"I Show You How I Solved It!"

Empowering Novice University Students to Learn Programming and Mathematics Through Self-produced Videos to Potentially Teach to Their Peers

Terry Inglese[1], Lukas E. Fässler[2], and Patrik Christen[3(✉)]

[1] Institute for Information Systems, FHNW University of Applied Sciences and Arts Northwestern Switzerland, 4002 Basel, Switzerland
[2] Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
[3] Institute for Information Systems, FHNW University of Applied Sciences and Arts Northwestern Switzerland, 4600 Olten, Switzerland
patrik.christen@fhnw.ch

Abstract. A relevant concern of Java programming and mathematics instructors is that first-year college students usually have difficulties in grasping the abstract concepts of these two disciplines. Consequently, a meaningful part of students may fail to pass their core exams in BIT, *Business Information Technology*. To overcome this issue, two maths and programming instructors and a researcher in psychology of education, implemented the so-called *exploratory video-based instructional intervention*, through which BIT students were asked to explain specific Java programming concepts and to describe how to solve three maths exercises using self-produced videos. From a diagnostic perspective, the instructors were able: a) to recognise the correctness of the syntactic knowledge, the conceptual knowledge and the strategic knowledge of Java programming and b) to distinguish whether students were correctly applying the foundations of mathematics, which are essential skills for becoming a good programmer. The results of this experimental study showed that first-year students appreciated the production of these self-explaining videos, resulting in mastering complex abstract concepts in mathematics and in programming.

Keywords: Programming · Mathematics · Self-explaining · Teaching · Student video · Transfer learning

1 Introduction

Programming and mathematics are two essential disciplines for learners who study *Business Information Technology* (BIT) at the FHNW University of Applied Sciences and Arts, based in Northwestern Switzerland. Furthermore, being a BIT student also means becoming a future qualified expert, able to professionally communicate and work professionally with diversified clients and specialists within the two fields of Business and Information Technology. However, in the past years, as a trend in our BIT programme, approximately half of

S.-H. Wang and Y.-D. Zhang (Eds.): ICMTEL 2022, LNICST 446, pp. 260–273, 2022.
https://doi.org/10.1007/978-3-031-18123-8_20

the first-year students do not pass their programming exam. One of the problems we encountered, as instructors, was the difficulty for many students to relate new abstract concepts to existing knowledge. One of the characteristics of our students are their diversified academic and professional backgrounds. Not all of them have a strong foundation in mathematics and almost many of them never experienced programming before enrolling in the study program.

The focus of this paper is to show how instructors (two programming/mathematics instructors – the second and the third authors of this paper – together with an instructor of psychology of learning, didactics and business communication) introduced in two courses (programming and mathematics) the so-called *exploratory video-based instructional intervention*, through which students were asked: a) for the programming course, to explain a specific programming concept, and b) for the mathematics course, to describe how to solve three maths exercises, using – in both courses – self-produced videos.

The instructional aim of this *exploratory video-based instructional intervention* (which was for students an optional task to earn additional learning points, added to their final grade) was threefold: first, to promote the choice of using multimedia in self-explaining (and explaining to other peers) specific concepts and procedures that are rather abstract and theoretical. Second, we, as instructors, wanted to analyse in which ways the multimedia modality could promote a shift – *from* a concrete to an abstract transfer of concepts – in student learning. We used the *video-transfer-task* methodology, through which students were asked to explain (in business mathematics) linear algebra exercises and (in programming) some Java programming elements. Moreover, this instructional intervention provided a basis for understanding how we can better design our instruction, using multimedia videos as an additional learning output. In fact, from a diagnostic perspective these videos can help us recognise the correctness of the syntactic knowledge, the conceptual knowledge, and the strategic knowledge of Java programming and understand whether students are grasping the foundations of mathematics and programming. Thirdly, we believe that promoting the communication techniques of the so-called *procedural discourse* could be beneficial, considered that in the future, these BIT students will need to communicate and to collaborate with diversified clients and business professionals using information technology.

2 Theoretical Insights from Computing Education Research

Within the so-defined *computing education research*, based on the thoroughly and insightful literature review on student misconceptions and mistakes in introductory programming courses, Qian and Lehman [25] provided an insightful explanation about the types of difficulties students are usually encountering while learning the foundations of programming. As early as the 80ies, scholars in education and cognitive psychology started to focus on understanding the quality student's thinking while learning programming, including the analysis of misconceptions,

their "alternative conceptions", preconceptions and others. Specifically, in computer science education and programming, series of pivotal studies [1,2,16–19,26] were conducted to comprehend students' incomplete understanding when learning computer programming. These studies are still very insightful nowadays, while today's instructors are searching for the right balance between designing the pedagogical teaching strategies and structuring the programming teaching content.

One aspect is the possibility to offer to novice learners more *concrete-abstract transfer* instructional opportunities to novice learners [17]. In fact, Qian and Lehman [25] (p. 1) underscored that "many sources of students' difficulties have connections with students' prior knowledge." The most common difficulties were found in *encountering unfamiliar syntax, natural language, maths knowledge, inaccurate mental models, lack of strategies, programming environments and teachers' knowledge and instruction.* More specifically, scholars including Qian and Lehman [25] recognised three main types of students' difficulties, based on general types of programming knowledge: first, *syntactic knowledge* (referred to knowledge of language features, facts and rules); second, *conceptual knowledge* (meaning how programming constructs and principles work and internal computer mechanisms) and third, *strategic knowledge*, especially the application of syntactic and conceptual knowledge of programming to solve novel problems, such as tracing and explaining code [25] (p. 16; see also [2]).

Specifically, for Qian and Lehman [25] (ibid.) *syntactic knowledge* referred to frequent errors, namely: the mismatched parentheses, brackets and quotation marks; using irresolvable symbols, because failing to declare a variable before using it, missing semicolons, using illegal start of Java expressions, which happens, because of unfamiliarity with the Java expressions; or the mistakenly use of the assignment operator (=) instead of the comparison operator (==). These mistakes are easily detectable and straightforward to address. The *conceptual knowledge* denotes the errors in the basic mechanics of programming languages, which can also lead to students' misconceptions that are related to student's mental models of code execution and computer systems (examples are the concept of variable, variable scope, conditionals, the looping construct, program execution, and object-oriented programming concepts and principles). The *strategic knowledge* is defined as the "conditional knowledge in cognitive psychology" [25] (p. 6), which is denoting the expert level of knowledge on planning, writing, debugging programs for finding solutions to new problems by applying syntactic and conceptual knowledge.

The authors emphasised that "students' difficulties in strategic knowledge are highly correlated to their difficulties in syntactic knowledge and conceptual knowledge" [25] (p. 6), caused by the inexact mental model of reference. Therefore, it is not enough to know only syntactics and semantics is not enough to be a good programmer; being able to understand the problem to solve and how to decompose it, is also paramount. In fact, "When a novice programmer debugs a program, he or she typically reads and traces code in a local manner – line by line – without a holistic view about programming ... (in) debugging

novices usually is not fixing the error but rather comprehending the program and locating the error" [25] (p. 7). Furthermore, Qian and Lehman [25] (ibid.) reported on the factors that contribute to these difficulties, such as: task complexity and cognitive load, natural language, existing maths knowledge, flawed mental models, inadequate patterns and strategies, environmental factors, and teachers' instruction and knowledge. For example: novices, which are learning to program, might be not familiar with all the requested programming languages' syntax, forgetting some elements, such as parenthesis, braces, semicolons, etc., therefore, *task complexity and cognitive load* might be a common error condition. In addition, the own *natural language* knowledge might hinder the use of the specific programming language, as the terms do not match.

Another related aspect is the level of proficiency in English, which can contribute to the success in learning to program (an example is the use of *then* in the *if-then-else* programming construct). The *existing maths knowledge* is an indispensable precondition of being able to program. If the levels of prior maths knowledge are low, this situation could become a source of misconceptions and of potential errors. Other related factors are *flawed mental models*, which could cause incomplete and inaccurate execution of coding and the tracing of the code. Consequently, *inadequate patterns and strategies for solving problems* might develop, due to the fragmentary mental model instead of a well-organised structure. In such cases, patterns and strategies for solving programming problems might be applied, failing "to reason at an abstract level when comprehending, writing and debugging code ... the programming knowledge gained by novice programmers usually is not organised into meaningful patterns and the connections between pieces of knowledge are not well established." [25] (p. 9). The *environmental factors* are language features related to the programming environment, which are supporting the debugging activities. Finally, the *teacher's instructional and knowledge* is referring to a type of instructor that *teaches rules rather than reasons*, promoting the memorisation of syntactic knowledge, rather than understanding and reason, thereby – unintentionally – fostering the development of incorrect mental models, which are then difficult to change.

Qian and Lehman [25] underlined that the dissemination of new instructional approaches and tools to overcome these issues have been limitedly researched and studied. In fact, they warmly advised going beyond simply documenting and addressing students' difficulties and incorporate the *conceptual change theories* and also consider the pedagogical content knowledge (PCK). They refer to two theoretical perspectives within the *learning science* and the *scientific disciplines*, which apply this scientific knowledge in everyday life. These two theoretical perspectives are: *knowledge as theory and knowledge as elements* [24] (p. 351).

Historically, the *knowledge as theory* has been the predominant approach, and it refers to a student's knowledge accurately represented as a coherent unified framework of a *theory-liked knowledge*, involving a student's interpretation of subordinate models and ideas. It refers to the Piaget's concepts of assimilation and accommodation [24]. "If a learner's current conception is functional and if the learner can solve problems within the existing conceptual schema, then the

learner does not feel a need to change the current conception" [24] (p. 352). The learner needs to be dissatisfied with this initial conception, to abandon it and to accept a conceptual change, and often novices are not equipped to do so.

Knowledge as elements, on the other hand, refers to an ecology of quasi-independent elements, "where a combinatorial complexity of the system constraints students' interpretations of a phenomenon ... student's understanding in terms of collections of multiple quasi-independent elements" [7] (pp. 352–354) This definition was proposed by diSessa [7], "the knowledge structures of novices consist primarily of unstructured collections of many simple elements that he calls p-prims (phenomenological primitives) ... developed through a sense-of-mechanism that reflects our interactions with the physical world ... do not have a status of a theory ... are generated from a learner's experiences, observations, and abstractions of phenomena... have more exploratory power than conceptual framework theory." [7] (p. 355).

Summarising, through the *knowledge as elements*, composed by facts, narratives, concepts, and mental models at various stages of development, novices connect and activate these knowledge elements, based on the relevance of the situation. During the conceptual change process, which resembles a *piece-by-piece* progression of knowledge building, these elements find their way to learning and to making sense, compared to the *theory-liked knowledge* process. Therefore, these two models of knowledge define different ways of designing curricula, with the focus on helping students organise and reorganise their learning process.

3 Practical Insights from Cognitive Psychology

The *knowledge as elements* theory has some elements in common with the *generative learning* theory. In fact, according to Fiorella and Mayer [10], there are eight ways to promote generative learning and conceptual change in learning, which means when learners are actively making sense of the information to be learned. These are: learning by *summarising*, by *mapping*, by *drawing*, by *imagining*, by *self-testing*, by *self-explaining*, by *teaching to others*, and finally by *enacting*. The self-teaching technique, using videos to explain concepts in mathematics and in programming (as in our cases), has been scientifically proven to be very effective in previous research cases (see [9,12]).

Additionally, based on his four decades of research in the establishing his science of instruction, Mayer [16] (p. 121; see also [1]) defined *meaningful learning* the process by which the learner combines the new learning material with their *schema* (the already existing knowledge), through the process of assimilation. As early as the early 1980s, he asked himself whether concrete models – *advance organiser* – for novice programmers could support the meaningful learning of computer programming, while promoting more understanding than memorisation, because "the payoff for understanding comes not in direct application to the newly learned material, but rather in the transfer to new situations." [16] (p. 123). The same conclusions were reached by Soloway [26] (p. 852), when he differentiated the difficulties, novices had between *syntax* and *semantics* of programming. Based on the recent scientific contributions of Fiorella and

Mayer [11] and by Bétrancourt and Benetos [3], important criteria for designing instructional videos were considered and shared within the instructional design research community. An instructional video "is intended to help people learn targeted material ... it is a form of multimedia instruction" [11] (p. 465), including visual material (video) and verbal material (voice and/or onscreen text). Beneficial is the *segmenting*, being able to break down the multimedia presentation in meaningful segments, to provide learners with a control of their own learning process. Hindering aspects of instructional videos are *faces on screens*, *adding practice without feedback* and *inserting pauses* [11] (p. 465).

Soloway [26] stated that a program has two audiences: *the computer*, because the instructions in a program turn the computer into a mechanism that dictates how a problem can be solved and *the human reader*. Therefore, the programmer needs to have an explanation as to why the program solves the given problem, "learning to program amounts to learning how to construct mechanisms and how to construct explanations. In teaching programming and problem solving in general a key objective is to develop useful methods of abstraction. A hallmark of expertise is the ability to view a current problem in terms of old problems, so that solution strategies can be transferred from the old situation to the current situation." [26] (p. 853). Programming is a *design discipline*, which is producing an artefact that performs desired functions. Being an artefact leads to the concept of *mechanism*, which specifies a chain of actions that promotes some desired effects. In these processes *change* is the norm and not the exception; therefore, programmers need to provide the evidence of *how* and *why* the artefact was designed in a certain way, to ensure that the next programmer can effectively modify the artefact, if needed. This complexity also includes being able *to explain*, because programming processes are basically *mechanism and explanations*. Nevertheless, syntax and semantics are not enough; on the contrary, being able to break problems down into sub problems, and knowing their interconnections, this is also important for mastering problem-solving-programming tasks.

This exploratory video-based instructional intervention provided a basis for how we can better design our instruction, using these multimedia videos, as an additional learning output. In fact, from a diagnostic perspective, through these videos help us: a) to recognise the correctness of the syntactic knowledge, the conceptual knowledge, and the strategic knowledge of Java programming and to understand whether students are grasping the foundations of mathematics; b) to understand how students elaborate their contents from a *procedural discourse* communication, and how to teach this skill. Another aspect that we wanted to understand is the level of the so-defined *procedural discourse*, which the students intuitively used to create and to comment their videos.

Procedural discourse means *the how to do-to perform definite communication's procedures* [8,22]. According to Farkas [8] (p. 42) *procedural discourse* relates to written and/or spoken discourse guiding people in the performance of a certain task. *Procedural discourse* is more than a logical and structured information. Based on two recognised perspectives: a) human problem solving in the context of systems theory [23] and b) rhetoric and the source of credibility [6],

procedural discourse stems from the purposeful human behaviour, intended as the "*telling someone who is in one set of circumstances how to transition to another set.*" [8] (pp. 42–43). *Procedural discourse* implies four steps: 1. a desire state, which is the goal of the user; 2. the prerequisite state, as a condition for moving towards the goal; 3. the interim state, referring to the milestones and sub-goals to reach through our actions; and finally, 4. the unwanted states, which the user wants to avoid. This type of communication, *rhetorical* by nature, is procedure-based, and guided through tasks to accomplish. By analysing the way how students explained their programming and maths videos, we learn how to integrate this *procedural discourse* component, as a communication *added value skill* to be develop.

In the next section, we will provide more context about the two courses: its instructional design and the methodology used, students' results and how they approached the *exploratory video-based instructional intervention.*

4 The Two Courses: Programming and Business Mathematics 2

4.1 Participants

The two courses *Programming* and *Business Mathematics 2*, were held in the second semester of 2019. Two different classes were enrolled in the two courses. 17 students participated in the programming course, which covers the basics of Java programming, including object-oriented programming. In the mathematics course, 18 students were enrolled, learning the basics of linear algebra with an emphasis on computational approaches.

4.2 Method

Based on the instructor's motivation to drastically reduce the gap between students who are successful in programming and maths exams and those who fail, and based on the fact that programming is strongly related to maths, the instructors of this paper designed and applied the *exploratory video-based instructional intervention*, offering students the option of producing self-video-based explanations of programming and maths concepts, together with a flipped classroom intervention, using the *E.Tutorial*, which will be explained later. The foremost instructional aim was to understand how students can represent maths and programming concepts and exercises, using multimedia explanations and whether they would gradually undergo cognitive change through these video-based interventions *from a concrete to an abstract transfer* of programming and maths concepts [4,13,14].

In the programming class, a 4-step-model, a blended learning concept, called *E.Tutorial* and developed by the ETH Zurich, was used. First, students read short text documents of the most important definitions, concepts, and tasks (referred to the *SEE* part). Second, guided by an electronic tutorial, the concepts of the *SEE* part were applied (referring to the *TRY* part). Third, these

concepts were then applied individually to a problem-based task, transferring of the knowledge (the so-called *DO* part). Fourth, individual solutions of the *DO* part were presented and discussed in a face-to-face meeting with the instructor (the *EXPLAIN* part). Some topics were extended with traditional lecturing. It can be sustained that the exam's outcomes improved, compared to the previous years, with a failing rate of 30% and an average grade of 4.25 (maximum grade 6). Although a rather traditional teaching approach was used in the maths class, approximately half of the in-class time was invested to solve maths problems. The failing rate was 0% and an average grade of 5.5 (maximum grade 6) was achieved.

To keep track of their own learning experience and to engage students in this process, we asked them to produce the *exploratory video-based instructional interventions*. In the programming class, they explained a concept of Java programming and how to program it. In the mathematics class, they were asked to create three videos explaining how to solve three exercises. The creation of the videos was rewarded with additional learning points.

5 Results

5.1 The Programming Course

Using a post-questionnaire, students expressed their opinions about the entire course. The *E.Tutorial* was very much appreciated, because of the freedom it provided to study alone, at their own pace, introducing one Java concept at the time to promote understanding of specific concepts and methods. Correspondingly, self-learning was perceived correspondingly as a challenge for other learners as they appreciated the teaching option of experiencing a live-coding explanation on a screen in the classroom. In terms of exam performance, the programming and maths instructor appreciated the fact that the exam's outcomes improved, compared to the previous years, reaching an average grade of 4.25 (maximum grade 6). Certain exam questions were easily accessible, because students could have solved these even applying rote learning. However, one question was designed as a *transfer knowledge* one. Students received a description in a non-programming terminology of what a program should do, and they had to implement its functionality in the form of a method. They were aware of the exam format.

From the video production perspective, no student complained about producing a video with their own tools. On the contrary, 16 out of 17 students took the chance to earn learning points by creating a video, where they needed to explain a particular Java programming element and, if possible, to relate the abstract concept to a familiar and concrete object. The duration of these videos ranged from 30 s to 5 min. To analyse the videos, we used the three-part framework of Bétrancourt and Benetos [3]: the *representational approach*, the *cognitive approach*, and the *instructional approach* (see also Carliner's physical, cognitive, and affective information design framework [5]).

From the *cognitive information design approach*, the instructor analysed the presented content for correctness: 12 out of 16 were correct. For only five out of

16 there was a relation to concrete objects; and finally, only in three cases (out of 16) abstraction was mastered.

From the *instructional approach*, the programming videos showed the following features: two out of 16 introduced a definition; 13 out of 16 provided an explanation; 12 out of 16 introduced an example; two out of 16 used highlighting features, such as arrows, to let the audience follow the content presented within the videos; four out of 16 used rhetorical questions, like: *what is a variable*?

From the *representational approach*, six out of 16 addressed themselves to the audience, either with using *I*, *you* or *we*; 11 out of 16 used background music; students represented themselves, using animated-anonymous personas provided by the software they used; or they represented students in a classroom at the blackboard, personas with names, personas in stressful situations, like being a procrastinator and needing to deliver the programming homework, or creating a funny Christmas story, where a student needed to solve a programming problem, before being able to finally buy a gift for his girlfriend. Some students added slides with explanations from educational material, without commenting or elaborating on these contents. Some used their own words in explaining concepts. Some video-animations were completely disconnected, as if these presented their own mental schema, completely full of useless elements, disordered, superficial and with no context or relations to the instructional content.

Summarising, we were not fully satisfied with the video-based contents produced by the students. These were merely descriptive contributions, somewhat superficial, and not highly relevant to the course. Nevertheless, only the one contribution really surprised us. In fact, one out of 16 created a serious, but also intriguing and humorous video, featuring three students, becoming themselves examples of Java code and its explanation (Fig. 1). As mentioned previously,

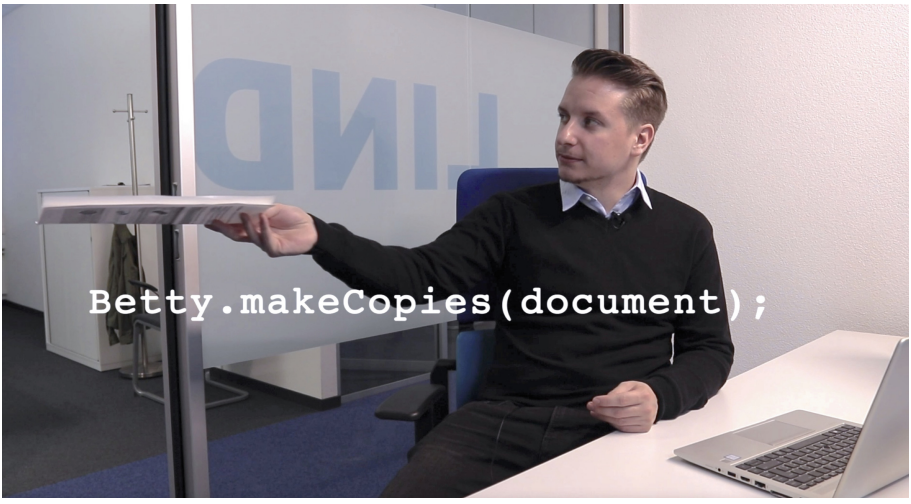

Fig. 1. A screenshot of an example video explaining abstract programming concepts on the basis of a concrete story and concrete objects.

according to Fiorella and Mayer [10], there are eight ways to promote *generative learning*, which means when learners are actively making sense of the to be learned information. These eight ways are: learning by *summarising*, by *mapping*, by *drawing*, by *imagining*, by *self-testing*, by *self-explaining*, by *teaching to others*, and finally by *enacting*. In this one case we considered this one video as a combination of *self-testing*, *self-explaining*, *teaching to others*, *enacting*, through a sort of theatrical representation, through which students became Java coding and Java programming concepts *actors and actresses*. The students, who engaged themselves in such a theatrical representation filmed on a video, were good students. One student started with no programming knowledge but was intrinsically motivated and matched with a student with considerable programming knowledge. Already in the class, they immediately selected the topic and started to sketch a story, based on clear instructional contents to communicate and to personally embed, representing Java objects, like people working in an office, as Fiorella and Mayer would define as "*embodied cognition*" [11] (p. 466).

This choice embodies how to transform abstract Java codes and Java concepts into concrete *actors and actresses*, finding relationships between Java codes and Java concepts. For example, in programming it is possible to copy an object; consequently, these students produced a copy of objects, presenting an assistant in an office represented by themselves. Additionally, they chose the topic: *objects and classes*, which is the most abstract and fundamental topic in the Java object-oriented programming language. These students applied a humorous approach to represents themselves as Java codes and Java concepts, covering from very basic to more advanced Java programming topics. Through the enactment, they tried to transform *an abstract programming concept to a concrete one, and promoting the transfer: from a concrete to an abstract*, through a serious storytelling theatrical representation [15]. As instructors we asked ourselves, if showing this excellent video example would help novice students to understand these programming concepts.

5.2 The Business Mathematics 2 Course

The final grades were showing that all students successfully passed the exams. 16 out of 18 students took the chance to earn learning points in creating three videos, where they needed to explain one exercise per video. Also, in this case, no student complained to produce a video with their own tools.

A total of 48 (16×3) videos were produced by the students, with approximately a total of 167 min distributed over 16 students, ranging from a maximum of 21 min to a minimum of three minutes for each student explaining the three exercises. The instructor analysed the videos considering the three categories: *syntactic knowledge*, *conceptual knowledge* and *strategic knowledge* [25]. In terms of these three categories, all the 48 videos were correct. But a *no audio explanation in own words hypothesis* was suggested. In fact, a total of three student videos were produced without audio explanation. The instructor was not able to understand whether the procedures shown in these videos were correct or not.

In the good videos, students showed how to work with clear, orderly procedural writing, combined with clear explanations. For example, one student copied from the book all the elements to solve the exercises, giving context to the exercises before solving these and defining the work to be done, and afterwards explaining in their own words the beginning of the problems to be solved. Another provided solutions with their own words; someone else added their own insights and their own context to arrive at the correct result. Other students were able to link the solutions to the formulas and theorems on a formula sheet provided by the instructor during the exam. Some students enhanced the explanation with own content (summarised on cards), which was not explained in class, other used predefined screens and one student also corrected himself during the explanation, reaching the correct solution even if the exercise was challenging. Some spoke very freely and confidently. Others suggested hints on how to solve problems, using alternative strategies.

The less good videos reflected the behaviour of certain students in class. For example, distracted students produced videos without context or explanation, not mentioning the type of exercise he/she was dealing with, showing a lack of involvement, adding music to the video explanation instead of thoroughly explaining the exercise. When music instead of oral explanation was used, it became a difficult task for the instructor to verify whether the video-content was clearly understood or not.

It is interesting to note that the maths explanations required an abstract-to-concrete shift, since abstract concepts had to be defined first and then elaborated through concrete cases, e.g., calculations with concrete values. This shift was only recognisable if students used their own words to explain what they did and if they provided context or their own insight.

According to the *adagio* of the media philosopher Marshall McLuhan "The medium is the message" [20,21], he provocatively proposed to focus on the medium and not just the content. The medium impacts the content (the message) to be delivered, also becoming also the message itself. The videos were very insightful from an instructional point of view, because students were able to produce their videos without any technical difficulty. No one showed their own face, but they worked with their own voice. This way of creating the videos mirrored their own behaviour and personality features as students in classroom.

6 Limitations and Discussion

We are aware that our exploratory studies present several limitations. First, we had two experimental groups for both courses: Students were exposed to the same amount of content. Organising a control group was beyond the scope of the present study. Second, the two classes (*Programming* and *Business Mathematics 2*) were not the same ones. For a next educational experiment, we plan to work with a class that covers mathematics and programming instructional contents, to understand how students may combine the two disciplines. Third, the production of the videos was an optional and non-mandatory task, designed to help students with greater difficulties.

Through these videos, we learnt that the *exploratory video-based instructional intervention* could greatly improve the understanding of where students are in their learning processes. We do believe that the *conceptual change theories*, especially the *knowledge-per-element* could promote a beneficial aspect in designing learning activities within the curriculum. For a next round of instructional experiments, a suggestion is using a content analysis methodology, considering the following analysis lenses: a) the three levels of explanation, according to *the representational approach*, *the cognitive approach*, and *the instructional approach*, developed by Bétrancourt and Benetos [3], b) the logical and rhetorical construction of procedural discourse, designed by Farkas [8] and finally c) the Morain and Swarts [22] research questions, which are: *With which convectional forms of instructional discourse, do students use sound, text, still images and moving images? How do these modal forms of content help the viewers to access, understand, stay engaged with the instructional message? How do these usages of content differ in these videos that other students can rate good, average, poor? If these good videos were shown to weaker students, would they represent helpful learning tools for them? If yes, why, and how? How could we measure this additional video-enhanced understanding (the shift from concrete to abstract)?*

Additionally, we plan the creation of a multimedia accessible collection of all students' self-produced videos to be stored on the Moodle platform. The instructor will provide "an assessment grid", together with a vote system. During the last lesson of the course, the class will be looking at all the videos, voting these and give students the chance to promote more discussion around programming and maths concepts.

Summarising, through the *knowledge-as-elements*, composed by facts, narratives, concepts, and mental models at various stages of development, novices connect and activate these elements, based on the relevance of the learning situation. During these conceptual change processes, which may be perceived as a piece-by-piece progression of knowledge creation, first-year students could find their way to enhance their programming and mathematics understandings and skills, improved through the procedural discourse communication skill, which represent another vital competence for becoming a competent BIT professional. As mentioned, for Fiorella and Mayer [11], there are eight ways to promote *generative learning*. Through the *exploratory video-based instructional intervention*, students experienced four of the eight strategies (*summarising, self-testing, teaching to others, and enacting*). Providing students with more opportunities to learn by using more media that is spiced with creativity could make programming and abstract mathematics concepts more accessible to novice students.

Acknowledgement. We are grateful to the students of the programming and mathematics classes for their work and their active engagement in helping us to enrich our didactics and teaching strategies.

References

1. Adelson, B.: Problem solving and the development of abstract categories in programming languages. Memory Cognit. **9**(4), 422–433 (1981). https://doi.org/10.3758/BF03197568
2. Bayman, P., Mayer, R.E.: Using conceptual models to teach basic computer programming. J. Educ. Psychol. **80**(3), 291–298 (1988). https://doi.org/10.1037/0022-0663.80.3.291
3. Bétrancourt, M., Benetos, K.: Why and when does instructional video facilitate learning? A commentary to the special issue "developments and trends in learning with instructional video". Comput. Hum. Behav. **89**, 471–475 (2018). https://doi.org/10.1016/j.chb.2018.08.035
4. Bloom, B.S., Engelhart, M.D., Furst, E.J., Hill, W.H., Krathwohl, D.R.: Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook 1: Cognitive Domain. David McKay Company, New York (1956)
5. Carliner, S.: Physical, cognitive, and affective: a three-part framework for information design. Tech. Commun. **47**(4), 561–576 (2000). https://www.jstor.org/stable/43748975
6. Coney, M.B., Chatfield, C.S.: Rethinking the author-reader relationship in computer documentation. ACM SIGDOC Asterisk J. Comput. Documentation **20**(2), 23–29 (1996). https://doi.org/10.1145/381815.381826
7. diSessa, A.A.: Toward an epistemology of physics. Cognit. Instr. **10**(2/3), 105–225 (1993). https://www.jstor.org/stable/3233725
8. Farkas, D.K.: The logical and rhetorical construction of procedural discourse. Tech. Commun. **46**(1), 42–54 (1999). https://www.jstor.org/stable/43088601
9. Fiorella, L., Mayer, R.E.: Role of expectations and explanations in learning by teaching. Contemp. Educ. Psychol. **39**(2), 75–85 (2014). https://doi.org/10.1016/j.cedpsych.2014.01.001
10. Fiorella, L., Mayer, R.E.: Eight ways to promote generative learning. Educ. Psychol. Rev. **28**(4), 717–741 (2016). https://doi.org/10.1007/s10648-015-9348-9
11. Fiorella, L., Mayer, R.E.: What works and doesn't work with instructional video. Comput. Hum. Behav. **89**, 465–470 (2018). https://doi.org/10.1016/j.chb.2018.07.015
12. Hoogerheide, V., Deijkers, L., Loyens, S.M.M., Heijltjes, A., van Gog, T.: Gaining from explaining: learning improves from explaining to fictitious others on video, not from writing to them. Contemp. Educ. Psychol. **44**, 95–106 (2016). https://doi.org/10.1016/j.cedpsych.2016.02.005
13. Krathwohl, D.R.: A revision of bloom's taxonomy: an overview. Theory Pract. **41**(4), 212–218 (2002). https://doi.org/10.1207/s15430421tip4104_2
14. Krathwohl, D.R., Anderson, L.W.: A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. Longman, New York (2009)
15. Lugmayr, A., Sutinen, E., Suhonen, J., Sedano, C.I., Hlavacs, H., Montero, C.S.: Serious storytelling-a first definition and review. Multimed. Tools Appl. **76**(14), 15707–15733 (2017). https://doi.org/10.1007/s11042-016-3865-5
16. Mayer, R.E.: The psychology of how novices learn computer programming. ACM Comput. Surv. (CSUR) **13**(1), 121–141 (1981). https://doi.org/10.1145/356835.356841
17. Mayer, R.E.: Models for understanding. Rev. Educ. Res. **59**(1), 43–64 (1989). https://doi.org/10.3102/00346543059001043

18. Mayer, R.E.: How to be a Successful Student: 20 Study Habits Based on the Science of learning. Routledge, New York Oxon (2018). https://www.routledge.com/How-to-Be-a-Successful-Student-20-Study-Habits-Based-on-the-Science-of/Mayer/p/book/9781138319851
19. Mayer, R.E., Fay, A.L.: A chain of cognitive changes with learning to program in logo. J. Educ. Psychol. **79**(3), 269–279 (1987). https://doi.org/10.1037/0022-0663.79.3.269
20. McLuhan, M.: Understanding Media. Routledge, London (1964)
21. McLuhan, M., Fiore, Q.: The Medium is the Massage: an Inventory of Effects. Penguin Press, London (1967)
22. Morain, M., Swarts, J.: YouTutorial: a framework for assessing instructional online video. Tech. Commun. Q. **21**(1), 6–24 (2012). https://doi.org/10.1080/10572252.2012.626690
23. Newell, A.: Reasoning, problem solving, and decision processes: the problem space as a fundamental category. In: The Soar Papers (vol. 1): Research on Integrated Intelligence, pp. 55–80. MIT Press, Cambridge, MA, USA (1993). https://dl.acm.org/doi/abs/10.5555/162580.162585
24. Özdemir, G., Clark, D.B.: An overview of conceptual change theories. Eurasia J. Math. Sci. Technol. Educ. **3**(4), 351–361 (2007). https://doi.org/10.12973/ejmste/75414
25. Qian, Y., Lehman, J.: Students' misconceptions and other difficulties in introductory programming: a literature review. ACM Trans. Comput. Educ. (TOCE) **18**(1), 1–24 (2017). https://doi.org/10.1145/3077618
26. Soloway, E.: Learning to program = learning to construct mechanisms and explanations. Commun. ACM **29**(9), 850–858 (1986). https://doi.org/10.1145/6592.6594