A content analysis of SOLO-levels in different computer programming courses in higher education

Fojcik, M. K.[1], Fojcik, M.[2], Sande, O.[2], Refvik, K.A.S.[1], Frantsen, T.[3] and Bye, H.S.[3]

[1]*Volda University College - Department of Science and Mathematics*
[2]*Western Norway University of Applied Sciences - Department of Computer science, Electrical engineering and Mathematical sciences*
[3]*Volda University College – Digital competences in learning and teaching*

Abstract

The dynamic development of technology and the labour market changes the requirements of today`s education and the dissemination of knowledge. Information technologies (IT) and digital competencies (DC) are no longer knowledge just for the few that study Computer Science (CS), but it has become a part of common knowledge for every citizen. By using content analysis, this article will examine the developed content of two different "introduction to programming" courses from two different higher education institutions. Both institutions introduce programming to students outside of CS. This study aims to describe how the developed content of these courses aims to reach the different levels of learning outcomes, by using the framework Structure of the Observed Learning Outcome taxonomy (SOLO-taxonomy) developed by Biggs and Collis (1982). The results of the study show that introduction to programming courses in different professions have a different understanding of what programming is, or what it consists of. The courses about "introduction to programming" are planned and executed within its fields, which gives the students a different perspective on what programming is, compared to the average IT or CS course. This means that the term "good programming skills" is different for a teacher, engineer, or computer scientist because of their unique goals and motivations for why they learned to program in the first place.

Keywords: Programming skills, digital competencies, 21st century skills, didactics in IT education, introduction to programming

Introduction

The changing situations in our society (digitalization, modernity, 21st century, pandemic, etc.) demand rethinking teaching- and learning designs in all levels of education. According to the reports 2/3 of the students-nowadays are going to work in jobs that do not exist yet (World Economic Forum, 2016). From an educational perspective, the rapid changes in technology and the development of new services in modern life creates a need for new competencies related to information technology (IT) and computer science (CS). It is necessary to develop skills and gather knowledge about practical applications of digital tools, both in private and in professional situations. The career-choices may change with time; therefore, it is important to teach the students to be less dependent on specific tools and solutions, and more focused on problem-solving and the structure of digital technology.

That brings a new perspective into education, creating topics and structures that have not existed before. The term 21st-century skills have been developed to describe creativity, innovation, critical thinking, metacognition, collaborative skills, problem-solving skills, and information- and digital literacy (Griffin and Care, 2015). The idea is not just to teach new skills, but to teach the students to acquire knowledge in more self-direct learning so that the students are less dependent on memorization and reiterating someone else's thoughts and more inventive and constructive in creating own ideas.

Many governments decided that the way to supply pupils and students in all levels of education with IT-knowledge and the necessary development of DC is to teach them

computer programming (Bocconi, Chioccariello & Earp, 2018; Mannila & Nordén, 2017). The solution is to spread programming skills to a variation of professions to develop DC in modern citizens. Therefore, many universities have different courses that introduce programming for students in different fields. There is programming for kids, engineers, mathematicians, physicists, bio technicians, teachers, and many more. Not to mention that most courses teach only one programming language, instead of focusing on the concepts of programming.

Looking at how the school system is implementing programming can be beneficial to how other professional studies or pure programming studies could implement programming. More specifically by using time both in the beginning and during the studies on how one should think when learning to program (not focused on language, but concepts, processes, etc.) so that the students get time to adapt to that way of thinking (Computational Thinking).

This article will analyse two courses in introductory programming adapted for different studies. One course from Western Norway University of Applied Sciences (HVL) is about introducing programming to a full-time undergraduate Bachelor engineering-program in automation and robotics, while the other from Volda University College (HVO) is about introducing programming to a part-time one-year postgraduate program in programming for teachers that are going to teach programming in primary or secondary school (K1-K12). On the outside, it may look that these study-programs have nothing in common than programming. Therefore, the authors will try to analyse the content of the courses and compare them to each other and the level of understanding the students can develop in the SOLO-taxonomy by Biggs and Collis (1982).

The research question for this study is: *Can an introduction to programming-course for non-computer scientists teach higher levels of observed learning outcomes in programming?*

This paper is divided into six sections. First is a theoretical background then a description of the methodology used in this article. Followed by two sections of results, one section is a comparison of the courses with criteria of IT-programming, the second section presents a comparison of the courses to own professions. Then the paper discusses central topics and challenges and concludes the findings of the study.

Theoretical background and situation today

Programming, more specifically coding that has till now been a central part of IT and CS has been divided and separated to be introduced in different fields and professions, while other methods and structures from IT and CS have not been shared or implemented in other studies. This creates a gap between what IT and CS field consider programming to be and what everyone else thinks it programming is. In 2006 the computer scientist Jeanette Wing introduces the term "Computational Thinking" (norsk: *Algoritmisk tenkning*), as a description of fundamental skills like reading and writing, and it is defined as "an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing" (Wing, 2006, 2008, p.3717). In other words, Computational Thinking is a collection of significant skills "necessary for applying the tools of computer science to understand the world around us" (Selby, 2014,

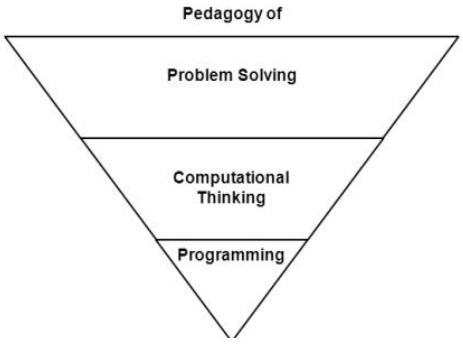

Figure 1: Conceptual framework (Selby, 2014, p.19).

p.1). This concept is based on thought process, application methods, and logical patterns that IT and CS, therefore, it does not need technology to be implemented (Bocconi, Chioccariello, Dettori, Ferrari & Engelhardt, 2016). The term Computational Thinking is not explicitly used in government documents, but it can be divided into concepts like abstraction, algorithmic thinking, automation, decomposition, and generalisation (Bocconi, Chioccariello & Earp, 2018).

For educational purposes, there were constructed different frameworks for how can programming skills help students and pupils to learn better. Selby (2014) proposes a conceptual framework where programming skills is a term within Computational Thinking, which also is within Problem Solving Skills (Figure 1.). This gives a structure on how to place programming to national curricula in school subjects. Another propose of structuring Computational Thinking is divided into three dimensions and connecting it to programming terms, which are: Computational concepts, Computational practices, and Computational perspectives (Figure 2.) (Lye & Koh, 2014).

Dimension	Description	Examples
Computational concepts	Concepts that programmer use	Variables Loops
Computational practices	Problem-solving practices that occurs in the process of programming	Being incremental and iterative Testing and debugging Reusing and remixing Abstracting and modularizing
Computational perspectives	Students' understandings of themselves, their relationships to others, and the technological world around them	Expressing and questioning about the technology world

Figure 2: Description of Computational Thinking dimensions (Lye & Koh, 2014, p.53).

In Norway, the government developed a new curriculum in 2019 in every subject in primary and secondary school, to update and modernize what the pupils and students are going to learn (LK20). One of the biggest changes was to introduce programming for everyone. From 2020 the subject's mathematics, science, music, and art and handwork, have the responsibility to teach the pupils how to program, and how to relate programming knowledge both to the subject and to their digital competencies. Research like Mathew, Malik, and Tawafak (2019) showed that learning computer programming can help students to achieve knowledge, skills, and experience significant for developing digital skills. This creates a new interest in programming in different fields and professions.

In many courses in higher education today, it is up to the teacher to introduce programming for the students and develop the students' IT-knowledge. How this is done from one course to the next depends on the teachers' competencies in CS. It is the teacher that chooses the structure, the curriculum, and the dissemination for his or her course. If the content has some structures of programming or coding, it may be presented as an introduction to a programming course. This creates a variation of what the introduction to programming could contain. There are a few general ideas, but no official standard that specifies how much a non-computer scientist should know about programming. Or how many of DC can be developed through programming.

Methodology

This paper aims to analyse the content of two "instruction to programming" courses and will make use of content analysis as a method to answer the research question. By quantitating the content for these programming courses, we would see what level of learning outcome that is intended for each course (Bryman, 2016). The different levels of learning outcomes that are intended are well defined in the coding manual for the analysis. The coding manual is developed before starting the analysis and builds on the Structure of the Observed Learning Outcome taxonomy (SOLO-taxonomy) (Biggs and Collis, 1982; Biggs, 2012). This taxonomy is intended to use for evaluating student's outcomes in a course and should be a crucial tool for lecturers to develop their courses. Therefore, the coding was developed manually based on this taxonomy, and the article aims to analyse which levels of learning outcomes are intended in the course.

The analysis and comparison of how non-scientist tech higher levels of observed learning outcomes in an introduction to programming course, is divided into two sections. Firstly, the article will present criteria combining SOLO-taxonomy levels with an introduction to programming, which is a way that a "pure" programming course for IT and CS would be. This analysis will show how much IT-programming is developed in the chosen non-computer scientists' courses. Then a second analysis is conducted, where there are given new criteria. The new criteria are developed by combining SOLO-taxonomy levels with the professions or study programs on which the courses are integrated into. In this way, there will be a comparison of the two courses both to an IT-criteria and to the profession (automation and teacher education) from which it stands.

Content analysis

According to Bryman (2016, p. 285) content analysis is «an approach to the analysis of documents and texts that seeks to quantify content into determined categories and in a systematic and replicable matter». This paper will present a content analysis of plans of the semester, teachers' presentations and notes, the curriculum in the course, and the assignments and activities that are given to the students during the semester. This analyse uses previously determined levels of Biggs and Collis taxonomy (1982) as categories in systematically analyse of the content. The theme of the lectures is shortly described and then compared to the description of each level of observed learning outcome.

SOLO-taxonomy

SOLO stands for: Structure of the Observed Learning Outcome, and it is a taxonomy introduced to evaluate students learning first presented by Biggs and Collis (1982). The taxonomy divides different stages of natural growth, learning, and skill development, and describes them in simple, yet general terms. The visual model comes from an extended model of SOLO taxonomy presented by Biggs (2012) (Figure 3.). This model presents 5 phases or levels of observed learning outcome:

1 – Pre-structural phase – No learning is observed
2 – Uni-structural phase – Few simple procedures are observed
3 – Multi-structural phase – Some procedures are observed, but not in relation to one another
4 – Relational phase – Many procedures with connections between them are observed
5 – Extended abstract – Procedures are observed and applied in different topics and situations

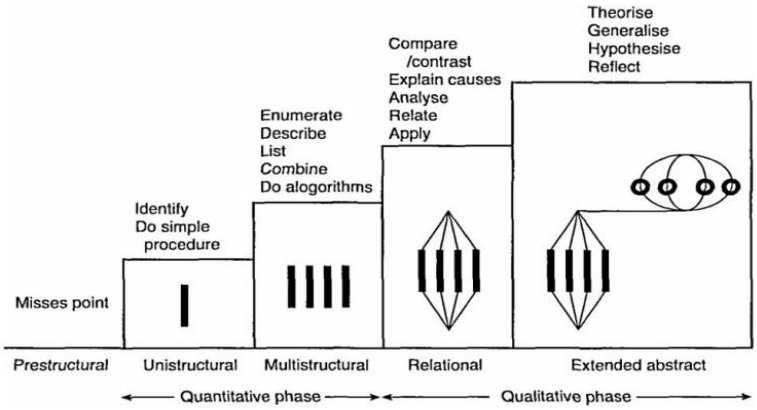

Figure 3. A hierarchy of verbs used in forms of curriculum objectives (Biggs, 2012, p.48).

Description and analyse

Through content analysis, a description of both courses was made. The analysis covered the semester plan, curriculum, dissemination, and form for presentation of the lecture, activities, assessments, and exercises given to students during the course.

Description of the courses – HVO

The first course presented in this study is from postgraduate teacher education at Volda University College (HVO). The students are simultaneously working in primary or secondary schools and part-time studying programming. Therefore, the whole program is conveyed through the Internet-based Learning Management System /Digital Learning Platform (Canvas). One of the demands of this program is that the students have pupils they can teach during the semester. This program has a total of 15 ECTS divided into part 1 (fall semester) and part 2 (spring semester). In this paper, only the content of the first part is analysed. The curriculum is a collection of articles, book-chapters, and films explaining the fundamental aspects of programming with children.

The students have many described learning outcomes, but the main purpose of the course is to introduce experienced teachers to new technology and a different way of thinking. Many assessments and activities during the course are developed to let the student try out and reflect on different aspects of programming that can be beneficial for their own pupils. Therefore, the students have some flexibility to adapt the curriculum to their own pupils. For example, a K2 teacher needs to focus more on precise instructions and algorithmic thinking, and how to separate right from left, while a K10 teacher needs to find software that can support pupil's inquiry of science and mathematics. The content of the course is created in such a way to introduce the concepts and methods of programming rather than introduce syntax and structures of a program.

Description of the courses – HVL

The second course presented in this study is an undergraduate course in programming for beginners developed for a three-year automation engineering program at Western Norway University of Applied Sciences (HVL). This course is introduced to students in the fall semester in their first year. The course has a total of 10 ECTS, but the students have two more subjects simultaneously, and then more programming in their second and third year. The course covers the basics of programming structures, which are going to be further developed in future subjects. The curriculum is based upon one book, which is not specialized for the automation line.

Automation engineers have some description of learning outcomes, but many of those are about skill development and is integrated into more than one course. The students need to learn the highlights of controllers, databases, and other different machines and equipment. Using different equipment, the students are learning to connect hardware and software in an industrial setting. In automation, the main goal is to create real-time secure, robust programs. In this field, a faulty program is not an option. Faulty programs controlling dangerous equipment is potentially life-threatening.

SOLO-taxonomy based on general IT knowledge – criteria

To compare how much programming is presented in the two courses, and what level of learning outcome is observed in students, the criteria for comparing IT and CS knowledge with SOLO-taxonomy have been developed in Table 1.

Level	Name	Description
5	Extended Abstract	Develop the skill of creating, testing, and operating programs, making and distributing class, libraries for multiple uses, cooperation and collaboration with other programs and users.
4	Relational	Create a working program, divided into different elements (class, structure, etc.) full debugging, collaboration and cooperation with the user, saving state, file handling, network, database, etc.
3	Multi-structural	Sophisticated elements in coding, combining several different actions, user interface etc.
2	Uni-structural	Simple skills, using variables, comparing elements, simple loops. Everything in one file, with no dividing in sections or class or structures, etc.
1	Pre-structural	Barely anything or just a very general or divided knowledge – wrong assumptions and unconnected information, etc.

Table 1: Description of observed learning outcomes in an introduction to programming for IT-students.

Analysis using general IT criteria for HVO

Observed Learning Outcome of IT and CS programming are compared to the content of a programming course in teacher education in HVO. Table 2 presents the linear view of lessons with themes presented in the course. The first and third column shows what kind of themes and activities were given to the students. In the second and fourth columns, the analyse of SOLO-phases or SOLO-levels is presented with the criteria from Table 1.

Theme	IT SOLO-level	Activity	IT SOLO-level
Introduction, Programming in schools, Computational thinking	1, 2	Trying “Hour of Code” Reflection about programming in school with a historic overview	1, 2
Scratch – visual language, Sequence and easy loops, Unplugged programming, Scratch with pupils	1, 2	Trying Scratch, Create projects with pupils Reflection on how to work with Scratch in class	1, 2
Micro:bit – visual language, Electricity, circuits, voltage,	1, 2	Trying Micro:bit, Share your experience, Paper “Planning, doing and evaluating	1, 2

Integrating programming in subjects in school		programming-lesson with own pupils" - 1000 words	
Text programming language – Python, Debugging, Differences between visual and text language	1, 2	Trying Python, Solve debugging exercises, Discuss different programming language (Scratch-Micro:bit-Python)	1, 2
Evaluating pupils work Common mistakes in text-programming, Learning design, Backward by design	1, 2	Own program in virtual- and text programming language and paper explaining it - 1000 words, Reflect on different approaches in programming	1, 2

Table 2: Description of the HVO course content in relation to SOLO-taxonomy for IT-students.

Analysis using general IT criteria for HVL

Table 3 presents the linear view of lessons, laboratories, and activities that were presented during the course. In the second and fourth columns, the analyse of SOLO-phases or SOLO-levels is presented with the criteria from Table 1. Also, the comparison to what kind of Observed Learning Outcome of IT programming can be found in automation engineering students taking this course from HVL.

Theme	**IT SOLO – level**	**Activity**	**IT SOLO - level**
Lectures			
Programming environment	1, 2	Programming Environment	1, 2, 3
Variables and operations	1, 2	Presentation of variables and operations	1, 2
If-sentences	1, 2	Conditions	1, 2
Text or Graphis User Interface	1, 2, 3	Cooperation with user, correct and understandable information or dialog	1, 2, 3
Loops	1, 2	Repetition of actions	1, 2
Tables	1, 2, 3	Grouping of variables	1, 2, 3
Methods	1, 2, 3, 4	Structure of program, dividing into smaller (independent) units	1, 2, 3, 4
Serial port (RS232/USB)	1, 2	Connection with real world, reading and sanding real measurements	1, 2
File operations	1, 2	Saving and reading of information	1, 2
Laboratories			
Operations	1, 2	Experiences with variables and operations	1, 2
Operations	1, 2, 3	Experiences with variables and operations	1, 2, 3
Easy loops	1, 2	Repetition of operations	1, 2
Loops and if-sentence	1, 2, 3	Conditions, multi conditions	1, 2, 3
Easy operations with serial port	1, 2, 3	Creating and modification of easy programs using real measurements	1, 2, 3, (4)
Easy operations with serial port	1, 2, 3	Creating and modification of easy programs using real measurements	1, 2, 3, (4)

Table 3: Description of the HVL course content in relation to SOLO-taxonomy for IT-students.

SOLO-taxonomy based on criteria from professions and fields

There are different aspects of the professions that introduce programming that is limiting or challenging the structures and methods used in IT and CS. To get a better

understanding of how much a profession influences a programming course, a new analysis was conducted. This time the criteria for analysis connects SOLO-taxonomy to a description of the profession or field where the course is a part of. There are different aspects and elements that are considered by creating different levels and expectations from programmers in these professions.

Analysis using teacher-profession specific criteria for HVO

The main purpose of a teacher is to guide the social process of learning. Therefore, the teachers' goal is not the dissemination of technical knowledge, but to motivate, interest, explain, and support pupils learning. A teacher teaching programming in primary school does not need to know all the structures of objected programming, because this knowledge is not directly usable for the grade the teacher is teaching. Rather than focusing on technical aspects, a teacher in primary school should know how and when to teach programming so that this knowledge can support children's learning and development. Table 4 shows the criteria for SOLO-taxonomy for introducing programming developed in this study for teacher-profession.

Level	Name	Description
5	Extended Abstract	Show pedagogical, didactical, and content knowledge of programming, the skill to use simple programs, understanding and explaining computational structures, supporting computational thinking in students/pupils in all levels of education, etc.
4	Relational	Explaining why the program works, reading and understanding others code in order to debug programs, etc.
3	Multi-structural	Independent writing of the algorithms and create simple working programs, explaining efficiency and accuracy of student's own work, etc.
2	Uni-structural	Independent writing of simple commands/sequences, using simple loops and/or if-sentences, etc.
1	Pre-structural	Barely anything or just a very general or divided knowledge – wrong assumptions and unconnected information, etc.

Table 4: Description of observed learning outcomes in the introduction to programming for students in teachers-profession.

Table 5 presents the again the structure of the course from HVO, but this time the analysis compares the course content to the criteria from Table 4, describing the teacher profession.

Theme	teacher SOLO-level	Activity	teacher SOLO-level
Introduction Programming in schools Computational thinking	1, 2, 3, 4	Trying "Hour of Code" Reflection about programming in school with a historic overview	1, 2, 3,
Scratch – visual language Sequence and easy loops Unplugged programming Scratch with pupils	1, 2, 3, 4, 5	Trying Scratch Create projects with pupils Reflection on how to work with Scratch in class	1, 2, 3, 4,
Micro:bit – a visual language Electricity, circuits, voltage Integrating programming in subjects in school	1, 2, 3, 4, 5	Trying Micro:bit Share your experience Paper "Planning, doing and evaluating programming-lesson with own pupils" - 1000 words	1, 2, 3, 4,

Text programming language - Python Debugging Differences between visual and text language Pupils working with Python	1, 2, 3, 4, 5	Trying Python Solve debugging exercises Discuss different programming-language (Scratch-Micro:bit-Python)	1, 2, 3, 4,
Evaluating pupils work Common mistakes in text- programming Learning design Backward by design	1, 2, 3, 4, 5	Own program in virtual- and text programming language and paper explaining it - 1000 words Reflect on different approaches in programming	1, 2, 3, 4,

Table 5: Description of the HVO course content in relation to SOLO-taxonomy for students in teacher-profession.

Analysis using automatic-engineer-profession specific criteria for HVL

Automatic engineers need to have both the technical knowledge of programming, and practical experience with different equipment. In this profession, there is no place for error, and the internal and external security of a program is central for the development of such an engineer. Table 6 shows the criteria developed for automation engineers based on SOLO-taxonomy and the engineering profession.

Level	Name	Description
5	Extended Abstract	Model of software as a model of real action/automation, libraries, interfaces, analogies between real-world and software
4	Relational	Dividing action into smaller, repeatable elements, reusability, safety, friendliness of the software
3	Multi-structural	Repetitions of actions, more complicated elements, user interface, error-handling
2	Uni-structural	Understanding the idea of planning and operations order, the definition of variables (connected with real measurements), and simple operations on them
1	Pre-structural	Barely anything or just a very general or divided knowledge – wrong assumptions and unconnected information, etc.

Table 6: Description of observed learning outcomes in the introduction to programming for automation-engineering-students.

Table 7 presents the structure of the course from HVL with the analysis and comparison of the course content to the criteria from Table 6, describing the automation-engineer profession.

Theme	automation SOLO-level	Activity	automation SOLO-level
Lectures			
Programming environment	1, 2	Programming Environment	1, 2
Variables and operations	1, 2	Presentation of variables and operations	1, 2
If-sentences	1, 2	Conditions	1, 2
Text or Graphis User Interface	1, 2, 3	Cooperation with user, correct and understandable information or dialog	1, 2, 3
Loops	1, 2,	Repetition of actions	1, 2
Table	1, 2, 3	Grouping of variables	1, 2, 3
Methods	1, 2, 3, 4	Structure of program, dividing into smaller (independent) units	1, 2, 3, 4
Serial port (RS232/USB)	1, 2, 3, 4	Connection with real world, reading and sanding real measurements	1, 2, 3, 4

File operations	1, 2	Saving and reading of information	1, 2, 3
Laboratories			
Operations	1, 2	Experiences with variables and operations	1, 2
Operations	1, 2, 3	Experiences with variables and operations	1, 2, 3
Easy loops	1, 2	Repetition of operations	1, 2
Loops and if-sentence	1, 2, 3	Conditions, multi conditions	1, 2, 3
Easy operations with serial port	1, 2, 3, 4	Creating and modification of easy programs using real measurements	1, 2, 3, 4
Easy operations with serial port	1, 2, 3, 4, 5	Creating and modification of easy programs using real measurements	1, 2, 3, 4, 5

Table 7: Description of the HVL course content in relation to SOLO-taxonomy for automation-engineering-students.

Discussion

In the courses named in this article, the students are not going to be programmers, but they need to know how to use programming as a tool for different aspects of their profession. This creates different approaches to programming and to introducing programming that is challenging to compare. There is a tendency in the results that the higher education institutions want the students to learn both theoretical and practical knowledge about programming, but the analysis of the content of the courses shows that the practical assessments the students do are in many cases not adequate with the theoretical knowledge. In other words, the level of observed learning outcomes in the lesson/theory that was presented for the students, and the activities and assessments the students were asked to do while working on this theme differ. This leads to the assumption that the students understand the theory at a higher level than the practical exercises.

Another result showed in this study is that the students in non-IT or non-CS programs learn a different kind of programming. In some cases, the difference is rather small (automation-engineers) and mostly consist of different priorities and emphasis on different parts of the program, data presented in Diagram 2. While in other cases, like the teaching profession, the programming-theme is highly selected, separated, and used for a variety of purposes that does not need to be related to IT or CS, data presented in Diagram 1.

This result shows a worrying aspect of programming in today's society. The aspect is that programming has become a tool taught and used differently in different fields. Therefore, a teacher teaching programming in primary school, a teacher teaching programming in higher education, a teacher teaching programming to engineers or bio technicians or any other profession, all of those teachers need to have different pedagogical and didactical competencies, even if all of them teaches programming. The programming skills in the future may become as essential as reading and writing. And maybe someday the digital competencies would be much more detailed ether to a certain programming language or to general methodologies and aspects used in programming.

The challenge today is how to present programming in such a way that supports understanding programming structures without becoming a programmer (more experience, individual assessment, peer review of students' assessment, customize tasks).

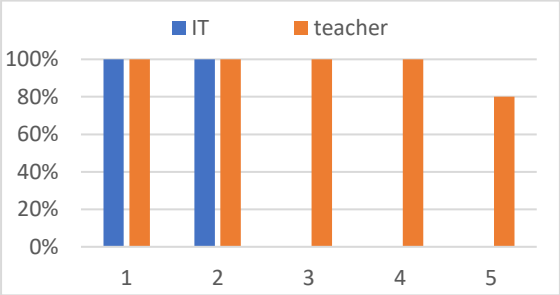

Diagram 1: Description of how well the course from HVO fits the IT and teacher criteria of SOLO-taxonomy.

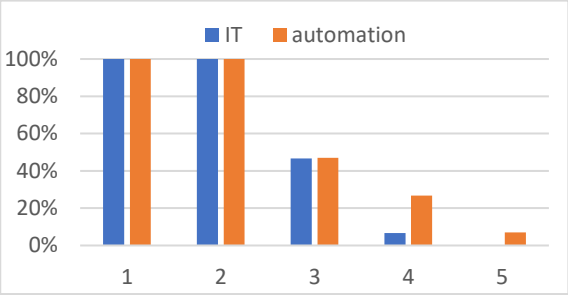

Diagram 2: Description of how well the course from HVL fits the IT and automation criteria of SOLO-taxonomy,

The main difference in these courses is that HVO teaches a course that is organized and developed from scratch, to give the students the most pedagogical and didactic approach to programming and how to further teach programming to children. The advantage of developing new content specific to the profession is to approach students in a way that is directly relative to their work, and which gives them the possibility to develop their own ideas in programming. The disadvantages are that the course is so specific, and the content is so precise that it may be challenging to standardize such skills or to develop them in further education.

HVL teaches a course that is generally an IT programming, with some elements of automation. The advantages of teaching students a more general IT programming is that it fits with everything, with other courses, further development, it is commonly described, and such skills are fundamental in all work with IT and CS. The disadvantages are that the students would not learn the need of customizing programming their profession.

What should schools and higher education institutions do? Should programming be taught with a general IT approach, or should it be customized to the profession and field of knowledge? What takes more time and resources, or what gives better learning outcomes, to teach already develop content with few examples of customization, or to develop new content just for one course, that is not replicable or transmittable to others. The answer may be not to customize the entire field of programming, but to create a fundamental base of computational thinking, that does not differ from field to field, and is not based on any particular programming language, and then customize the "pure" programming accordingly to the profession the students have.

Conclusion

This paper was trying to answer the research question of if or how can an introduction to programming-course for non-computer scientists teach higher levels of observed learning outcomes in programming. By examining the content of two courses for non-computer scientists, this study has found that the IT-programming that is developed and observed in these courses is mainly on the Uni-structural and Multi-structural level of SOLO-taxonomy. Mainly because the professional goal usually deviates from the purposes and goals of IT and CS programming. To reach higher levels of observed learning outcomes like Relational or Extended Abstract, the criteria could not be just about the technical aspects of programming. The results show that the skills in programming differ from students from different professions.

References

Biggs, J. (2012). What the student does: Teaching for enhanced learning. *Higher education research & development*, *31*(1), 39-55.

Biggs, J. B., & Collis, K. F. (1982). *Evaluation the quality of learning: the SOLO taxonomy (structure of the observed learning outcome)*. Academic Press.

Bryman, A. (2016). *Social research methods*. Oxford university press.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). In Kampylis, P., & Punie, Y.(Eds.), *Developing computational thinking in compulsory education–Implications for policy and practice*. Luxembourg: Publications Office of the European Union.

Bocconi, S., Chioccariello, A. and Earp, J. (2018). *The Nordic approach to introducing Computational Thinking and programming in compulsory education.* Report prepared for the *Nordic@BETT2018 Steering Group*. https://doi.org/10.17471/54007

Grandell, L., Peltomäki, M., Back, R. J., & Salakoski, T. (2006). Why complicate things? Introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (pp. 71-80).

Griffin, P., & Care, E. (2015). The ATC21S method. In *Assessment and teaching of 21st Century Skills* (pp. 3-33). Springer, Dordrecht.

Hawkins, W., & Hedberg, J. G. (1986). Evaluating LOGO: Use of the SOLO taxonomy. *Australasian Journal of Educational Technology*, *2*(2).

LK20 (2020). *Læreplanverket i Kunnskapsløftet.* Utdanningsdirektoratet. Retrieved from https://www.udir.no/laring-og-trivsel/lareplanverket/

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, *41*, 51-61.

Mannila, L., & Nordén, L. (2017). *Att undervisa i programmering i skolan : Varför, vad och hur?* (Upplaga 1. ed.). Lund: Studentlitteratur.

Mathew, R., Malik, S. I., & Tawafak, R. M. (2019). Teaching Problem Solving Skills using an Educational Game in a Computer Programming Course. *Informatics in Education*, *18*(2), 359-373.

Selby, C. C. (2014). *How can the teaching of programming be used to enhance computational thinking skills?* (Doctoral dissertation, University of Southampton).

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

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725.

World Economic Forum. (2016). The future of jobs: Employment, skills and workforce strategy for the fourth industrial revolution. In *Global challenge insight report. Geneva: World Economic Forum*. Retrieved from https://reports.weforum.org/future-of-jobs-2016/chapter-1-the-future-of-jobs-and-skills/#view/fn-1