Strengths and weaknesses of a visual programming language in a learning context with children

MARIA HJORTH
Strengths and weaknesses of a visual programming language in a learning context with children

ABSTRACT

In a world where computers are a part of most people’s everyday life, learning how to instruct one to perform time consuming and/or complex tasks is beneficial. Visual programming languages aim to make the experience for people programming computers the best it can be by bypassing some of the issues of syntax and translation from mental plan to executable program. However, text-based languages come out on top when it comes to the programming languages most used. This paper aims at finding the strengths and weaknesses of teaching a visual programming language to novices in order to contribute to the otherwise lacking empirical evidence within the field of teaching computer programming. The methods used in order to collect data and answer the research question took inspiration from methods used in ethnomethodology. These methods were: observation through participation within a group of programming novices and semi-structured interviews with programming tutors. What can be seen from the study is that visual programming languages offer a quick introduction to the world of programming that in many ways plays down the difficulties within the area by making programming playful and creative. On the other hand, the boundaries of the language are quickly reached and require the students to switch to a text-based language. Also, the visual programming language did not help the students learn how to plan and troubleshoot their programs. When progressing to a text-based programming language where planning and troubleshooting are required steps of the process this may become troublesome.

Styrkor och svagheter hos ett visuellt programmeringsspråk i en inlärningssituation med barn

SAMMANFATTNING

1. INTRODUCTION ........................................................................................................... 1
  1.1. Research question ................................................................................................. 2
  1.2. Purpose .................................................................................................................. 2
  1.3. Scratch .................................................................................................................. 2

2. THEORY & LITERATURE STUDY ........................................................................... 2
  2.1. Definition of a Visual programming language .................................................... 2
  2.2. Education and programming ................................................................................ 2
    2.2.1. Skill acquisition ............................................................................................... 3
    2.2.2. Learning styles ............................................................................................... 3
    2.2.3. Learning programming .................................................................................. 3
    2.2.4. Gender and emotions ..................................................................................... 4
    2.2.5. Teaching styles .............................................................................................. 4
    2.2.6. Teaching programming .................................................................................. 4

3. METHODS .................................................................................................................. 5
  3.1. Inspiration from ethnomethodology ...................................................................... 5
  3.2. Participant observations ...................................................................................... 5
  3.3. Interviews ............................................................................................................. 5
  3.4. Data analysis method ........................................................................................... 5
  3.5. Method discussion ............................................................................................... 5

4. RESULTS ................................................................................................................... 6
  4.1. Research Participants ......................................................................................... 6
  4.2. First contact ......................................................................................................... 6
  4.3. Design first, function later .................................................................................. 6
  4.4. Boundaries of the language ................................................................................ 7
  4.5. Common problems & Questions ......................................................................... 7
    4.5.1. Interface .......................................................................................................... 7
    4.5.2. Troubleshooting ............................................................................................. 7
    4.5.3. Need of other types of knowledge ................................................................. 8
  4.6. Learning curve ..................................................................................................... 8
  4.7. Scratch accepting redundancy & the Trial/Error Method .................................... 8

5. DISCUSSION ............................................................................................................. 9
  5.1. From novice to advanced beginner ...................................................................... 9
  5.2. The importance of feedback when problem-solving .......................................... 9
  5.3. Positive reinforcement & learning styles ............................................................. 10
  5.4. Gender differences ............................................................................................ 10
  5.5. From a tutor perspective ..................................................................................... 10
  5.6. Future research ................................................................................................... 11

6. CONCLUSIONS ....................................................................................................... 11

7. REFERENCES .......................................................................................................... 11
Visual programming languages in a learning context with children

**Name:** Maria Hjorth  
**E-post:** mariahjo@kth.se  
KTH Royal Institute of Technology  
School of Computer Science and Communication  
**Supervisor at CSC:** Rebekah Cupitt  
**Commissioned by:** NOX Academy AB  
**Submitted:** 15th June 2017

**ABSTRACT**

In a world where computers are a part of most people's everyday life, learning how to instruct one to perform time consuming and/or complex tasks is beneficial. Visual programming languages aim to make the experience for people programming computers the best it can be by bypassing some of the issues of syntax and translation from mental plan to executable program. However, text-based languages come out on top when it comes to the programming languages most used. This paper aims at finding the strengths and weaknesses of teaching a visual programming language to novices in order to contribute to the otherwise lacking empirical evidence within the field of teaching computer programming. The methods used in order to collect data and answer the research question took inspiration from methods used in ethnomethodology. These methods were: observation through participation within a group of programming novices and semi-structured interviews with programming tutors. What can be seen from the study is that visual programming languages offer a quick introduction to the world of programming that in many ways plays down the difficulties within the area by making programming playful and creative. On the other hand, the boundaries of the language are quickly reached and require the students to switch to a text-based language. Also, the visual programming language did not help the students learn how to plan and troubleshoot their programs. When progressing to a text-based programming language where planning and troubleshooting are required steps of the process this may become troublesome.

**Keywords**  
Learning, Teaching, Visual programming languages, Scratch

**1. INTRODUCTION**

Humans have since ancient times utilized technology in order to expand cognition and gain knowledge [1]. Before spoken languages arose, humans expressed themselves using symbols and drawings in caves, a form of visual communication and cognitive technology expanding knowledge [1] [2]. New forms of communication are something that humans continuously engage in and communication with computers is something that has become increasingly important in our society. Computers have taken a natural place in everyone’s everyday life to aid us as a problem-solving tool. Future social sustainability may rely on developing a society where more people are able to communicate with computers using programming languages and that these become as natural to humans as the written word [1].

A constant striving for better communication with computers over the years has led to new computer programming languages and in 2016 the languages C, Java and Python were the most used according to Cass [3]. These formal, constructed languages, in contrast to natural, spoken languages, such as English, have been consciously created to solve different problems and to instruct computers to calculate and perform tasks for us [2].

When it comes to programming languages, learning and teaching them is notoriously hard and may be caused by the differences between natural and formal languages. Formal languages and natural languages both have features such as syntax and semantics but formal languages have none of the ambiguity, redundancy or allegorical expressions that permeate natural languages. Growing up speaking a natural language may therefore not help you when learning a formal language since there is a hard adjustment needed. [4]

Programming in general can be described as the process of using a formal language to transform a mental plan into one compatible with a computer. The closer the programming language is to the mental plan of the programmer the easier the process of creating a program will be [5]. The goal for researchers within the area of programming and programming languages has therefore been to make this transformation from mental plan to program as easy as possible [2].

An example of a text-based programming language (TBPL) where its fundamental vision is to make programming simpler and easier to read is one of the most used ones, Python. A part of their documentation that they
call the “Zen of Python” reads as follows:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts […] [6]

Even though the goal of Python was to make the language more accessible, it still requires the user to know the exact written syntax of the specific language. This requirement is one of the things that Visual programming languages (VPLs) have been trying to change by switching focus, from syntax to semantics, problem solving and application of logic [7] [8] [9]. Other goals sought with VPLs are: making programming more accessible to specific audiences and improve the speed of which users perform programming tasks. [9] VPLs have therefore become common in many different domains such as end-user programming, modelling, rapid prototyping, and design activities for architects, artists, children, engineers, and scientists. However, there is a lack of software engineering support mechanisms for VPLs to assist those working with these languages in contrast to TBLs. [10] [2]

According to Whitley, all research that nevertheless has been done regarding VPLs is based on the idea that:

[…] visual notations can be properly integrated with text to form VPLs and visualization systems that will improve the human programming experience. [11]

As can be seen, the area of programming is an important one where more research can help the programming languages evolve and make them easier to use, teach and learn. This research may in turn help evolve our societies.

1.1. Research question

The research question that will be answered in this paper is:

In learning contexts with young novice programmers, what are the strengths and weaknesses of visual programming languages?

1.2. Purpose

The purpose of this research is to get an understanding of the strengths and weaknesses of learning with a VPL as an adolescent novice programmer. When our societies in the future may need more people being able to code, understanding of how programming is learned and should be taught is of importance. The research will focus on the strengths and weaknesses that VPLs inherently have when it comes to teaching children programming by looking at one VPL named Scratch and how it is taught.

1.3. Scratch

Scratch was primarily created to learn by making animated stories and games [12]. It is a VPL with its foundation in the object-oriented programming paradigm. Object-oriented programming is based on the notion of “objects” that respond to messages via the use of its own “methods” [13]. Java and C++ are both examples of TBLs within the object-oriented paradigm [14].

To understand the results presented in this paper, some understanding of the interface of Scratch is needed (see Figure 1).

Scratch is a single-window programming interface that consists of a “script area” where tokens, called “code blocks”, are dropped and snapped onto other code blocks in order to create an expression that can be read by the computer and displayed on the “scene”. Multiple expressions can exist in the script area at the same time. code blocks that are not snapped onto other blocks will not be read and executed by the computer. Each “sprite” can be seen as its own entity, object, with its own script area. Different code blocks do different things and blocks that do similar things are grouped together in “categories”. An example of a category is “movement” where all code blocks that in some way makes a sprite move on the scene are found.

Figure 1- The Scratch interface

2. THEORY & LITERATURE STUDY

To answer the main research question, it is necessary to look at previous theories that address how humans learn and more specifically, how we learn to program and the ways in which this informs how programming languages are taught and learned.

2.1. Definition of a Visual programming language

To define a visual programming language, it is first important to define what visual programming is. According to Burnett [9] visual programming is:

[…] programming in which more than one dimension is used to convey semantics. [9]

These additional dimensions can be created by multidimensional objects, using for example spatial relationships or pictorial representations where each one of these objects or relationships is a token comparable to the word in a TBL. Put together these tokens create visual expressions that if incorporated in a programming language as semantically significant syntax, it defines the language as a visual programming language. [2] [9]

2.2. Education and programming

Programming skills is one key area of development in children’s education and learning in schools [13] [16]. It is therefore natural to apply existing learning strategies and methods to the teaching of programming for children.
### 2.2.1. Skill acquisition

Dreyfus and Dreyfus [17] describe that humans go through five stages when acquiring a new skill. All people begin their journey as novices and can eventually reach the level of expert usually with a lot of practice. [17] The five stages of skill acquisition are:

1. Novice: A novice learns objective facts, features and rules for determining actions based upon the facts and rules they have learned. A novice does not adapt what it does to the context.
2. Advanced Beginner: Advanced beginners has begun recognizing and handle situations that are not covered by given facts, features and rules. Advanced beginners are context sensitive but does not quite understand what they are doing.
3. Competence: The level of competence is reached when the skill acquirer considers the entire situation and consciously chooses an organized plan for achieving the goal.
4. Proficiency: Proficiency is reached when the skill acquirer no longer has to consciously reason through all the steps to determine a plan in order to achieve a goal.

Winslow uses this definition in his work to determine whether an undergraduate computer science student can reach the level of expert in four years of studies. He reaches the conclusion that students reach the level of competence within computer science in four years of studies rather than expert. He also mentions that one can be a novice in some areas and an expert in others. [18]

### 2.2.2. Learning styles

Acquiring skill through education is affected by a number of factors, one being personality. People learn in different ways and often favor one learning method over another. These learning methods are commonly called a student’s learning style. Felder and Silverman [19] describe some of the major learning styles amongst students. These styles describe how a student learns looking from the different dimensions of learning such as perception, input, organization, processing and understanding. A student often favors one style in a dimension over another and an example may be a student that favor the global style when understanding a subject but want the input to be visual and the process to be active. Forcing this student to learn using the opposite learning styles in the dimensions may lead to them becoming frustrated and failing tests. Figure 2 shows the different learning styles and to which area of learning they belong. It also shows the teaching styles, not to be confused with teaching methods, that can describe what a teacher focuses on within the different areas when teaching. [19]

<table>
<thead>
<tr>
<th>Preferred Learning Style</th>
<th>Corresponding Teaching Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensory intuitive</td>
<td>perception</td>
</tr>
<tr>
<td>visual auditory</td>
<td>input visual</td>
</tr>
<tr>
<td>inductive</td>
<td>organization</td>
</tr>
<tr>
<td>active reflective</td>
<td>processing active</td>
</tr>
<tr>
<td>sequential global</td>
<td>understanding sequential</td>
</tr>
</tbody>
</table>

The different learning styles will now be briefly described in the order they were mentioned by Felder and Silverman.

First are the “sensing learners” who like working with facts and real objects and like to see a connection to “the real world” with what they are learning. In contrast, the “intuitive learners”, favor working with abstract theoretical material and are usually quicker workers than the sensing learners. [20]

The “visual learners” remember best what they see. Charts, diagrams and symbols are input methods that work for a visual learner. “Auditory learners” on the other hand remember well what they hear and best what they hear and then say. “Kinesthetic learners” remember best what they taste, touch and smell. [19]

“Inductive learners” and “deductive learners” are almost opposites where inductive learners take particulars, like observations, and proceed to generalization while deductive learners take principles and general concepts and proceed to consequences and applications. [19]

“Active learners” are “doers” in the sense that they rather learn by experimenting, while “reflective learners” rather observe processes to learn. [19]

“Sequential learners” want new input in a, if possible, chronological and/or logical order, one thing at a time. “Global learners” on the other hand usually have a hard time understanding sequential teaching methods until they have gotten enough input so that they can understand it “all at once”. [19]

In an interview with Richard Felder made by Cardellini [20] Felder explained that most people are primarily visual, active and sensory learners. This causes issues when learning in a school system that primarily offers lectures that rather suits auditory and deductive learners who are intuitive. [20] Naturally, knowing about and being able to cater to different learning styles is also important when learning programming.

### 2.2.3. Learning programming

Computer programming in general is something that is both hard to learn and teach according to studies done on undergraduate students, where programming is a part of their education. The issues are mostly caused because there are few studies researching the pedagogy of programming languages and because a deeper understanding of how students learn in general is needed. As a result of this, when efforts to learn programming fail the fault may not lie in the technology but may be caused by of the lack of theoretically informed practice. [21]

In Traynor and Gibson, the issues with the lack of correct
pedagogy for teaching programming are expressed as follows:

Traditional teaching methods do not adapt well to the domains of coding and problem solving, as it is a skill best learned through experience. [21]

Some of the issues that TBL programming novices encounter, according to Miliszewska and Tan [22], arise not because they lack prior computing experience but because they lack problem-solving skills. This is caused by the fact that most schools do not offer logic/problem-solving modules.

Winslow tries to explain this problem-solving process that the student must learn in four steps:

1. Understand the problem
2. Determine how to solve the problem:
   2.1. In some form
   2.2. In computer-compatible form (note that novices have trouble going from a to b)
3. Translate the solution into a computer language program
4. Test and debug the program. [18]

But it is not only problem-solving that a student must learn in order to learn programming. Students must also learn how to design a program, construct it, compile and run it as well as achieve a basic understanding of the hardware and software components of the computer. Since all this knowledge is necessary for making a program work it can cause much of the frustration that students feel when learning. [19]

Other difficulties that students face when learning how to program are comprehension and visualization of abstract terms, such as data types and pointers as well as writing correct syntax. Even though the student with the help of pseudocode (informal code that contains the structure of a program but with non-specific syntax) can describe the problem and how to solve it, they may have a hard time visualizing how the solution will be handled by the computer and especially how it should be written syntactically in order to be executed properly. [22]

2.2.4. Gender and emotions

How people learn can differ not only because of personality and different learning styles but also because of their gender, how they feel when they enter a learning situation and how the learning situation make them feel. Kinzie and Joseph describe the differences between genders, in middle school ages, regarding what they want in game education tools. All children appreciated explorative interactions as well as problem-based interactions in games. In order to appeal to girls with educational games and encourage them in areas such as mathematics and science they found that one should incorporate creative interactions where the girls can, for example, modify character appearances, elements in the game or create artwork. Boys on the other hand are more intrigued by active and strategic interactions where limits and consequences are set in the game by for example time or character death. If offered appealing opportunities to develop related skills and self-esteem both genders may find the opposite genders favored interactions just as stimulating. [23]

In a study made by Um et al. [24] the importance of feeling positive emotions during the learning period were researched. In the study, the conclusion was drawn that positive emotions promote knowledge construction and problem solving. The study also showed that positive emotions lead to increased satisfaction for the same material.

2.2.5. Teaching styles

As can be seen in Figure 2 there are corresponding teaching styles to each learning style. When it comes to the dimension of organization, induction and deduction are different learning preferences but also describe different ways of teaching. Teaching by using induction includes methods such as problem-based learning, discovery learning and inquiry learning. The more traditional methods used, for example a typical lecture setup, are however based on deduction where teaching starts at fundamentals and then proceeds to application. In this traditional lecture setup, that is most often used in engineering education, the auditory, abstract (intuitive), deductive, passive and sequential learners are favored. [19]

Different teaching styles work better for different areas and different people which is especially important to remember when teaching programming that inherently is hard to teach by the traditional methods since it is best learned through experience. [21]

2.2.6. Teaching programming

When teaching programming to novices, a teacher is always exposed to the risk of the students putting their focus on the syntactic issues rather than the computational semantic power of the programming language. This focus on syntax prevents the novice students from understanding the role of the programming language as a tool for problem solving. [25]

Winslow [18] describe the pedagogy that should be applied in order to increase the novices’ understanding of programming:

1. Learn the syntax and semantics of one language feature at a time
2. Learn to combine this language feature with known design skills to develop programs to solve problems (this expands the students design skills and includes patterns and procedural skills such as planning, testing and reformulating)
3. Develop general problem solving skills. [18]

Miliszewska and Tan describe some pedagogical tricks a teacher can utilize that can help programming students absorb the information better. The first trick is an analogy trick to help students understand programming concepts such as input/output and sorting. By utilizing analogy and “real world” examples to explain these concepts they become easier for the students to grasp. The second trick mentioned is that the code in some way has to be relevant making the students see the purpose of what they are learning. The third trick is to constantly reinforce what the
students have learned by repetition. [22]

Much of the research regarding learning and teaching programming has been done on adults. According to Piaget’s theory of cognitive development adolescents age 11 and adults belong to the same stage of cognition which in learning means that they are regarding cognition, equal. [26] On the other hand, there are differences between children and adults when learning and one of the more important examples is that adults have more experience and previous knowledge to rely on when learning something new [27]. Sadly, little is known on how children learn how to program and this omission will hopefully be somewhat answered with this research. It is important to consider all these different styles of learning, the level of skill that is acquired, how the learning and teaching environment might affect cognition when studying how children learn to program using visual programming languages.

3. METHODS

3.1. Inspiration from ethnomethodology

When researching a question based on people’s experiences, societal concepts or phenomena, qualitative methods are often used. When conducting research with a qualitative method, open-ended questions in interviews, observation and audiovisual data are usually collected and analyzed [28]. Since the purpose of this study was to find the strengths and weaknesses of VPLs when taught to novice programmers a qualitative method was chosen that were inspired by methods used within ethnomethodology. Ethnomethodology is a branch of sociology that is often used within human computer interaction that in a paper by Dourish and Button [29] is described as:

Ethnomethodology is a particular analytic orientation to the practical issue of the problem of social order. [29]

In HCI ethnomethodology is for example used in order to get an understanding of work from the inside by fieldwork investigation [30][31]. This is done in order to get insights in the methods and practices used in activities and interactions so that these insights can be used in the design of technology to support work especially. [29] Therefore it seems fitting to take inspiration from methods used within ethnomethodological studies of HCI [30][31], which is why observation by participation and semi-structured interviews were chosen.

3.2. Participant observations

The main qualitative method used in the study is an observational research method where data was collected through observation and limited participation [32]. The reason for choosing observation as the main method was to keep conversations, reflections and reactions as natural as possible. The method was also chosen to eliminate some of the issues associated with interviewing children, such as parent permits, truthful answers and reflection depth [32]. Since observation as a method does not require the children to do something out of the ordinary or act in any particular way, hopefully the data therefore is a close to the truth as possible.

It was crucial that the documentations of the observations were as detailed as possible. When possible, data were documented during the observations, but most of the data were annotated shortly after the end of an observation day. In a study with children, ethics are central and during the observation period their well-being was always prioritized over the results. The results were documented when the children were not present in order to remove the feeling of being supervised, judged or graded in any way.

The observations took place during a five-day programming camp held by NOX Academy in Stockholm and the children were observed from 9am until 3pm. During these days they were taught the VPL Scratch during the first two days of the week, where the first day consisted of guided lectures and the second day was more open for them to try out the language on their own. The two following days, the novices were taught HTML and CSS. The last day of that week, they had the opportunity to choose whether they wanted to work with Scratch or with HTML/CSS. The novices were in total observed for 6 hours a day for 5 days. Of these five days, two to three days consisted solely of Scratch programming.

3.3. Interviews

To make sure that the observations were correct, semi-structured interviews with five VPL tutors were conducted. Semi-structured interviews are usually preceded by observations or informal unstructured interviews; in this case they were based on the observations. The semi-structured interviews were held in Swedish, recorded and transcribed. The quotations used in the paper were translated into English as true to their original meaning as possible. The interviews took place the week after the observation week and they were about an hour each. The interviews were held at the NOX Academy office in Stockholm.

One of the benefits of using semi-structured interviews is that questions and concepts can be prepared beforehand so that some of the data from different interviews can be compared. At the same time, semi-structured interviews provide the researcher with the option of modifying the interview questions during the interview so that everything interesting within the area is covered. [33]

3.4. Data analysis method

In order to analyze and structure the data gathered by the participant observations and the interviews with the tutors, an approach where patterns are derived from data was chosen. This analytic approach, also called thematic analysis, is a common method of data analysis within qualitative research. The method is usually chosen in order provide an illuminating description of a phenomenon which is the goal of this paper. [34]

3.5. Method discussion

In retrospect, the research methods chosen, participation observation in combination with semi-structured interviews, worked well for the purpose of answering the research question. If anything could be changed it would be the length of the study and the number of participants. Furthermore, by interviewing people developing VPLs and also interviewing people teaching TBLs to novices, the answer to the research question could have been more triangulated and possibly the validity of the conclusions could have been increased. It would also have been
interesting to look at more VPLs and how they differ and what effect that has on the research question.

4. RESULTS

Firstly, there will be a description of the group and a general description of the observations made regarding what the group did, their reactions and differences within the group over time. After the general description, the major themes that crystallized themselves from both the observation week and the interviews will be described. These include: “Design first, function later”, “Boundaries of the language”, “Common problems and issues”, “Learning curve” and “Redundancy and trial and error method”.

4.1. Research Participants

The group that was observed consisted of 8 children between the ages 10-13 who had no previous experience of programming. Since the children had no previous programming experience they all fit within the definition of a novice made by Dreyfus and Dreyfus and will therefore henceforth be referred to as “young novice programmers” or “the novices”. These young novice programmers attended a one week long day camp in beginners programming offered by the company NOX Academy. Six of the novices were boys and two were girls.

4.2. First contact

The first contact the novices had with the VPL during observation was in the afternoon of the first day. Before that, they got to know each other by learning everyone’s names and by playing team building games. When they during these exercises got to introduce themselves and what they were interested in, most of the novices said that they liked to play computer/video games and that they were curious about programming. In the afternoon when they were ready to start programming, the tutor started out by showing basic examples, such as getting the default Sprite to move. As part of these examples, the tutor explained how to find the code blocks in the interface, what they meant and how to execute the code. The novices listened and copied what the tutor did. They were highly focused on what the tutor did, did not talk to each other or help each other out and the room was quiet except for the questions that arose. The method used by the tutors the first day can be described as a mix between the inductive and deductive method. This method was described by the tutors as teaching by “live-coding”.

The first day, the interaction among the novices while programming was almost non-existent and almost no deviations from the code that they had tried together with the tutors was done by any of the participants. On the other hand, the curiosity was high amongst the group and many questions like “what happens if we change the value to something higher” and “why can’t I put the code block here” were raised. The first day, many focused a lot on the design of their games and chose different backgrounds for their Scenes, colors, Sprites, sounds and costumes for their Sprites. Many created a story of what their Sprite did which they could express verbally when asked but not express in code.

At the end of the day when recapitulating everything that had been learned, the novices were asked what was the most challenging one child exclaimed “Everything!”.

Although, when the child was asked what the code blocks did, the child had no issues explaining exactly how they functioned.

All tutors interviewed confirmed that starting with examples was the way they usually organized they lectures every time they were to teach Scratch and that with Scratch it is easier to just show the children examples and explain at the same time. They also mentioned that they started out with examples in order to quickly get the novices to see that it does not have to be that hard and to spark interest:

I think I usually start with examples and maybe a short explanation of what programming is […]. Children have a short attention span; things have to happen constantly. To enter a long theoretical discussion is not that appreciated. If you show an example it gets more exciting because things are happening. Otherwise they often become unfocused and bored.

The tutors also described the advantage of using examples in order to make the programming and the language as undaunting as possible:

Many can become frightened when they see the program [Scratch] for the first time and think “What am I supposed to do now? How?”. It is easier if someone says “Do it like this” and then explain why so that they get the [programming] theory as well and then get a challenge so they have to start thinking about what they have learned and think for themselves. Then they can get full freedom and you [as a tutor] can help them and inspire them to new ideas.

Already by day two, the group showed a greater amount of self-confidence than day one and most of them had become advanced beginners. Some of the novices had tried to code by themselves at home after the first day and had watched videos of how others had built Scratch games. Most of the novices, when asked, were eager to get started, had already started coding, wanted to get help with their project or discuss their code even before the day had officially started. The group had a hard time leaving their projects behind during recess regardless if they had a problem, were stuck or wanted to try something new. Many of the novices had their own ideas inspired by games they played and they wanted to make the games more complex by adding more features such as levels, scoring systems and penalties. The novices started trying out code blocks that had not been shown or discussed the previous day. They began applying knowledge from the day before on new areas of programming.

4.3. Design first, function later

In Section 4.2 the tendencies of the novices starting out by focusing more on design and then more on function when they were becoming advanced beginners further into the observation period was mentioned. This tendency was confirmed in the interviews with the tutors, where all but one described the same pattern. The majority also described that they thought this was because in the beginning when the programming is hard they focus more
on what they feel like they know from before for example using a simple computer graphics program much like Microsoft Paint to design their own Sprite.

 Especially with novice children they quickly realize that they can draw their own sprites, they recognize that [process] and does it. They sort of make it their own.

When they slowly get more comfortable with the programming language, they focus more on what they want their Sprite to do than on how it looks and they use code more to also change the appearance of the Sprite.

4.4. Boundaries of the language
From day two until the end of the observation period, the novices began finding inherent limitations in Scratch where the executions of their ideas became very complex or impossible to create with the language. This was something that the tutors also had seen and they described it like this:

You cannot do exactly everything that you want with Scratch, there are very solid boundaries that you act within. Within these boundaries it works really well and is a fantastic program. But as soon as you want to step outside of these boundaries and do other things it does not work.

Two of the kids encountered a bug in Scratch the first day that the tutors had a hard time working around. The code block called “Touching” had a bug where it could not determine whether it was touching something or not. During the observation, another issue arose regarding nested if statements that none of the tutors could solve without deleting the code and going about it in another way. The tutors expressed that they thought that it should work because it would have in a TBL but somehow Scratch could not handle that type of complexity.

In one of the interviews, a tutor expressed the thought that it is within boundaries we become most creative whereas with full freedom we have a hard time creating:

There are many limitations [in Scratch], but it is a good thing. It is fun to be creative within some boundaries because that is when creativity becomes most interesting. [...] If you can do whatever you want it can be a disadvantage and be an obstacle when you try it [programming] out for the first time and try to learn.

The boundaries of the language may be both a positive and a negative when learning programming with a VPL. But the boundaries are not the only thing that comes with both opportunities and problems.

4.5. Common problems & Questions
Problems that occur when learning a VPL such as Scratch can vary depending on teaching strategies and the students. In this case, the young novice programmers and their teachers encountered some difficulties.

4.5.1. Interface
Initially, most of the questions and problems the children had were about the interface and how one interacts with the code. They had no problems understanding the concept of dragging and dropping the code from the code block area to the script area where the code is composed. They also had no issues understanding where the result of the execution is shown.

One example of an interface problem the children did encounter the first day was that they didn’t understand how to and why a new code block needed to be added to already existing code in order for the computer to read it correctly. The children had not understood that the code is constructed like a puzzle where circles fit in the circular holes and where the tab of one piece fits into the groove of another. This issue was also described by the tutors:

Something that I have especially noticed are questions like ‘why can’t I put the code here?’, why code need to be placed as it does to function. It is a crucial part [of programming] and it is so natural [for oneself] that one knows that it has to be put there. It is so easy [as a tutor] to do little changes in the structure of the code and read the code. I would say reading the code is one of the hardest things [for the children].

At the end of the first day, working with the code blocks correctly was no longer an issue for the majority of the group since they with some practice understood the concept of working with it as a puzzle.

However, they still had some issues understanding the categories. They understood that similar code blocks could be found in a Category but they still clicked their way through each one to find which category they were looking for.

There are a lot [of questions like] ‘Where is that?’, ‘Where do I find that?’ since there are six different categories to choose code from. Usually they click on categories until they find it but the question still arises. It [Scratch] is very pedagogic otherwise, the circular code blocks fit in the circular holes, but not all get that straight away.

With time most of the novices issues with the interface did subside as they got used to the programming environment. One of the biggest problems the novices encountered was, however, troubleshooting. This was a problem that did not resolve itself during the observation period.

4.5.2. Troubleshooting
How the computer reads the code, where to find the code block and which code block does what was a bit confusing to them. Because they found it hard to read their own code the way the computer reads it they found it hard to troubleshoot on their own. During the interviews with the tutors, they described this as a common issue where the novices, when trying to troubleshoot on their own, usually solved the problem by deleting the last thing they did and rewrote it, deleting the entire code and starting from the beginning or by accepting the issue.

It is not the same thing with Scratch [as with TBLs], that one learn it from the language itself, it does not really point you to why it does not work. Troubleshooting does somewhat require that one has previous experience from it.

Day two the novices had a better understanding of when
things happened in the code and where to look when encountering an issue. However, reading the code from start to finish, like a computer, still caused problems for most as their increased knowledge also made their programs more complex, which in turn made reading them more complex.

From the first day until the last day the novices had a lot of redundancy in their code. Mostly because they had issues troubleshooting and reading their code. Therefore, they did not remove any of the redundant code in fear of breaking anything. More about redundancy is found in Section 4.7.

One of the tutors explained the advantage of how the VPL kept the novices interest up by displaying another type of error than standard error messages:

\[ \text{It [Scratch] is very advantageous in the way that the errors they encounter rather is “It does not do what I want it to do” rather than a lot or red error messages and nothing that is happening. It is important to keep their interest up.} \]

Since the novices had issues with troubleshooting, this was something they had to learn from the tutor together with the language itself. But how to troubleshoot was not the only thing they had to have knowledge about in order to understand the programs they wrote.

### 4.5.3. Need of other types of knowledge

Throughout the observation period the novices had a hard time understanding the coordinate system that the scene is divided into, which is caused by the fact that they had not the previous mathematical knowledge needed. This caused them to have issues when using code blocks that determined exact position on the scene by using \( x \) and \( y \) coordinates and also to understand that for example negative values on the \( y \)-axis made the sprite move down whereas negative values on the \( x \)-axis made the sprite move left. A majority of the novices solved this issue by trial and error. If the sprite moved the wrong way, they switched positives to negatives or vice versa.

Initially a majority of the group also had a hard time understanding that each sprite has its own script area where you are required to write the code that you want for that specific sprite. They had to learn how to think in an object-oriented way. This quickly changed since by the end of the first day, after trying out Scratch more they had understood how to work with different sprites and their script areas by trying it out.

Later on the issues the novices faced were caused by the code becoming more complex. Mostly, they had issues understanding how nested conditional statements worked which is logical thinking in its purest form together with knowing when which condition is taken into account. They had to understand the basis of logical thinking in order to understand how conditions worked.

However, all they had to know and learn still did not seem to slow them down to much since they solved it by using the TE-method.

### 4.6. Learning curve

By talking to the novices the second day about their ideas and helping them by drawing flow-charts, most could code by themselves. It also became clear, by discussing and asking some of the novices who were able to create more complex projects, that they had a deeper understanding of the concepts of programming such as if statements, for and while loops and operators. Many of the novices needed to be challenged and got ideas for how to make their projects more interesting and complex but without getting a lecture on how to do it. The method used by the tutors day two was leaning more towards induction so that, according to one of the tutors, the novices had to figure out for themselves how they could apply what they had learned from the day before to their projects and understand the concepts by knowing how they worked in a real-life context.

By day two some of the novices could, without clear instructions, complete or almost complete a project solely by knowing how the end-result should work and dared to use code blocks that had not been introduced to them by the tutor or only briefly introduced for example messages/function calls and variables. These novices often used methods of getting to the finished result that differed from what the tutor expected and differed from the methods the tutor showed others who needed help with similar ideas.

The days when the novices focused on HTML and CSS instead of Scratch most turned to Scratch when they got stuck with HTML, if they were bored and during recess.

In the interviews with the tutors a question regarding how long they approximated it took for the novices to go from the level of novices to advanced beginners was asked. All tutors came to the conclusion that they felt that it didn’t take log at all and the answers differed depending on how the tutor defined advanced beginner and depending on the individual. In general, the answers ranged between a few hours to a couple days.

\[ \text{After about 1,5 days they have most of the tools needed and it is very easy for them to, in many cases, explore for themselves and find things in the menu [of categories].} \]

Two of the tutors described the advantage of Scratch when it comes to the transformation of ideas from mental plans to code as well as it enabling creativity:

\[ \text{It is much easier to transfer ideas and easier to test if it is possible or not. One does not have to finish everything and then try out the idea but one can rather test it along the way.} \]

The learning curve seemed to be influenced by the fact that Scratch accept redundancy and that the novices could utilize the trial and error method when transforming their mental plans into code.

### 4.7. Scratch accepting redundancy & the Trial/Error Method

During the observation period the code the children wrote was full of redundancies and in some cases blocks were not
connected to anything but were simply located in the script area. In the interviews, the tutors described that they had seen this as well and thought that it was caused by the fact that Scratch allows the children to code by the trial and error method (TE-method) where the code can be executed although it is not fully composed:

One of the biggest advantages is something I have touched upon earlier [in the interview]. It is the fact that you can feel your way towards the solution without it [Scratch] crashing and you can drag and drop blocks [into the script area] that you are not even using, which is also a disadvantage. It therefore looks like a lot of code but it is only a small part of it that is running.

The TE-method was possible for the children to utilize due to the fact that Scratch could compile and parse the loose code blocks and the redundant code without crashing. Together with the immediate feedback the children got from the system and the fact that they could try out their code without it being complete they seemingly felt their way forward to the result they wanted.

However, the fact that there was a lot of redundant code caused a bit of an issue when the children tried to troubleshoot. Because the code they had written was cluttered with redundant parts and they had issues reading the code like a computer they had a hard time finding and filtering these parts out. The tutors also described this redundancy in the interviews, both as a positive in the sense that there could be a lot of redundancy without the program crashing and it therefore helped the students explore and try out the code, but also that it caused issues because it made their code cluttered. One tutor explained the positives about Scratch accepting redundancy in a case of a young novice girl he had tutored:

She had made it overly complicated, but she had not asked for help. With simply pure determination she had just sat down and made it work. Much of the code was redundant but there was always something that made it work [the way she wanted it to]. It was pure chaos but it worked.

What can be seen from all of the results is that there are good and bad sides to everything that Scratch encourages and enables. Important to reflect upon is whether the positives outweigh the negatives.

5. DISCUSSION

Learning and teaching programming is not an easy task especially since there are more things that have to be understood than merely the programming language. VPLs seemingly come with many strengths but issues are also present.

5.1. From novice to advanced beginner

The first and most prominent topic that can be discussed based on the results is the time it took until the children to go from novice VPL users to advanced beginners. In Section 2.2.1. Dreyfus and Dreyfus’ five different steps of skill acquisition were described and from these definitions it can be seen that the children became advanced beginners during the observation period. What they could not do by themselves was troubleshoot and beforehand decide on the architecture of the code which is why they did not reach the competence level.

The reason the children never reached the level of competence may be caused by the VPL itself. Scratch, by enabling the TE-method, does not force the users to plan their programs beforehand since non-finished code and ideas continuously can be tested. By not learning how to plan and properly troubleshoot using Scratch the children will have to learn how to properly do that when possibly learning a TBL at a later point.

5.2. The importance of feedback when problem-solving

One can argue that the biggest advantage of a VPL when teaching and learning may be that it inherently helps students with their problem solving by not causing syntax errors. VPLs switches the focus from syntax to semantics, as well as enabling users to try their solution without it being complete. Since Scratch allows for the TE-method it enabled and continues to enable the children to learn in a truly, by Felder and Silverman described learning style, inductive way, where creativity without a high skill level is supported.

In Section 2.2.3. Winslow’s problem-solving process shows that in order to fully have learned how to problem-solve a student has to be able to understand the problem, come up with a solution in form of a program and be able to troubleshoot and debug it. The final step in the problem-solving process was not fulfilled by the children during the observation period. The reason they did not learn how to troubleshoot may be because of the lack of feedback they got from the program and the fact that the feedback they did get did not point them in the direction of where it went wrong. Since all possible combinations of tokens into expressions are parsable by the computer it will never give an error message which might have helped them to learn where problems arise and where to look. The user is therefore expected to read the program from start to finish to find the area that does not operate as intended which is a part of programming that takes a while to learn.

From the results, the TE-method also seemed to be the main reason why the children wrote a lot of redundant code. This redundant code caused problems for the children when they were trying to troubleshoot by themselves. Since Scratch could accept a lot of redundancy and loose code blocks in the script area, without error messages or other feedback, there was more code that the children had to read through. Since this was something they did not feel comfortable doing on their own it posed an issue for the tutors who had to try to teach the student how one should read and parse code.

The possibilities that the enabling of the TE-method provides therefore also caused issues that the children had to handle by learning the rules of how the computer reads the code, which is not as intuitive.
5.3. Positive reinforcement & learning styles
It is easy to see that Scratch is built for novice programmers who favor the active rather than reflective way of learning, as described by Felder and Silverman, as well as favor visual input rather than verbal.

When talking about problem solving and troubleshooting in Scratch, the fact that errors do not show up as red error messages but rather showing up as a different result than expected is both a weakness as mentioned above but also a strength. As mentioned in Section 2.2.4, Um et al. found that positive emotions while learning promote knowledge construction and problem-solving, therefore it is positive that errors are not as big of a letdown for the users. This enable learning by induction as well as cater to the active learning style described by Felder and Silverman, see Section 2.2.2, since the learners might not get as scared of making mistakes when trying out their ideas. The fact that the novices showed interest in programming before they started working with it, as described in 4.2, may also have helped them learn faster since they entered the learning environment with positive feelings.

The TE-method, the immediate feedback and the fact that errors do not feel as failures, together with the colorful interface and customizable Sprites and backgrounds may be why Scratch inspires creativity. From the results, it can be seen that Scratch was not an obstacle for creativity amongst the children but rather became a tool for design and function that inspired and invited the novices to create something of their own, just like they would with clay or paint and canvas. This creativity and enabling of exploring may be why the interest did not seem to diminish among the children during the week and why they, when they felt other things were boring or hard, returned to Scratch.

5.4. Gender differences
In Section 2.2.4, Kinzie and Josephs study [23] was mentioned where the differences between what children in middle-school wanted for an educational game was described. According to their study all children appreciated explorative and problem-based interactions which Scratch enables with for example its feedback system. In the study, girls seemed to respond positively to creative elements in educational games and customizable Sprites by letting its users choose, design and draw their own sprites and backgrounds as well as utilize code in order to make changes to the appearance of the sprites may help more girls and creative children embrace VPLs and programming. Since the design interaction does not require the users to now programming it is easy to start out with. On the other hand, the active and strategic interaction, enjoyed mostly by boys, is something the users have to create themselves by adding for example a scoring system or a timer to their games. Even though adding these elements by using code is not hard it requires the user to understand Scratch and it may not be as easy to begin with as with design. By giving the novices the option of starting out in their comfort zone and from there begin learning programming the novices may feel more empowered and may not see programming as threatening.

5.5. From a tutor perspective
As described by Imel [27] adults have a lot of the previous knowledge that they can apply when learning something new. Children do not have this advantage and therefore tutors have to be prepared to not only teach programming but also logic, math and explain the logic behind different programming paradigms. But they do not only have to know what the children know from before they also have to keep in mind the different personalities and preferences that may exist within the group.

In Section 2.2.3, Traynor and Gibson’s [21] theory of traditional teaching methods falling short when teaching programming was described and as shown in Section 4.2, the tutors seemed to actively try to minimize the theoretical discussions and rather use live coding via examples as the main teaching method. The tutors wanted the novices to quickly get to the point where they could code by themselves and learn by doing. This shows that Scratch enables the tutors to teach by using Felder and Silverman’s inductive methods mentioned in Section 2.2.5 which according to Traynor and Gibson is advantageous when learning programming since it is non-traditional.

Another advantage of using Scratch as a tutor is the fact the the VPL with its immediate feedback in the form of continuous execution of code, provides the tutors with an easy way of showing and describing abstract concepts. The tutors can describe the programming concepts by showing the students why it is important for their projects and also by showing what it does in a real-life example. Scratch therefore naturally provides the tutors with Miliszsewska’s and Tan’s two first teaching tricks described in Section 2.2.6. The third trick described by Miliszsewska and Tan, regarding repetition, is something that Scratch itself does not provide and is therefore something that the tutors have to provide in their teaching. Scratch also bypasses the syntax issues seen amongst TBL students [25] (see Section 2.2.6), which help the tutors focus more on teaching and helping the students with semantics, programming logic and problem-solving.

Scratch caters better to some learning styles than others. The active learners mentioned in Section 2.2.2 who favor learning-by-doing have an advantage when working with Scratch since the best way forward is to try. Tutors also have to remember that Scratch is inherently visual and therefore does not support auditory learners who may need lectures to fully understand the concepts. Even though the active and visual learners are in a majority amongst most groups of people according to the interview with Felder [20], it may be advantageous that tutors let the students work in pairs where the reflective students can watch and learn while the active learner is trying the VPL out.

Another thing that the tutors have to keep in mind are the boundaries of the language. Since it did not take more than a couple days for the children to work independently with Scratch they quickly reached these boundaries. Many of the children had fantastic ideas that became very complex or impossible to realize with Scratch. They also created code that, according to the tutors, logically should have worked but because of a bug or a boundary in Scratch did not. One could argue that Scratch primarily was built to be
an introduction to programming and therefore should not support more complex programs. One can also argue that a vast amount of possibilities could be intimidating and therefore restrict the creativity that can flourish when boundaries are set. But since the children reached this boundary fast, it would be interesting and a help for the tutors if the children could see the underlying TBL and from there expand their programs by writing their own text-based code and break free of Scratch in their own time.

5.6. Future research
Since the field of programming languages and programming pedagogy is lacking there is much to be done for future researchers. It would be of interest to conduct similar observations and interviews to the ones in this paper but for students studying a TBL as their first programming language and to compare the two datasets with a quantitative method as well as qualitative in order to find out, for example, how long it took for the participants to understand and solve the same problem and how enjoyable they felt it was.

6. CONCLUSIONS
The goal of this paper was to find the strengths and weaknesses of a VPL in a learning context with young novice programmers. Even though much can be said about this, one of the most protruding strengths of the VPL is that it reduces the learning curve and turn novice users of the language into advanced beginners quickly. Another strength of the VPL it that it gives the user positive feedback by accepting redundancies and reducing potential crashes which may invoke positive emotions that in turn may increase cognition. The VPL also opens up for creativity which can especially help girls to start programming. Some learning styles are best addressed with Scratch such as the active, inductive and the visual learning style, but with the right organization in classes and a mix between the inductive and the deductive teaching style all learners can benefit from the use of Scratch as an introductory programming language.

One of the biggest weaknesses of the VPL lies in the fact that the boundaries of the language are quickly reached and that they are complicated or impossible to circumvent. Therefore, the tutors were restricted by the language when questions and ideas that overstepped these boundaries arose. The students were also not spared from learning some rules of programming because even though much of the interaction with the interface was user-friendly and easily understandable they still had to understand how they should place the code and how it should be placed. They also had to understand some of the underlying logic of programming such as how the computer parsed the code in order to work with Scratch and especially troubleshoot. They also had to understand other kinds of knowledge in order to work with the program such as math. The fact that the children could go ahead and create a program without a plan as well as create a lot of redundant code without the program crashing is also a weakness when learning programming since more advanced languages require a plan, structured code and no “loose ends” in the code. The transition from Scratch to a TBL may therefore not be easy in that sense.

As a tutor for children learning to program, VPLs help play down the area of programming so that more children can feel that this is something that is not frightening but rather fun and creative. The weaknesses of the VPL found in this study show that it is not easy to cater to everyone and everything and that some compromises must be made, but with the correct pedagogy these weaknesses can be compensated for. VPLs may be one step towards a society where programming is as common as writing.

7. REFERENCES

[12] J. e. a. MALONEY, ”The scratch programming language and environment,” ACM Transactions on Computing Education (TOCE), vol. 10, nr 4, p. 16,
2010.


[34] J. SMITH and J. FIRTH, "Qualitative data analysis: the framework approach.,” *Nurse researcher*, vol. 18, nr 2, pp. 52-62., 2011.