting that Sajaniemi et al. do not consider whether the assignment has already been introduced into the learning process or not, whereas in our categorization, it was a key factor. On the other hand, our grouping was based on the everyday concept of algorithm.
  - The different categories and concepts of the variables are related to each other in the following way (figure 1):

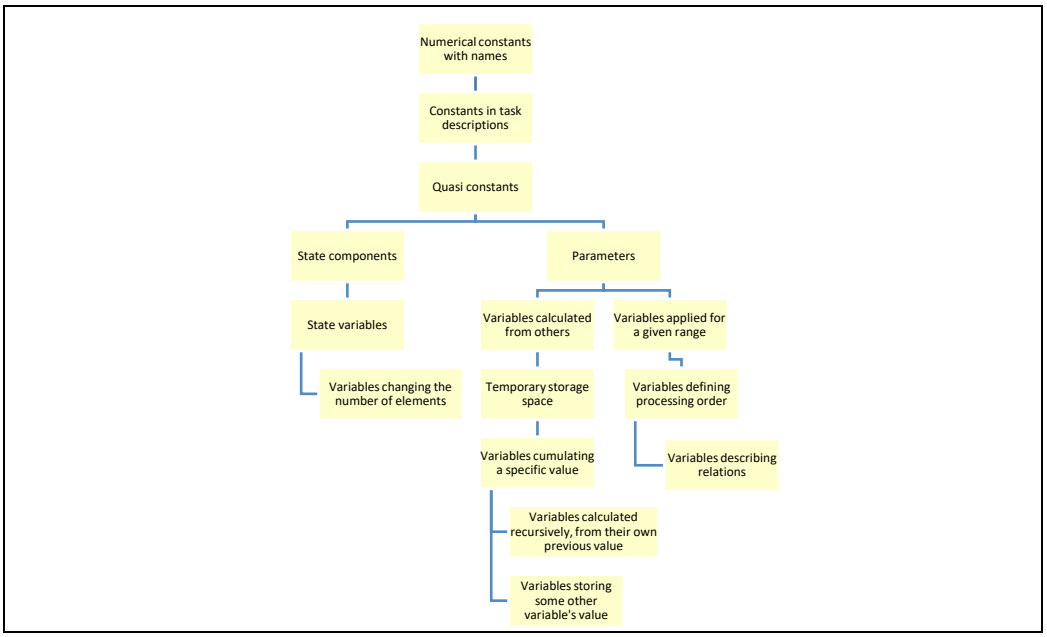

Figure 1: Structure of our categorization

6. Which Type of Variable can be Understood Easily and which with Difficulty?

Usually, the role of scalar variables can be understood easily than the role of more complex data structures. Good examples of that are *quasi constants* and *variables cumulating a specific value*. Variables of these categories can be scalars and arrays as well. For a beginner, it cannot be easy to understand that an array is a variable, which has one identifier, but it contains more elements.

The next difficult role is *parameter* because it can be challenging to understand the difference between formal and actual parameters because they have different identifiers. At the introduction of this category, it can be a good solution to use the same identifiers for formal and actual parameters. Programming languages use different implementations of parameter handling, so learners need to distinguish parameter handling methods. We did not mention the parameter handling method earlier because it depends on the programming language, and we focused on the algorithm, not on coding.

Those variables' role, which are *calculated from their own previous value,* is not difficult to understand but what is difficult is the recursion and the way of thinking behind that. This kind of variables will be used, for example, when we want to solve a recursive problem without recursion e. g. we want to memorize all the elements of a recursive sequence instead of calculating the last element recursively.

7. Object-Oriented Programming

The beginning of learning programming is easy if the learner can imagine herself as the executor of the program. Classical von Neumann programming builds upon this principle with sequential executing. Variables will be needed in order to store, preserve, calculate information. In this paradigm, we split the task in top-down direction to simple procedures, and we pass the necessary data to the procedures.

OOP programs want to represent the phenomena of the real world, which consists of independent and co-ordinate objects. So, when we are programming, the most natural way of thinking is to choose those objects from the real world that are important to us and give them functions.

An object can be described by two parts (like the objects in the real world):

- Attributes: inner and outer attributes, this is a data field.
- Activities: These are the operations that can be executed by or on the object. These are the methods of the object.

Usually, we need more similar objects in a program, so it seems to be expedient to summarize the common attributes. This way, we get a schema of these similar objects, which is called *class,* and each object will be a concrete entity of the class.

Understanding the concept of object and class is difficult, like understanding the concept of variable and type. Beginner programmers can avoid type definitions until they use variables with a given structure (like int, real, etc.). The concept of *datatype* is abstract; it is not easy to understand. If the programmer should define a datatype (or data structure), then she should create all operators, procedures, functions, and data to that. If we skip the concept of inheritance, then the concept of class is very similar to the concept of datatype. But the concept of class is needed for understanding the concept of inheritance.

It is getting more problematic when a solution of a task needs more parallel working objects in time. It is understandable when we want to examine the function of objects individually. But it is challenging to understand the parallel working of the whole system.

In general, most introductory programming textbooks and courses have a syntax-driven organization of the material. The programming language has the "main role", and it is described in a bottom-up way, starting with the simpler constructs of the language and then progressing to more advanced constructs [1].

According to Sorva et al. [11], the defined roles by Sajaniemi et al. [7] can be related to data structures:

- An example of an Organizer is a variable that contains an array of numbers during sorting,
- a variable that references a stack could be a Container,
- a variable that contains a reference to a node in a tree traversal algorithm and a variable that keeps track of the search index in a binary search algorithm can be considered to be Walkers.

As we mentioned in the introduction, beginners are wary of the concepts of class and object-oriented programming.

According to our grouping, *"variables defining processing order"* and *"variables describing relations"* can be understood easier if we refer the former group as classes and refer the latter group as one of the data of a class.

An interesting task can be when we want to represent rational numbers. This can be necessary if we want to calculate very big (i.e., 100 digits) numbers. In this case, every number should be represented in an array. Should we use independent methods, operators, or class methods here?

```
a := add(b, c) (independent method)

a := b + c     (operator)

a := b.add(c)  (class method)
```

In all cases, variable `a` will have the role *variables cumulating a specific value*. We want to answer the question above with another example. We can represent a stack data structure with independent methods or with a class.

Representation with independent methods:

```
stack_elements: array
stack_top: integer
…
procedure push(stack_elements, stack_top, element)
  stack_top := stack_top + 1
  stack_elements[stack_top] := element
end procedure
…
procedure pop(stack_elements, stack_top, element)
  element := stack_elements[stack_top]
  stack_top := stack_top – 1
end procedure
…
```

Representation with a class:

```
class Stack
  Set of values
    elements: array
    top: integer
  …
  Set of operations
    procedure push(element)
```

```
        top := top + 1
        elements[top] := element
      end procedure
      …
      procedure pop(element)
        element := elements[top]
        top := top - 1
      end procedure
      …
End class
```

According to our experiences, if we start teaching programming with the procedural paradigm, then the procedural representation could be a good transition towards object-oriented programming. With this transition, students can understand that the roles of variables do not depend on the programming paradigm, and they can understand the concept of encapsulation if they compare the procedural and the object-oriented representation.

8. Conclusion

When introducing the concept of the variable, it is best to apply approaches based on the data concept of everyday thinking.

This is close to the core concept of one-object languages (Logo, Scratch), except that even in these languages, we should avoid using variables (Logo is much more suitable for this because apart from the state components of the turtle, everything else can be a parameter). Functionalization (such as spreadsheets or Logo's functional elements) is also important when introducing the concept of the variable.

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

Bibliography

1. J. Bennedsen, M. E. Caspersen: *Teaching Object-Oriented Programming — Towards Teaching a Systematic Programming Process,* Proceedings of the Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object-Oriented Concepts, 18th European Conference on Object-Oriented Programming (ECOOP 2004), 14–18 June, (2004), Oslo, Norway.
2. T. Bell, I. H. Witten, M. Fellows: *Computer Science Unplugged,* An Enrichment and Extension Programme for Primary-Aged Children; http://csunplugged.org (Retrieved: 26.01.2021.)
3. P. Bernát, L. Zsakó: *Methods of Teaching Programming – Strategy.* XXX. DIDMATTECH 2017, Trnava University in Trnava, pp. 40–51, (2017)
4. J. Cunha, J. P. Fernandes, J. Mendes, J. Saraiva: *Spreadsheet Engineering.* Lecture Notes in Computer Science, Vol. 8606, pp. 246–299, (2015)

5. Y. Hofuku, S. Cho, T. Nishida, S. Kanemune: *Why is programming difficult? Proposal for learning programming in "small steps" and a prototype tool for detecting "gaps. in Informatics in Schools.* Sustainable Informatics Education for Pupils of all Ages, Potsdam, Springer, (2013)

6. C. H. A. Koster: *Systematisch leren programmeren,* Educaboek, 1984.

7. J. Sajaniemi, M. Ben-Ari, P. Byckling, P. Gerdt, Y. Kulikova: *Roles of Variables in Three Programming Paradigms,* Computer Science Education, Vol. 16, No. 4, 261–279 Dec (2006)

8. T. S. Sklirou, A. Andreopoulou, A. Georgaki, and N. D. Tselikas: *Introducing Secondary Education Students to Programming through Sound Alerts,* EJERS, Vol. 5, No. 12 (2020): 130–139

9. Zs. Szalayné Tahy: *How to teach programming indirectly – using spreadsheet application.* Acta Didactica Napocensia; Cluj-Napoca Vol. 9, No. 1, (2016): 15–22.

10. P. Szlávi, G. Törley, L. Zsakó: *The most difficult notion of programming: The variable,* In: E. Salata, A. Buda: Education – Technology – Computer Science in Building better future Radom, Poland: Wydawnictwo Uniwersytetu Technologiczno-Humanistycznego w Radomiu, (2018) pp. 108–118., 11 p.

11. J. Sorva, V. Karavirta, A. Korhonen, A.: *Roles of Variables in Teaching.* Journal of Information Technology Education: Research, 6(1), 407–423. Informing Science Institute (2007)

Authors

TÖRLEY Gábor

ELTE Eötvös Loránd University, Faculty of Informatics, 3in Research Group, Martonvásár, Hungary,
ORCID: 0000-0002-0496-936
e-mail: gabor.torley@inf.elte.hu

ZSAKÓ László

ELTE Eötvös Loránd University, Faculty of Informatics, 3in Research Group, Martonvásár, Hungary,
ORCID: 0000-0002-4614-1509
e-mail: zsako@caesar.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 3, Number 2. 2021

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.3.2.1436

License