Programming as a mathematical instrument: the implementation of an analytic framework

Andreas Borg, Maria Fahlgren, and Kenneth Ruthven

Karlstad University, Department of Mathematics and Computer Science, Sweden
andreas.borg@kau.se

This paper relates to an ongoing project using design-based research as a methodological approach in which students with no prior experiences of using programming as a mathematical tool are observed trying to solve mathematical problems with the help of programming. The Instrumental Approach is used as conceptual framework in which the concept of instrumental genesis describes the process where the programming environment as an artefact together with student-developed mental schemes forms an instrument in order to solve mathematical problems. The development of schemes is of special interest in this paper where Vergnaud's components of a scheme provide a framework for analysing transcripts of talk between student pairs and the programming code that they generate.

Keywords: mathematics education, computer programming, instrumental genesis.

INTRODUCTION

During the past decade, there has been a renewed recognition of programming as an important digital competence to be developed as part of the general education of all students, and of its particular relationship to mathematical competence. This has been recognized in changes to curricula in many countries: in France, Finland, and Sweden, for example, programming is included in mathematics curricula. In Sweden, where the study described in this paper took place, programming was in 2018 included in mathematics from year 4 in lower secondary school. In upper secondary school, programming is to be used as a tool for mathematical problem solving.

Papert (1980) argues that using programming in school and in mathematics could have positive effects on children's learning and could help students to develop new cognitive skills. According to Hoyles and Noss (2015), the use of programming in mathematics education is seen by most students as an engaging activity where they can independently "build, learn from feedback and debug" (p. 7). Programming is also a means for developing creativity and ability in problem solving (Romero, Lepage, & Lille, 2017) and offers a natural opportunity for students to be exposed to mathematical concepts closely related to programming, e.g. iterations (Noss, 1986). But although the introduction of programming has the potential of offering new possibilities for learning, Drijvers and Gravemeijer (2005) argue that the integration of new technologies in mathematics education can be complicated and that it would be naïve to believe that "we can separate techniques from conceptual understanding and that leaving the first to the technological tool would enable us to concentrate on the latter" (p. 164). Instead they argue that machine techniques and conceptual understanding must be interwound and be developed simultaneously. Drijvers and Gravemeijer

(2005) consider this interwinding as a fundamental part of *The Instrumental Approach* which will be described in the following section.

THE INSTRUMENTAL APPROACH

The Instrumental Approach originates from the field of cognitive ergonomics (Rabardel, 2002) and considers the process where a subject, involved in a goal-driven activity, uses an artefact (a material or abstract object) to act towards a given objective. During the process when the subject appropriates the artefact to her/his needs and integrates the artefact with her/his activity the subject develops mental utilization schemes associated both with use of the artefact and with the objective of which the artefact should act towards (Rabardel, 2002). These schemes can be usage schemes – directed towards the artefact itself - or instrumented action schemes - directed towards the object of the activity. The artefact together with the associated mental schemes constitutes an instrument for the subject, where the instrument is regarded as a psychological construct. The process through which the instrument is formed is called the *instrumental genesis* and is, in this ongoing research project, followed with special interest when studying how students (the subjects) use a programming environment (the artefact) when solving a mathematical problem (the objective).

The development of schemes

In order to study the instrumental geneses of students, the development of mental schemes is therefore of special interest. Vergnaud (1998) argues that mental schemes can be divided into four different components; *goal and anticipations, rules of action, operational invariants, and possibilities of inferences* and this paper will focus on students' use of different rules of action and operational invariants. The rules of action are considered by Vergnaud (1998) as the generative part of the scheme, directed by operational invariants (Buteau, Gueudet, Muller, Mgombelo, & Sacristán, 2019). Every action is built upon some information or concepts and Vergnaud (1998) thus regards concepts-in-action as a vital part of the operational invariants. The second part of the operational invariants consists of theorems-in-action, regarded as "proposition[s] which [are] held to be true" (p. 168) by the subject when s/he acts. Vergnaud (1998) argues that there is a relationship between concepts-in-action and theorems-in-action since "concepts are ingredients of theorems" (p. 174).

Buteau et al. (2019) used Vergnaud's (1998) four components of a scheme as an analytic frame in order to analyse how university students engage in mathematical inquiries using programming as a mathematical tool. They argue that their use of the framework has "deepen[ed their] understanding of what is at stake in terms of students' learning in this particular context" (p. 17) and has served as a means to illustrate students' instrumental geneses. In accordance with the work of Buteau et al. (2019), Vergnaud's (1998) components of a scheme will serve as an analytic framework for this study and the research question that the study is addressing is: *What are the instrumental geneses of upper secondary school students' use of programming environments in trying to solve mathematical problems pre-designed to lend*

themselves to programming? The question of methodology which this specific paper addresses is: *How can Vergnaud's (1998) components of a scheme be operationalized for this study?* Since the instrumental geneses in this study relate to students' use of programming environments as a mathematical problem-solving tool, the schemes developed during the intervention relate to the problem-solving process as a whole and not to specific parts of it. But due to limited space within this paper, only a specific section of the problem-solving process will be described.

METHOD

The findings presented within the paper are part of a research project using design-based research as the overarching research method. In 2019, 27 eleventh grade students in a Swedish upper secondary school participated in the teaching intervention of the first design cycle of a lesson in which students are intended to solve mathematical tasks using a non-standard problem-solving strategy (e.g. an exhaustive trial) involving use of the programming environment. The students were, at the time of the intervention, taking the same introductory course in programming and thus had basic knowledge of coding but no experience of using the programming environment as a mathematical tool during their ongoing course in mathematics. Due to the students' prior study of coding, it was assumed that they had already developed basic usage schemes related to the use of the artefact (programming environment). The focus of this study is thus on the development of instrumented action schemes directed towards mathematical problem solving.

Data collection and data analysis

During the intervention students worked in pairs and three of the pairs were followed more closely through the use of screen-capturing software which also recorded the conversation between students in each pair. This data was of special interest when studying the development of schemes since it allowed the researcher to identify stable behaviours important when analysing students' instrumental geneses (Buteau et al., 2019). The researcher (who acted as the teacher) also wore a microphone to record conversations between students and the researcher. All voice recordings collected during the teaching intervention of the first design cycle were transcribed using NVivo. During the analysis, data from the recordings were grouped into themes relating to different parts of the problem-solving process. Then these themes were coded in an iterative process using Vergnaud's (1998) components of a scheme. Both verbatim abstracts and code generated by students have served as evidence when analysing the development of students' schemes.

RESULTS AND ANALYSIS

In this section, examples will be given of how verbatim abstracts from conversations between students during the intervention have been used together with the generated programming code to analyse the development of students mental schemes using Vergnaud's (1998) components of a scheme as the analytical framework. In this paper,

the use of rules of action and operational invariants will be of special interest. The mathematical problem will not be described in detail in this paper but concerns finding the ages of three sisters. The problem can be mathematised algebraically in terms of relationships involving the ages of the sisters but not in a manner which permits the students to solve the problem using algebraic methods already known to them. Therefore, it was anticipated that the students might use the programming environment in order to conduct an exhaustive trial, a process which will be analysed in the following sub-section.

Developments of schemes relating to the implementation of an exhaustive trial

During the intervention, the researcher asks a pair of students called Sophie and Richard to describe their problem-solving strategy and how this had been implemented using the programming environment. Sophie explains how the pair have used nested loops in order to conduct an exhaustive trial:

> Sophie: Yes, we have a nested *for*-loop so the first one... Eh... Or starting with *a* is zero and every time it goes around then *a* increases by one. But before that happens, *b* is set to *a* plus eleven and then comes the next *for*-loop which then tests every age between *a* and *b*, which is then the mid sister. And if this formula we came up with is true then the loops should stop. […]

The outer loop (Fig. 1) is thus used by the pair to systematically increase the value of the variable *a* which concerns the age of the youngest sister. Within this loop the value of *b*, the age of the oldest sister, can be calculated using a given relationship between *a* and *b*. The inner loop is used to vary the variable concerning the age of the mid sister and thus runs for integer values between *a* and *b*. Within the inner loop an IF statement is used to test a mathematical condition within the task involving the ages of the sisters.

```
7   int a;
8   for(a = 0; a < 1000; a++){
9       int b = a + 11;
10      for(int f = a; f < b; f++){
11          if ((b+1)*(f+1)*(a+1)-(b+1)*(f+1)*a == 432){
12              break;
13          }
14      }
15  }
16  System.out.print(a);
```

Figure 1: Screen shot visualizing the nested loops generated by Sophie and Richard

Based on the verbatim abstract above, several rules of action used by the pair could be identified: (a) *formulating the problem situation as amenable to solution through exhaustive trial;* (b) *making use of programming to implement a solution strategy based on exhaustive trial*; (c) *creating iterations through defining conditions for loop(s); and* (d) *making use of the conditional operator IF to (i) evaluate given conditions in order to (ii) perform different actions based on the validity of given conditions*. It could be argued that these rules of action are justified through several concepts-in-action involving the ideas of (a) *conducting an exhaustive trial*; (b) *systematically combining variables*; (c) *establishing a loop relating to a variable*; (d) *nesting loops (and statements within them) in order to achieve an appropriate sequence*

of variable-related actions; and e) *using conditions within loops and conditional operators in order to extract a solution within a given range*. These concepts-in-action form two theorems-in-action which guide the actions of the pair: (a) *Systematically combining variables serves as a means of achieving an exhaustive trial when more than one variable is in play* and (b) *Establishing nested loops relating to the key variables in play is a means of systematically combining these variables*.

The ideas of students Emilia and Fredrik, on the other hand, are less developed. In particular, they struggle to articulate – either orally or in code – how to systematically combine variables. In the nested loops shown in Fig. 2, the outer WHILE loop is used to check if the variable *guldmynt_f* (dependent on the sisters' ages, and recalculated with each traversal of the loop) is less than or equal to 432 (a crucial value in the problem). The inner FOR loop includes the same condition and initialises a control variable *i* incremented on each traversal, which is then (mis)used within the loop, apparently with the intent of systematically increasing the variable *b*, the age of one of the sisters.

```
16          while(guldmynt_f<=432){
17              for (int i = 0; guldmynt_f <=432 ; i++) {
18                  b = b+i;
19
20                  guldmynt_f = (a+12)*(a+1)*(f+1)-(a+12)*(a)*(f+1)-432;
21
22              }
23          }
```

Figure 2: Screen shot visualizing the nested loops generated by Emilia and Fredrik

Fredrik then realizes that the other age variables *a* and *f* are never assigned new values within the loops.

Fredrik: We should increase everyone? I think.

Emilia: No, but we just need... We just need to increase the age of Cinderella (variable *a*) because the others increase automatically because we have written *a* plus twelve there and a little something else as well.

Fredrik: Yes but... We still need to increase *f*.

Fredrik deletes the calculation of *b* in line 18 and inserts $f = f + i$ instead. Later, Fredrik returns discussion to the control variable *i,* which he relates to testing values associated with an (unspecified) year and age:

Fredrik: But we need to find out what year it is... Because now they are zero years old... That's why we have to test values all the time.

Emilia's and Fredrik's way of coding indicates the use of concepts-in-action based on the ideas of (a) *conducting an exhaustive trial;* (b) *establishing a loop relating to a variable*; and (c) *nesting loops (and statements within them) in order to achieve an appropriate sequence of variable-related actions*. But their failure to code loops which will systematically increase the values of variables indicates that the pair, unlike Sophie and Richard, have underdeveloped rules of action relating to how to use nested loops

to combine variables in order to conduct the exhaustive trial. As a consequence, there is also a lack of theorems-in-action guiding the pair's actions.

A third pair of students, Christian and David, initially have a clear view about how to use the programming environment as a mathematical tool.

> Christian: Actually, you could set it up mathematically… Or... and then just use *brute force* by testing lots of different combinations with the computer.

Although not made explicit by the verbatim extract above, the program structure (Fig. 3) later reveals the pair's intention of conducting an exhaustive trial in order to solve the given task. Unlike the other pairs, Christian and David also realize that they can take advantage of the fact that the ages of the sisters must take integer values, and so use this to establish an additional condition.

> Christian: But yes, what we can do is just have a eh... WHILE TRUE and then we have... And then we have... And then we check for that one so then you have like eh... If this… and then you have like *and*-signs for... Eh.... And check if something is INT (integers).

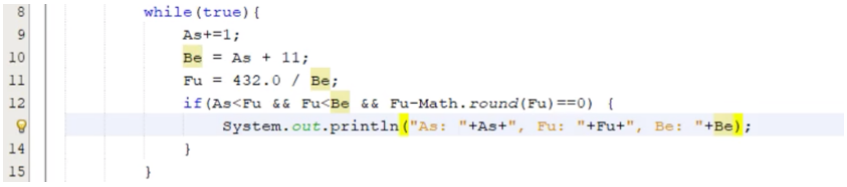

Figure 3: Screen shot visualizing the single loop generated by Christian and David

This additional condition allows the pair to create a program (Fig. 3), which only uses a single loop involving three variables corresponding to the age of each sister. The variable controlling the loop is *As* (age of the youngest sister), initialised as 0, and, at the start of each iteration, increased by 1. Thus, the loop systematically examines what happens as *As* increases from 1. Within the loop, the *Be* (age of the oldest sister) is then specified as *Be = As + 11* and the third variable *Fu* (the age of the mid sister) is calculated as *Fu = 432 / Be* (although it should be noted that the underlying definitions of the ages used in these two calculations are not compatible with each other). The IF-statement in line 12 checks three conditions which need to be met in order for the combination of ages to be a solution to the problem. The first two conditions check if *Fu* is the mid sister and the third condition checks if the value of *Fu* holds an integer value. If *Fu* is an integer, the difference between *Fu* and the rounded value of *Fu* equals zero. If all the three conditions are met the program should print the ages of the sisters.

The verbatim extract together with the code generated by Christian and David expose how the pair has used several different rules of action during the development of the program: (a) *formulating the problem situation as amenable to solution through exhaustive trial;* (b) *making use of programming to implement a solution strategy based on exhaustive trial*; (c) *creating an iteration through defining conditions for a loop; and* (d) *making use of the conditional operator IF to (i) evaluate given conditions*

in order to (ii) perform different actions based on the validity of given conditions. These rules of action are justified through several concepts-in-actions involving ideas of (a) *conducting an exhaustive trial;* (b) *computing linked variables when more than one is in play;* and (c) *using conditions within loops and conditional operators in order to extract solutions within a given range*. These concepts-in-action are related to a theorem-in-action used by Christian and David stating that: *Computing linked variables when more than one is in play serves as a means of reducing the number of variables to be systematically varied in an exhaustive trial*.

The examples given in this section are small extracts from students' problem-solving process when trying to solve a mathematical problem using a programming environment as a mathematical tool. Yet, they illustrate different approaches used by students in order to conduct an exhaustive trial and also difficulties relating to conceptual and computational understanding. The way Sophie and Richard try to conduct an exhaustive trial differ from the method used by Christian and David. This is illustrated by the components comprising their developed schemes, although some generic components relating to the problem-solving strategy are common. Emilia's and Fredrik's scheme could be regarded as deficient since it lacks several essential components relating to the use of nested loops, an action often perceived as conceptually difficult for novice programmers (Mladenović, Boljat, & Žanko, 2018).

DISCUSSION

Following Buteau et al. (2019), we have explored approaches to operationalizing Vergnaud's (1998) components of a scheme when studying students' instrumental geneses. Using conversations between students (and between students and the teacher) together with their generated code has made it possible for the researchers to extract different components of schemes explicitly stated by students (or shown in their program structure). This in turn has presented a possibility for the researchers to search for similarities and differences within different schemes as well as analyzing which components are missing from deficient schemes. This is illustrated by the example where Emilia and Fredrik, just like the other two pairs, had begun to develop an instrumented action scheme directed towards mathematical problem solving, involving the use of exhaustive trial to solve the mathematical problem. But the lack of well-functioning rules of action relating to the use of nested loops, in order to systematically combine variables, hindered this pair to implement their problem-solving strategy. This deficiency within their instrumented action scheme illustrates that Emilia and Fredrik may have had under-developed pre-existing usage schemes directed towards the artefact itself relating to the use of (nested) loops.

We also argue that defining specific components of the scheme based on conversations between students should not be seen as a straightforward process. In the analytic and iterative process there has to be a balance between defining components, on the one hand, generic enough to be able to search for commonalities across cases, but on the other hand still specific enough not to lose the key characteristics of each case.

Since most of the data involves conversations between students during an ongoing problem-solving process involving a particular situation, it cannot strictly be argued that the findings provide evidence of "the invariant organization of behavior for a certain class of situations" (Vergnaud, 1998, p. 167). But, at the least, the data shows how, in solving the task, students generate proto-schemes or schemes-in-progress which together with the artefact start the formation of an instrument. Indeed, this is no more than the notion of a dynamic, constructive process of instrumental genesis.

REFERENCES

Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristán, A. I. (2019). University students turning computer programming into an instrument for 'authentic' mathematical work. *International Journal of Mathematical Education in Science and Technology*, 1-22. doi:10.1080/0020739X.2019.1648892

Drijvers, P., & Gravemeijer, K. (2005). Computer algebra as an instrument: Examples of algebraic schemes. In D. Guin, K. Ruthven, & L. Trouche (Eds.), *The didactical challenge of symbolic calculators: Turning a computational device into a mathematical instrument* (pp. 163-196). Boston, MA: Springer.

Hoyles, C., & Noss, R. (2015). A computational lens on design research. *ZDM, 47*(6), 1039-1045. doi:10.1007/s11858-015-0731-2

Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education Information Technologies, 23*(4), 1483-1500. doi:10.1007/s10639-017-9673-3

Noss, R. (1986). Constructing a conceptual framework for elementary algebra through Logo programming. *Educational Studies in Mathematics, 17*(4), 335-357. doi:10.1007/BF00311324

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

Rabardel, P. (2002). *People and technology - a cognitive approach to contemporary instruments* Retrieved from https://hal.archives-ouvertes.fr/hal-01020705

Romero, M., Lepage, A., & Lille, B. (2017). Computational thinking development through creative programming in higher education. *International Journal of Educational Technology in Higher Education, 14*(42), 1-15. doi:10.1186/s41239-017-0080-z

Vergnaud, G. (1998). A comprehensive theory of representation for mathematics education. *The Journal of Mathematical Behavior, 17*(2), 167-181. doi:10.1016/S0364-0213(99)80057-3