Students' Visualisations of Programming Concepts: An Exploratory Analysis

Rolando Gonzalez[1], Trond Klevgaard[1], and Kjetil Raaen[1]

Kristiania University College, Oslo, Norway

Abstract. This study explores the intuitive visualisations of code created by novice programming students with the aim of uncovering how these reflect their understanding of fundamental programming concepts. Through a thematic analysis of student drawings, this exploratory research identifies common themes and categorises different modes of visual expression. The findings reveal great variations in how students can visualise code, and suggest that drawings can serve as an effective tool for both diagnosing student misconceptions and supporting their learning of programming. This study highlights the potential of incorporating visual exercises into programming education to improve concept comprehension.

Keywords: Programming visualisation · Drawing · Introductory programming · Exploratory research

1 Introduction

When teaching introductory programming, educators may wonder how well students grasp the fundamental concepts being introduced. What does a variable, loop, or array represent in the mind of a novice programmer? Furthermore, how do students develop a coherent understanding of complex code processes that integrate multiple techniques?

Programming is a challenging subject for many students. Worldwide, the failure rate is around 30% [5]. Educators use different tools and methods to make programming more concrete and to help students understand. Examples of tools include Scratch for children and young adults and BlueJ in higher education. Examples and analogies are commonly used in the classroom or lecture theatre to contextualise and explain code.

The study explores how novice programming students understand code through a thematic analysis of their intuitive visualisations. The research objective of this study is to explore the understanding of code by new students through their intuitive visualisation of code. Further, we aim to gather insight that can help educators understand how students think, such as what misconceptions the students may have and a broader set of alternatives to explain code, both orally and through visualisation.

2 Background

Although some literature reports on drawing as a learning method, it has been challenging to find sources that examine this topic within programming. However, here also there is a tradition of using visualisation. The following therefore accounts for how students may have encountered programming-related visualisation methods in addition to providing a review of academic studies on drawing as a learning tool in this field.

2.1 The background of the students

In recent years, as computer technology conquers more of society, programming knowledge has become increasingly important to all citizens. This is not only because programming is a useful skill, but because it is considered an effective way to train logic, reasoning, and problem-solving abilities. Consequently, programming is becoming increasingly common in primary and secondary education curricula[15]. We expect this to lead to better basic programming skills from our first-term students in the coming years, and we cannot assume that our first-term students have not learnt some programming before.

Some of this teaching also uses visualisation techniques, further complicating the pictures of the participants' prior knowledge of programming visualisation. One of the most popular tools for teaching young learners programming is the Visual Programming Language (VPL) Scratch [13]. It is text-based, but the text is embedded in a graphical format, blocks, making it easier to understand. A similar VPL called App Inventor by MIT gave good results in learning basic programming concepts among students at a university in Taiwan. Some of the advantages found were not having to worry about syntax or code errors that can be frustrating, and the motivation of the students is also positively affected [16].

Professional programmers use standardised formal and semi-formal languages to express ideas visually throughout the design process. The most important family of such diagrams is the *Unified Modelling Language, UML*[10]. Participants in this study were not introduced to UML or other modelling languages as part of the course. However, given their ubiquity, participants may have come in contact with them previously.

Educators sometimes use metaphors and analogies to create a bridge between abstract and real-life objects and situations to make it easier to grasp. For example, a variable may be talked about as a box or container in which we can put the values inside; although this may cause false beliefs such as that since a box in real life may contain multiple things, students may conclude that variables contain multiple values [7].

2.2 Using student visualisation for learning programming

Drawing has traditionally been considered a core competency in the visual arts. It has also been extensively used in primary education as a learning and communication tool for children [4]. Recent years have seen increased interest in academic investigation of pedagogical uses of drawing in a wide range of university courses [2], including the sciences [1]. Whilst rare, some authors have also focused specifically on how drawing can be used to improve or illuminate the process of learning programming. One study [11] focused on emotions and thoughts about programming, rather than the task itself. The researchers asked the students to produce a drawing answering the question *What does programming mean to you?*. They used a relatively straightforward analysis that looked at the following attributes: actors, activities, aspirations, and affect. This analysis is partly quantitative; for example, they mention that positive emotions are present in 38% of the drawings, negative emotions in 28%, and mixed emotions in 15%. They also see a marked move from positive to negative emotions from first- to second-year students. The authors suggest using drawing to help educators understand the emotional state of the students and maybe adjust teaching to keep the challenge appropriate.

Although this approach is informative, it is not a direct part of teaching. Another study [8] attempts a more direct approach: Task students with visualising algorithms without limiting style or content. Visualisations of algorithms often focus on the movement and flow of data, rather than the actions themselves, so these visualisations are not directly visualising the code. Participants were asked to give peer feedback to other students' visualisations. Later, the learning for all participants was tested. The goal here is improved learning, and the form and content of the visualisations are given less importance. They found that, in general, these exercises improved learning. Although this was not the primary focus, the authors commented that the students produced more diverse visualisations than those in standard learning materials.

3 Method

In this study, we used a qualitative analysis approach to analyse drawings collected from novice programmers.

3.1 Data collection

Data collection was conducted in 2020 at a Norwegian university college in a first-year introductory programming course. The course included students from five different Bachelor's in IT specialisations and Bachelor's in Digital Marketing. The programming language used was JavaScript. Midway through the 12-week semester, students received a two-week assignment. They were asked to draw their interpretation of three descriptions of functionality, completed as one of five types of tasks in groups of 2-5 students. The students were informed that the drawings could be used as part of a research project, ensuring anonymity, and that the assignment would be accepted without the drawing. The descriptions to be drawn included techniques like loops, arrays, conditional statements, functions, click events, and output to HTML. The students were instructed to draw with pen, pencil and paper, photograph the drawings with their phones, and write a short text describing them. The wording of the instruction was (translated from Norwegian):

In this task, you should create your personal interpretation of the code through drawing. There is no right solution

For this research, we chose not to focus on the short text descriptions; however, we look through them to see if they were of significance. We found that the great majority of the students' text descriptions were very short and did not add more understanding to the drawings. We received between 400 and 450 submissions. For the analysis of this paper, 30 of the drawings were selected. We experienced saturation; that is, few new codes appeared after this number.

3.2 Data analysis

Our qualitative study is based on the process and analysis of thematic analysis presented by Braun and Clarke [6]. Furthermore, we have chosen the approach Braun and Clarke refer to as inductive, or bottom up, as this was highly explorative. The process consists of six main steps, beginning with researchers getting to know the data through notes, reflections, and discussion, then creating codes and themes at different levels of refinement, and finally producing a report. Although Braun and Clarke mainly refer to speech or text, their process can also be applied to visual materials, as Rose's highly similar content analysis method shows[14].

Thematic analysis of this research was performed mainly by two of the three researchers working in parallel. Throughout the process, the two researchers would meet to discuss and write down their thoughts and reflections in a separate document that is especially used as a preparation for the next discussion part of the paper. Analysing the drawings took considerable time, comparable to analysis of longer in-depth interviews. The third researcher was included at critical points in the process to provide general quality control and general feedback. When the two researchers came to their second version of refined codes, they started working more closely to group codes and identify themes, before the findings chapter was written, laying the foundation for the process of writing the discussion chapter.

3.3 Limitations

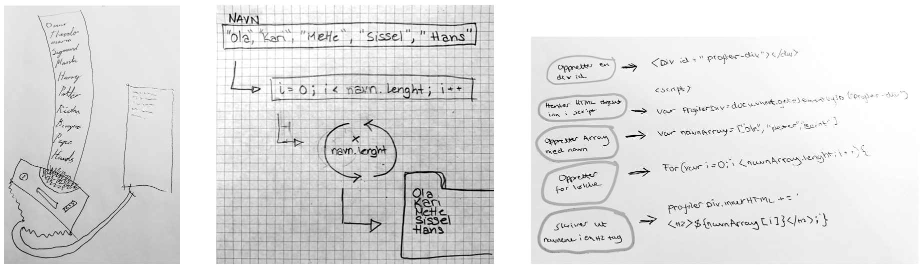
We reviewed 30 drawings, so there may be details that were not captured. We do not have data on how many students did not submit drawings, and the students had varying levels of drawing experience, which may have affected their output. Some students' drawings could have been influenced by external influences, such as UML, or by their peers during group work. The task was assigned midway through the semester; giving the task at other points would likely lead to different results. Furthermore, this research is exploratory and we did not use a predefined theoretical framework to interpret the drawings. Lastly, we do not know whether the drawings submitted were initial attempts or revised versions, nor do we have information on the time spent on the task.

4 Findings

This chapter presents the findings of our research study. It is divided into several sections, each detailing one theme. We identified the following themes:

1. Explanatory modes
2. Structure
3. Level of detail
4. Ways of drawing the for-loop
5. Flow, time and ordering
6. Transfer of information
7. Active versus passive elements
8. Figurative drawings and elements
9. Relationship between process and results

4.1 Explanatory modes


NAVN
"Ola", "Kari", "Mette", "Sissel", "Hans"
i = 0; i < navn.lenght; i++
navn.lenght
Ola
Kari
Mette
Sissel
Hans
Oppretter en div id
Henter HTML objekt inn i script
Oppretter Array med navn
Oppretter for løkke
Skriver ut navnene i en H2 tag

```
<Div id = "profiler-div"></div>
<script>
Var ProfilerDiv=document.getelementbyID ("Profiler-div")
Var navnArray= ["Ole", "peter", "Bernt"]
For (var i = 0; i <navnArray.lenght; i++){
profiler Div.innerHTML += '
<H2>${navnArray[i]}</H2>';}
```

(a) An image-based response.

(b) Text and image integrated in a diagram.

(c) A textual drawing, executed as bullet-points.

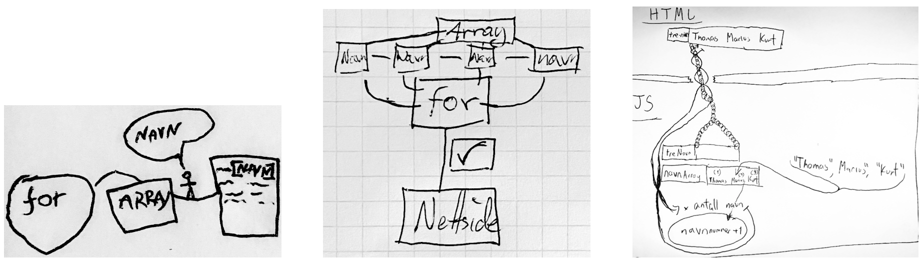
Fig. 1: Examples of the various explanatory models.

The students' responses can be categorised into different explanatory modes. These can be divided into two broad categories, text and image, which in turn can be broken down into subcategories. Text can be written as code, partial code, descriptive prose, short prompts, or keywords. Images range from abstract diagrams to metaphorical representations using figurative elements. Although some students submitted entirely image-based (Fig.1a) or purely textual (Fig.1c) responses, most combined the two. For example, a figurative drawing accompanied by descriptive text, or code can be integrated into a diagram (Fig.1b). Although diagrammatic drawings may contain elements recognisable as block diagrams, flow charts, or network diagrams, these tend not to be executed using UML or other formal drawing techniques. Similarly, it can be observed that although some students use graphic conventions from borrowed from cartoons like speech/thought bubbles or visual onomatopoeia (like jagged lines expressing energy, as in Fig.1a) their drawings are not conceptualised as comics *per se*.

4.2 Structure

Most drawings contain three main components representing the for-loop, the array, and the webpage. Some students break these into smaller units, resulting in a more detailed and/or fragmented representation. Others chose to omit a component, usually the for-loop or the array. Diagrammatic drawings commonly use rectangles to group and isolate content. These rectangles, or “containers”, can hold a variety of content, from pieces of code to simple labels like “for-loop”. Although containers are most commonly rectangular, other shapes can be used. Some drawings use a variety of shapes, possibly to indicate differences in the functions contained. In other drawings, shapes are joined together to create more complex forms capable of grouping different types of content whilst simultaneously maintaining their distinction.

4.3 Level of detail



(a) Drawing showing a low level of detail.

(b) A drawing showing a more detailed representation of the array.

(c) A drawing detailing the relationship between programming languages.

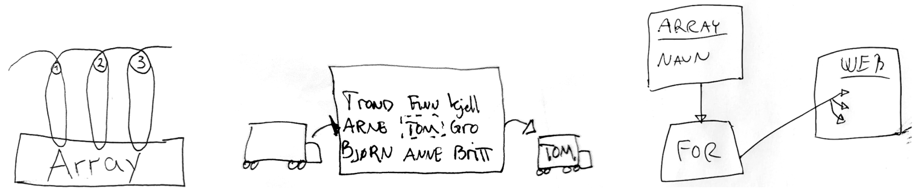
Fig. 2: Examples of the various levels of detail.

The drawings can be categorised by level of detail. Detail can refer to the extent to which the workings of the for-loop, array, or web page are visually explained. Most drawings, like Fig.2a, barely attempt such a visual explanation. Instead, they only show connections between a series of ’black boxes’, containers named for their function. In such drawings, the inner workings of the components are hidden. Detail may also refer to process. In low-detail drawings, arrows can be used as black boxes, representing not only direction or connection, but changes in content or the passing of time.

Higher levels of detail can be shown in different ways. The for-loop or array may be divided into smaller parts, as in Fig.2b. These parts can be combined, viewed as modules that build on each other, or connected by lines or arrows. Students might add code or descriptive text to serve as explanations or, in the case of diagrams, as content within containers. Text-based drawings may feature bullet point lists that explain the process step by step, as in Fig.1c. Instead of visualising how the process works in principle, some students chose to create examples, often by inventing a list of names to populate the array.

4.4 Ways of drawing the for-loop

The for-loop is one of the three main components in the drawings, and the students can be broadly put into one of the three main categories based on how they handle the loop. The first group, which comprises a bit less than half of the students, has chosen to draw the loop and loop process explicitly. The second group, a bit more than half of the students, has not drawn the loop figuratively, but rather used other ways to conceptualise it. The third group seems to have omitted the loop completely.



(a) The for-loop interpreted literally as a looping form.

(b) The for-loop drawn as scenario.

(c) The for-loop represented only as a 'black box' with arrow.

Fig. 3: Examples of ways to draw the for-loop.

Students who have figuratively included the for-loop draw elements symbolising repetition, movement, and direction. Examples include arrow lines forming a circle, spirals, interconnected loops, incomplete circles with start and stop points, lemniscate (infinity symbols), and wavy lines. Some represent the required number of operations by drawing multiple lines or arrows. Additionally, some drawings indicate incrementation with details like 'i++' or '+1', or note the number of repetitions.

In the case of students who indirectly include repetition and process, we see drawings of scenarios. One scenario is a drawing of an empty truck on the left side of a box representing a storage building, and the same truck on the right loaded with names (Fig. 3b). This drawing does not directly show the process, but we infer that the products are loaded inside the building. Another scenario shows a shopping list and store shelves with products. Here, we must imagine the buyer picking products off the shelves one by one.

The third group of students has not shown any direct indication of repetition (Fig. 3c). Some students have included loop code alongside other code snippets or drawings. Others go directly from the array to the web page using an arrow.

In another example, the drawings show information bits in a row, with arrows, commas, or lines pointing to the web page, resembling an information caravan.

4.5 Flow, time and ordering

The drawings show different orderings of the three main components: for-loop, array, and output. By ordering, we mean that the process is initiated by one component and continues to the next. With most drawings going from left to right or top to bottom, equal numbers of students draw the for-loop first, then the array, and vice versa, with the output at the end. Among students who omit the output, it is more common to draw the array before the for-loop. A minority start with something other than a for-loop or array, such as a button that initiates the process. In some cases, the drawing has no identifiable order.

Most students draw the process as linear and stepwise, with a (more or less) clear start and end. However, students also created nonlinear depictions, some resembling network diagrams (Fig.2b) and others containing free-floating elements without connections. A loop element may, for instance, simply begin in thin air and disappear into nothingness. In very few cases, one has what appears as a circular process (Fig.2c), and in one drawing the flow direction is ambiguous (Fig.1a). In the case of a drawing using comic strip elements, with parts of the code represented by stick figures, the logic could be said to follow the logic of the cartoon, the conversation between characters. Another dimension connected to flow is time. A drawing such as in Figs.1a and 2c in a way depicts a moment in time where the process is happening in front of our eyes; we are in the middle of it happening. However, other drawings, such as Figs.1b and 2b, show the start and end; you choose the time of the process when you look at them.

4.6 Transfer of information

The transfer of information between components occurs in a variety of sequences, travelling between the loop and the array, between the array and the output representation, or between the loop and the web page.

Connections enabling information transfer is illustrated in various ways. Lines are commonly used to connect components (Fig.2b). Arrows can be used to indicate direction as well as action (Figs.1b and 3c). Students draw lines and arrows that stretch from component to component, stop before reaching a component, or extend into components to suggest different levels of connection.

Whether students draw each array value that is being handled or if they depict it as a repetitive black-box process varies. We have drawings showing a line representing the for-loop entering the array and exiting on the other side carrying information. We also have examples of a loop pattern entering and leaving the same array box (Fig.3a). A drawing of the array repeated as a series of horizontal units shows a line representing the for-loop entering each unit, causing names to fall down.

Some of the more esoteric and scenario-based drawings use more complex or very different means of expressing the transfer of information such as the jagged lines of energy shown in Fig.1a.

4.7 Active versus passive elements

The drawings typically show elements as either active or passive; they do something or have something done to them. Placing an element at the beginning of the process suggests that it is the active part, interacting with the others. Depending on whether the for-loop or the array is placed first, the drawings show the for-loop accessing the array (Fig.2a) or the array providing information to the loop (Fig.3c).

In rare instances, the drawing indicates that the website or HTML initiates the process. Usually, the website or output is simply the result of a prior process. Sometimes, no main active element is clear; there might just be an arrow from an array leading to an output without specifying an active element.

An element in the drawing can be both passive and active depending on the step in the drawing. For example, an array might first send information to a loop, making the array active and the loop passive. In the next step, the loop could become active in generating the output.

4.8 Figurative drawings and elements

A minority of drawings are figurative, rather than diagrammatic, using everyday metaphors to explain the functionality. Examples of figurative, metaphorical drawings include one showing personifications of the for-loop, array, and website having a conversation, a scenario in the grocery store where the shopping list is the for-loop and a shelf full of products is the array, a printer literally printing out the names on a piece of paper (Fig.2a), and trucks representing the for-loop driving through a warehouse representing the array, collecting names as if they were goods (Fig.3c).

Figurative elements can also be used as part of diagrams, typically at the start or end of the process. Sometimes, they book-end an otherwise abstract digital process. Showing output on real-world objects like a laptop, printer, folder, browser window, and document or paper is common (Figs. .2a. and .3b). Real-world objects can also serve as a starting point for the process. For example, a figurative button can indicate that the functionality requires input to be initiated.

Figurative elements in the form of ideograms serve as visual shorthand, representing a certain content without directly showing it. For example, straight or wavy lines can be used to signify text. Even near-abstract elements, like lines connecting components, can add narrative to the diagram when interpreted figuratively. For example, such lines may double as wires or roads. One student drew a physical wall symbolising a barrier between programming languages which the for-loop was required to forcefully break through (Fig.2c).

Some drawings include human figures. These can form part of a metaphoric scenario in which personifications of the for-loop, array, and web page are having a conversation or in which the transfer of information occurs by a person walking down a path which also doubles as a line connecting two parts of the functionality (Fig.2a). Several drawings contain smiling faces, as part of the human form, as freestanding symbols, or as integrated into drawings of technological equipment, such as printers (Fig.1a).

4.9 Relationship between process and results

In several drawings, we observe a divide between the 'internal' workings of the code and its 'external' output. This separation manifests itself in various ways. One way is a tendency to show the output displayed on elements related to the material world, contrasting with abstract depiction of internal processes. Examples of such elements include laptops, screens, folders, or sheets of paper. Real-world associations may be reinforced through contrasting drawing styles, where the material object is drawn in a three rather than two dimensions. Another way in which drawings show the divide between internal and external is by separation. In such drawings, a space can be seen between the parts showing the code and the parts showing the output. This space can be further accentuated with elements like arrows, indicating a "jump" between the process and the website, or a ticked checkbox showing a successful data transfer (Fig.2b). In some drawings, the output is shown falling down, as if affected by gravity, unlike the internal processes unaffected by such real-life factors.

5 Discussion

In this research, the students illustrate their understanding of a small program by drawing. This study focusses on visualising programming concepts on paper. Due to the exploratory nature of this work, direct comparisons in the literature are scarce. Therefore, the findings are discussed on their own merits, where we have no direct comparison.

5.1 Drawings as a communication tool

Drawing is a communication tool that can be actively used to build knowledge [3]. One of the main challenges in learning programming is that its abstract and dynamic nature makes it difficult to grasp [9]. By drawing, the students make the invisible visible in a double sense. They are forced to conceptualise how they think a program works. At the same time, they create visual artefacts that can be interpreted by others.

Reflecting on the students' drawings, we see commonalities. First, they choose and draw components they perceive as involved. Second, they illustrate the connections and order of the processes. The drawings also detail information processing, differentiate between active and passive components, and show the process flow. Emotions are sometimes communicated through elements such as smiling faces. Students use different styles and categories for drawing. An interesting question arises whether the drawing category influences thinking or vice versa. Flow charts, for example, encourage stepwise thinking.

Reading the drawings is an act of interpretation that necessarily relies on the reader's personal background and professional training. As Andrade's research [3] shows, meanings can be read in a drawing which were not intended by the person who created it. However, the potential for misunderstanding can be reduced if the drawing is accompanied by an oral explanation.

Not all students included the three main parts indicated by the task. This can be seen, for instance, in drawings lacking a representation of the loop. One reason behind such omissions may be that the student in question has felt a particular technique, or interaction between techniques, to be outside their understanding. We also note that an almost equal number of students draw the array before the loop as the opposite. Such drawings give the appearance of an "active" array delivering information to the loop. From the educator's perspective, one may ask if this shows a misunderstanding of the process. Does the student believe that the array is doing the work rather than the loop? Some students show concretely with lines and arrows how the loop "injects" values into the web page, while others completely abstract the transfer of information. The concreteness of may lead us to conclude on a clearer mental model of what the for-loop is doing.

Based on the findings in this work, we see several possibilities in different contexts as to how the use of drawings, including intuitive drawings, can potentially improve teaching and learning. A first output of this exploration is the discovery of many different manners of drawing programming, which one may also understand as discovering that there are several ways in which students understand programming. An educator may have their approach to teaching based on their personal understanding and preferences, and we hope that this research will reveal that we may be more different when it comes to our mental models regarding programming than we first expected. A lecturer may visualise concepts through drawings while explaining, not only in one category of drawing, but multiple categories on different levels of detail to reach students that have different backgrounds and are on different levels of understanding. In addition, a drawing may be a tool in a supervision situation between a student and a student assistant or teacher, or in discussions between peers in coconstructing understanding and knowledge.

Creating a drawing is an active and iterative process that can improve understanding and correct misconceptions. Unlike abstract discussions, drawings provide concrete entities, reducing cognitive[3] load and simplifying discussions by enabling easy reference to parts. Quillin and Thomas [12] identify different pedagogical motivations for using drawings in education. The motivations may, for example, be fostering active learning, foster memorisation, make it easier for a teacher to evaluate the student's understanding, and better be able to give feedback. In supervision, asking students to draw can start discussions, letting the student set the premise for further knowledge construction - you as a supervisor let the student reveal their mental model and understanding. This is particularly useful when vocabulary is limited, as drawings can intuitively represent issues and concepts. Drawing, discussing, and using gestures can be combined to describe, analyse, and add movement and direction when needed. In addition, the concreteness of the drawing can become something that the student can use to connect and build their academic language on.

5.2 Code, output and the material world

As described in the findings, we have observed that several drawings emphasise a separation between the "internal" workings of the code and its "external" output. By internal we here refer to those processes that are hidden from view in daily life, and by external we mean the visible or otherwise perceptible results of these processes. Moreover, we have noted a tendency to show the output (and in some cases input) drawn figuratively, as an object from the material world, whilst the code has been rendered in an abstract manner.

When asked to draw the internal workings of the code, students are implicitly asked to visualise the invisible. Unless they choose to reproduce the code as text, mirroring the way programmers usually experience code, this necessarily involves a degree of invention. As shown in the findings, such inventions can be more or less fantastical. Some students chose to illustrate code using figurative drawings and imagined scenarios, whilst others conceptualised it as information flows represented diagrammatically using geometric forms such as rectangles (containers), lines, and arrows (connections). When it comes to the output created by code, the situation is different. The students have all seen the results of code displayed on their computer screens or on other devices.

On a basic level, the gap between an 'abstract inside' and 'figurative outside' can be interpreted as the confluence of two different types of drawing. The first concern itself with visualising the invisible, and the second with representing objects the students have abundant visual references and memories of. However, it may also point towards difficulties in conceptualizing how code and output are connected. Students may not previously have been required to think carefully about how the relationship between hardware and software plays out in practice.

Most of the students chose to represent the functionality in a diagrammatic way. However, as explained in the findings, some also created figurative drawings. These use metaphors taken from daily life to explain the workings of the code. As noted in the background chapter, metaphors can be used to describe something unfamiliar in familiar terms. Metaphorical explanations may therefore be interpreted, not only as a student's effort to represent something unfamiliar using references they already possess but also as a way of lessening the threatening aspects of the unknown. The use of human forms, smiling faces, and anthropomorphised machines may be seen as further evidence of a desire to imbue technological processes with relatable human qualities and positive emotions.

5.3 Choice of visual language

When describing the visual language used, it is important to keep in mind the limitations of the method. Although not measured, it is safe to assume that few students have formal training in art or visualisation and probably did not invest much time or effort. This may explain the schematic representations or simple figurative elements. Despite these limitations, there is variety in the data. Many diagrams are improvised using arrows, rectangles, or circles to represent processes. The term 'loop' suggests circles or spirals, logical ways to illustrate the function of a loop. Some drawings omit repetition, indicating potential misunderstandings that educators can address to aid comprehension.

Some diagrams show similarities to formal modelling languages such as UML state diagrams. This might be due to previous exposure to UML or similar languages, or it might indicate that the designers of UML and the students work from the same intuitive principles. Either way, these intuitive drawings are a good tool to start exploring more formalised methods for mapping programming patterns. However, it should be noted that the task here is too simple to break down into multiple units in most visual languages, so students would have to improvise even if they had a good understanding of UML.

In contrast to, but often combined with, the diagrams, some students use more figurative elements in the form of humans and physical objects. Some think in terms of a human doing a job, others represent the various elements as physical objects. If the array is drawn as a warehouse that stores numbers, that is a metaphor we could use in future conversations with this student. When humans are present, they are mostly stick figures but show clear expressions. Do these expressions reveal some feeling of the students? Or are they drawing happy faces to comfort themselves?

6 Conclusion

Our thematic analysis reveals significant potential for drawing as a teaching tool in introductory programming. Through a structured but open-ended analysis, we see that drawings provide insight into the thought process of the students. We conclude that drawings can be developed as a technique to facilitate the learning of fundamental programming concepts. For instance, students' drawings can help identify areas of misunderstanding, such as incorrect representations of loops or arrays. By choosing an appropriate visual language, educators can align their teaching methods with the diverse cognitive styles of students, whether they favour abstract concepts or need more concrete and practical examples. Used as a basis for discussion between peers, student assistants, or teachers, drawings can provide information on the knowledge construction process.

To be useful in the development of future teaching tools, this work must be followed by a more concrete design and evaluation. Can we design a teaching scenario where we instruct students to draw, and then use the drawings to give feedback, either from teachers, TAs, or peers. This scenario can also be quantitatively evaluated for improved learning. Similar experiments can be conducted to determine which exact concepts are best taught in this way.

References

1. Ainsworth, S., Prain, V., Tytler, R.: Drawing to learn in science. Science **333**(6046), 1096–1097 (Aug 2011). https://doi.org/10.1126/science.1204153
2. Ainsworth, S.E., Scheiter, K.: Learning by drawing visual representations: Potential, purposes, and practical implications. Current Directions in Psychological Science **30**(1) (2021). https://doi.org/10.1177/0963721420979582
3. de Andrade, V., Freire, S., Baptista, M., Shwartz, Y.: Drawing as a space for social-cognitive interaction. Education Sciences **12**, 45 (01 2022). https://doi.org/10.3390/educsci12010045
4. Anning, A.: Drawing out ideas: Graphicacy and young children. International Journal of Technology and Design Education **7**, 219–239 (1997)
5. Bennedsen, J., Caspersen, M.E.: Failure rates in introductory programming: 12 years later. ACM Inroads **10**(2), 30–36 (apr 2019). https://doi.org/10.1145/3324888
6. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qualitative Research in Psychology **3**, 77–101 (01 2006)
7. Hermans, F., Swidan, A., Aivaloglou, E., Smit, M.: Thinking out of the box: comparing metaphors for variables in programming education. In: WiPSCE '18. ACM (2018). https://doi.org/10.1145/3265757.3265765
8. Hübscher-Younger, T., Narayanan, N.H.: Dancing hamsters and marble statues: characterizing student visualizations of algorithms. p. 95–104. SoftVis '03, ACM (2003). https://doi.org/10.1145/774833.774847
9. Medeiros, R., Ramalho, G., Pontual Falcão, T.: A systematic literature review on teaching and learning introductory programming in higher education. IEEE Transactions on Education **PP**, 1–14 (08 2018). https://doi.org/10.1109/TE.2018.2864133
10. Medvidovic, N., Rosenblum, D.S., Redmiles, D.F., Robbins, J.E.: Modeling software architectures in the unified modeling language. ACM Trans. Softw. Eng. Methodol. **11**(1), 2–57 (jan 2002). https://doi.org/10.1145/504087.504088
11. Moskal, A.C.M., Gasson, J., Parsons, D.: The 'art' of programming: Exploring student conceptions of programming through the use of drawing methodology. p. 39–46. ICER '17, ACM (2017). https://doi.org/10.1145/3105726.3106170
12. Quillin, K., Thomas, S.: Drawing-to-learn: A framework for using drawings to promote model-based reasoning in biology. CBE—Life Sciences Education **14**(1), es2 (2015). https://doi.org/10.1187/cbe.14-08-0128, pMID: 25713094
13. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y.: Scratch: Programming for all. Commun. ACM **52**(11), 60–67 (nov 2009). https://doi.org/10.1145/1592761.1592779
14. Rose, G.: Visual methodologies. Sage Publications (2001)
15. Szabo, C., Sheard, J., Luxton-Reilly, A., Simon, Becker, B.A., Ott, L.: Fifteen years of introductory programming in schools: A global overview of k-12 initiatives. Koli Calling '19, ACM (2019). https://doi.org/10.1145/3364510.3364513
16. Tsai, C.Y.: Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. Computers in Human Behavior **95**, 224–232 (2019). https://doi.org/10.1016/j.chb.2018.11.038