Patterns for Introductory Programming Education

Anja Bertels*
anja.bertels@th-koeln.de
Cologne University of Applied Sciences (TH Köln)
Gummersbach, Germany

Leonie Kallabis*
leonie.kallabis@th-koeln.de
Cologne University of Applied Sciences (TH Köln)
Gummersbach, Germany

ABSTRACT

In computer science education, introducing programming concepts to beginners poses unique challenges. This paper explores three pedagogical patterns designed to enhance the learning experience for novice programmers: Abstract Guided Instruction Through Storytelling, Block-Based Programming, and Self-Contained Kit. By employing narratives, metaphors, interactive tools, and comprehensive kits, these patterns aim to make programming more accessible and engaging. The patterns provide a systematic approach to addressing educational challenges in programming instruction and are designed for educators as well as learners.

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**.

KEYWORDS

Programming, Coding, Computer Science Education, Coding Kits

ACM Reference Format:
Anja Bertels and Leonie Kallabis. 2024. Patterns for Introductory Programming Education. In *Proceedings of 31st Conference on Pattern Languages of Programs, People, and Practices (PLoP 2024).* ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

In the rapidly evolving field of computer science education, introducing programming concepts poses unique challenges. Traditional methods often fail to bridge the gap between abstract concepts and practical coding skills, especially for learners with no prior experience. This difficulty is compounded by the inherent complexity of programming languages and the diverse cognitive demands placed on learners. To address these challenges, innovative educational methodologies have emerged, aiming to make learning to code more accessible, engaging, and effective.

This document explores three educational patterns designed to facilitate the entry into programming for beginners. Each pattern addresses specific aspects of the learning process, providing strategies to overcome common barriers and enhance the overall learning experience.

*Both authors contributed equally to this research.

PLoP 2024, October 13–16, 2024, Skamania Lodge, Columbia River Gorge, Washington, USA.

Hillside ISBN 978-1-941652-20-6

These three patterns can be placed within the dimensions of abstraction and interaction (shown in figure 1). In Computer Science, programming concepts like advanced data structures or algorithms are often described using the term abstract. In this paper, the term *abstract concepts* is used to refer to that understanding. For example, the concept of a variable is an abstract concept for novices because it addresses a mostly unknown domain for them. **Abstraction**, on the other hand, is understood in this work as the distance of a concept from its actual implementation. For the variable as an abstract concept, this would mean describing it as a labelled box that can contain certain items is high abstraction, in other words, the abstraction from the abstract concept. **Interaction** is understood as the ability to manipulate something. Placing a story that a learner can only listen to at low interaction, while having a tangible object with freely replaceable and manipulable elements at high interaction.

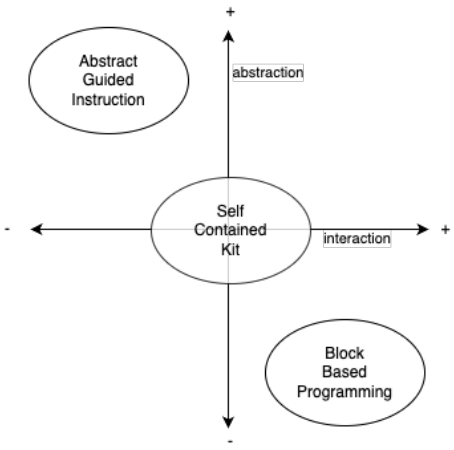

Figure 1: Relationships between the patterns regarding the amount of abstraction and interaction

The first pattern involves providing a high level of abstraction with minimal interaction by using metaphors within the framework of storytelling. This approach is particularly suitable for younger learners or those with no prior experience, as it simplifies abstract programming concepts and presents them in a relatable context.

The second pattern emphasizes high-interaction, low-abstraction learning solutions using Block-Based Programming. This method allows learners to engage with coding concepts through visual blocks, facilitating a hands-on, interactive experience. Block-Based Programming is especially effective for those who need to experiment with logic and problem-solving without getting bogged down by syntax.

The third pattern suggests using a comprehensive and self-contained set of learning materials that includes all necessary resources. This could be an all-in-one kit with hardware and software components, a board game, or an educational suitcase. The key aspect of the Self-Contained Kit is that all required elements are provided, enabling learners to apply their skills to real-world projects and effectively bridge the gap between theoretical knowledge and practical application.

Together, these patterns form a cohesive framework designed to support programmers in their educational journey, from the initial exposure to abstract concepts to the practical application of coding skills. There is no inherent gaps between each pattern, their effectiveness depends on the specific implementation. The patterns build on each other, creating a seamless progression that addresses specific challenges at each stage of learning.

Pattern writing and the use of specific formats have evolved significantly, with foundational contributions from Christopher Alexander [1]. Our pattern collection is structured to provide a clear, systematic approach to addressing educational challenges. Each pattern begins with the **Context**, detailing the educational setting and circumstances. The **Problem** section specifies the issue arising from the context. The **Forces** section highlights the complexities involved, considering the needs of students, educators, and the environment. The **Solution** offers a proposed resolution, including variations and a fictional example to illustrate practical application. The **Consequences** section outlines the benefits and liabilities of the solution, referencing previously mentioned forces. Lastly, **Known Uses** provide real-world examples of the pattern in action. This structured approach ensures clarity and applicability, with continuous fictional examples to aid understanding and relevance across different educational contexts.

In summary, these three patterns form an interconnected framework that guides learners from understanding abstract concepts through storytelling, to interactive problem-solving with Block-Based Programming, and finally to applying their skills with a comprehensive kit. This structured progression supports a continuous and engaging learning journey in computer science education.

2 ABSTRACT GUIDED INSTRUCTION THROUGH STORYTELLING

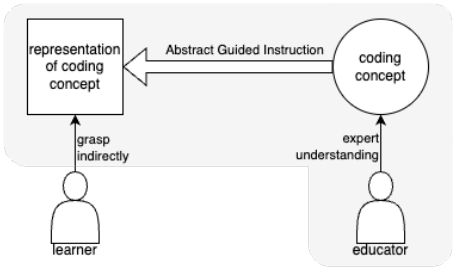

Figure 2: Usage of Abstract Guided Instruction to convey coding concepts through an abstracted representation.

2.1 Context

Learners are engaging with programming for the first time and have no prior experience. It is essential to introduce and explain various coding concepts to establish a foundation for programming.

2.2 Problem

Learning coding concepts without any prior experience poses a significant entry barrier, as it requires a completely new way of thinking [7]. The success in programming depends on several predictors such as algorithmic thinking, logical reasoning, and mathematical skills, none of which can be assumed as given.

2.3 Forces

- **Intangible Concepts**: Programming concepts are often abstract and difficult to understand, especially without practical examples and lack familiar references learners can relate to.
- **Lack of Familiar References**: Many concepts have little to no connection to already familiar topics, complicating comprehension.
- **Wrong Implications**: When learners use familiar topics to grasp concepts without guidance of an educator, they might use inappropriate metaphors that lead to wrong implications.
- **Computer Anxiety**: Learners often fear using computers, particularly for programming, and generally fear making mistakes, which leads to hesitant and insecure learning [8].
- **Cognitive Load**: The cognitive load is high when many diverse new concepts must be learned simultaneously [11].
- **Concept Integration**: It is challenging to recognize and understand the relationships between different programming concepts.
- **Sequential Concepts**: Many programming concepts are sequential and build upon each other, making understanding the basics crucial [6]. If foundational concepts are not fully grasped, it impedes the learning of more advanced concepts or the connections between them.
- **Motivation**: Without early successes, learners often lose motivation quickly [10].

2.4 Solution

Provide a high level of abstraction with minimal interaction by using metaphors within the framework of storytelling.

A suitable methodology to introduce programming to learners without prior experience involves using teaching methods that offer a high level of abstraction and require minimal direct interaction of learners. This approach allows learners to gradually familiarize themselves with new concepts without becoming overwhelmed. A central role of this approach is the use of Storytelling, which is defined as ”Augment[ing] the transfer of knowledge through a narrative method that uses rhetorical devices to create tension in order to present a fact-heavy topic in the form of a story” [3]. This method can help to link various concepts and present them in an understandable and relatable context. Appropriate metaphors should be used to make abstract programming concepts more tangible. Well-chosen metaphors help avoid misunderstandings and explain complex ideas in a way that is accessible and understandable for

beginners. Combining these approaches facilitates the entry into programming and lowers the learning barrier.

2.5 Consequences

Benefits.

- The use of storytelling and appropriate metaphors creates a **connection to familiar concepts**, improving understanding [2].
- Stories are **easier to remember**, helping learners to better retain concepts [2].
- A methodology with high abstraction and minimal direct interaction requires **detailed guidance**, leading to quicker successes for beginners.
- The **entry barrier** is very low due to the high level of abstraction and resultant accessibility, making it easier for learners to engage with new concepts.
- This methodology and extensive guidance minimize the room for errors, boosting learners' **confidence** and **reducing fear of failure**.
- Through storytelling, learners can identify with the narrative, which increases **learning motivation**.

Liabilities.

- High abstraction and extensive guidance create a very **restricted framework** with limited flexibility and exploration opportunities, which leads to a linear experience for learners that might be harmful to their natural curiosity.
- The **abstract context** can quickly become unsuitable as learners progress and the methodological framework no longer meets their needs.
- The **distance from code** due to high abstraction can make the transition from metaphors to text-based code with syntax challenging.
- With sequential concepts, a frequent **change of metaphors** may be necessary, complicating understanding and maintaining continuity in the learning process. This process also includes the challenge of creating fitting metaphors for educators.

2.6 Known Uses

Osmo Coding Awbie. "Osmo Coding Awbie" is an interactive game designed for children, using physical blocks that interact with an iPad to teach basic coding concepts. Players arrange these tangible blocks in front of the iPad to create sequences of commands that guide Awbie, a playful character, through various adventures in a colorful virtual world.

The storytelling revolves around Awbie's journey, filled with challenges and rewards, engaging children through a narrative that makes learning enjoyable. The adventure and treasure hunt theme serves as a metaphor for coding, illustrating concepts such as sequencing and logic in a playful context. The app is shown in figure 3

Figure 3: Coding Awbie App selection screen and level gameplay with use of tangible coding blocks

How to Code a Sandcastle. "How to Code a Sandcastle" is a children's book by Josh Funk that introduces young readers to programming concepts through a delightful story. The book follows Pearl and her robot friend Pascal as they try to build the perfect sandcastle on the beach.

Figure 4: Book cover and inside view

The book abstracts coding by breaking down the process of building a sandcastle into simple, step-by-step instructions that mirror coding logic. Through its engaging narrative, it explains complex ideas in an accessible way. The story of Pearl and Pascal captures the readers' imagination, making the learning process fun and relatable. The metaphor of constructing a sandcastle effectively conveys the principles of programming, such as planning, sequencing, and debugging, with Pascal the robot representing a computer following coded commands. The book cover and an inside view of the book is shown in figure 4.

Adventures in Coding. "Hello Ruby: Adventures in Coding" by Linda Liukas is a book that introduces children to the world of coding through the adventures of Ruby, a curious and imaginative girl. The book combines storytelling with interactive activities to teach foundational programming concepts.

Figure 5: Inside view of the book

"Hello Ruby" uses narrative and visual abstraction to convey coding principles. Ruby's adventures in a whimsical, illustrated

world make abstract programming concepts tangible and relatable for young readers. The storytelling is rich and imaginative, capturing children's interest and making the learning process enjoyable. Two examples of storytelling are shown in figure 5. Each adventure Ruby undertakes serves as a metaphor for various coding concepts. For instance, problem-solving quests and treasure hunts illustrate debugging and algorithmic thinking, allowing children to learn coding in a playful and engaging manner.

3 BLOCK-BASED PROGRAMMING

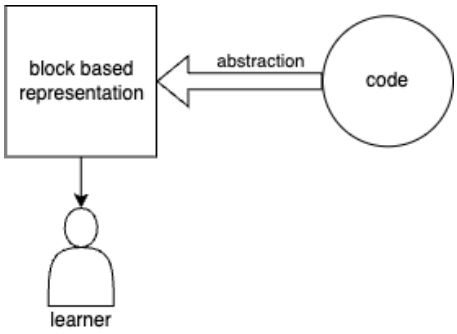

Figure 6: Usage of Block-Based Programming to convey code through an abstracted representation as blocks instead of text.

3.1 Context

Learners understand the basic concepts behind programming and algorithmic thinking at a high level of abstraction. They want to expand their knowledge and learn how to write syntactically correct code in order to create a working program.

3.2 Problem

There is a considerable discrepancy between understanding algorithmic processes and the ability to code from scratch. This discrepancy can be described as a difference in the levels of abstraction. Algorithmic thinking requires the learner to think about abstract concepts at a high level because its goal is to build a repeatable process. Writing text-based code with syntax requires the learner not only to apply algorithmic thinking (a new way of thinking for them), but also to translate it into syntax with which they may not be entirely familiar. These two points in the process of learning to code leave open a gap that has not been fully organized. There are no steps between these two points that help visualize the structure of the code or resemble syntax at a level of low abstraction. Writing code can be seen as a process that provides low- to medium-interaction, using common editors and syntax. This prevents learners from experimenting with the code and feeling free to change things and explore different solutions.

3.3 Forces

- **Cognitive Load**: Linking existing knowledge to new information while aquiring new knowledge can lead to a high cognitive load [11].
- **Barrier to entry**: The barrier to entry can be high for some learners because they have never worked with code or anything like it before.
- **Motivation**: Learners' motivation is influenced by the experience they have while learning. Providing an experience that does not meet learners' needs for competence, autonomy, and relatedness can lead to a decrease in motivation [9].
- **Situational interest**: When a learning task has personal relevance, novelty, activity level, and comprehensibility situational interest can be elicited, which is thought to precede and facilitate individual interest.
- **Prior Knowledge**: The learner's prior knowledge plays an important role in learning to code. Additionally, the degree to which that knowledge is embedded in the person's mental model can affect their learning experience.
- **Self-perception of competencies**: Abstract prior knowledge of programming, especially when concepts are applied to everyday situations, can hinder learners' confidence in their ability to actually write code [4].
- **Fear to use computers**: Learners may experience the negative emotion of anxiety associated with using computers, which can interfere with the learning experience [8].
- **Overwhelm**: Text-based programming using syntax can seem complicated and lead to feelings of frustration [5] and overwhelm.

3.4 Solution

Provide high-interaction, low-abstraction learning solutions using Block-Based Programming.

High-interaction, low-abstraction learning solutions provide something that enables high-frequency interactivity while remaining at a low level of abstraction. This structure can be created using Block-Based Programming, optionally combined with tangible objects such as microcontrollers. Block-Based Programming packages coding logic into individual blocks that can be assembled and nested using drag-and-drop in an editor. Using this type of programming language allows learners to create working programs without writing "real" code. This provides a low level of abstraction from code that uses typical syntax. One or more blocks represent a programming concept. The arrangement of blocks is based on the interrelationships of different concepts. For example, there might be a block to set a variable, or a block to "code" a loop. These visualizations are designed to represent concepts intuitively. For example, a loop has a shape that allows other blocks to fit in between it (see figure 9), and shows that those other blocks are contained within that loop.

The ability to drag and drop code blocks allows learners to interact with the code, such as exploring different ways to solve problems with a few drags and drops. Adding a tangible object to the mix gives learners additional feedback and a tangible representation of their solutions. Some Block-Based Programming editors offer to switch between the block-based view and the actual syntax of a particular programming language (e.g., Python), sometimes displaying both versions simultaneously. This reduces the gap between abstract concept and actual code further.

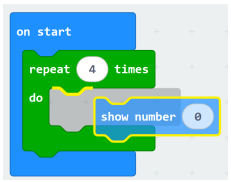

Figure 7: MakeCode Editor

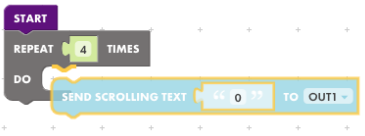

Figure 8: LittleBits Editor

Figure 9: A screenshot showing how blocks are constructed in a block-based editor to see if a block can contain another block and how they fit together.

3.5 Consequences

Benefits.

- **Computer anxiety** can be reduced through regular use with frequent positive experiences [8].
- Learner motivation through **relatable application**, **autonomy** while exploring and **high interactivity** is increased.
- Learners **gain confidence in their competence** to use programming languages by using Block-Based Programming that resembles text-based code with syntax.
- Block-Based Programming **reduces the gap** between prior knowledge and text-based code with syntax. Allowing learners to switch between block-based and actual code helps them build mental models, which reduces the gap between abstract knowledge and text-based code with syntax further.
- The interrelationships and nesting of different coding concepts, such as loops within loops, can be **represented in detail** and on a **low abstraction level**.
- **Code visualization** and **pre-defined blocks** help avoid mistakes like typos. It also rewards knowing important keywords like ”loop” by providing options on how to use them.

Liabilities.

- Block-Based Programming provides a **framework** that could be considered **limiting** by learners.
- Block-Based Programming is **limited in the concepts** that can be represented, since it is mainly usable for the representation of imperative programming.
- Despite the reduced gap, the **transition to text-based code with syntax** can still be challenging for learners and add **additional complexity**.
- Using abstract concepts with metaphors can lead to underchallenge and trigger negative learning emotions such as **boredom or frustration**.

3.6 Known Uses

In the following, known uses have been divided into two main topics, separating Block-Based Programming editors from the tangible objects that can be controlled by those editors.

[1] https://wiki.dfrobot.com/micro_Maqueen_for_micro_bit_SKU_ROB0148-EN (last accessed: May 31, 2024)

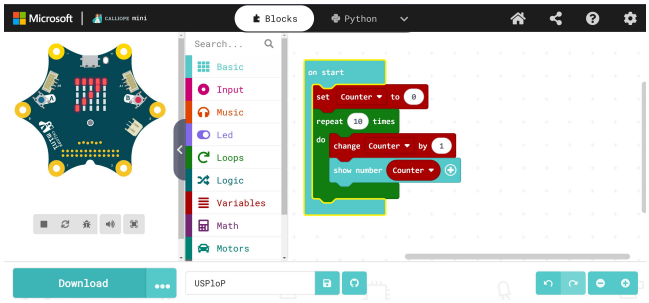

Figure 10: Screenshot of the MakeCode editor for Calliope with a code for a counter from one to ten.

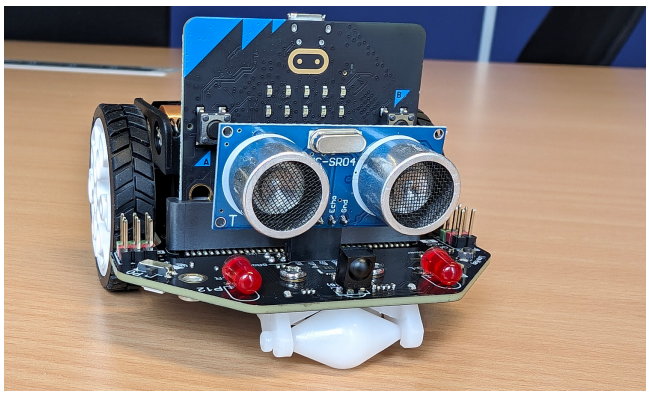

Figure 11: Image of the micro:bit with the Maqueen[1] extension built into a small robot that can move in all directions and provides sensors as a distance sensor (two ”eyes” in front).

Editors: MakeCode. MakeCode[2] is a Block-Based Programming editor. It contains an area that displays the available blocks, organized into categories that are color-coded. For example, a category for loops (shown in green in figure 10) contains several loop variants that can be added to the programming area on the right by clicking on them. The blocks on the right can be moved around using drag-and-drop and manipulated using keyboard shortcuts. As shown in the top center of figure 10, MakeCode allows the user to view the code version of the blocks in programming languages (Python and JavaScript). It allows a side-by-side view or a complete switch between views. The code can also be edited and programs can also be written completely in code. When using MakeCode with supported devices, such as the Calliope mini, it displays the microcontroller on the left side and simulates the code as much as possible on the screen.

Tangible Objects: Calliope mini, micro:bit, LittleBits. There are several tangible objects that are compatible with Block-Based Programming editors such as MakeCode. The Calliope mini[3] and micro:bit[4] are similar microcontrollers that provide a set of built-in sensors

[2] https://www.microsoft.com/de-de/makecode (last accessed: May 31, 2024)
[3] https://calliope.cc/ (last accessed: May 31, 2024)
[4] https://microbit.org/ (last accessed: May 31, 2024)

that can be controlled through programming. There are also expansion kits that include additional sensors, motors, etc. Figure 11 shows an expansion kit of micro:bit with different servo motors and a distance sensor. Both microcontrollers are easy to set up and use with a Block-Based Programming editor. When using expansion kits, libraries are available that provide blocks that can be used for programming. For example, a kit for building a robot provides blocks for setting the speed of a motor.

LittleBits[5] are color-coded modular electronic bits that snap together easily with magnets. It is possible to build a variety of working things by snapping the bits together. Figure 12 shows an example project with wiring diagram to illustrate how the bits are used together. The things learners build with littleBits can also be programmed using Block-Based Programming editors.

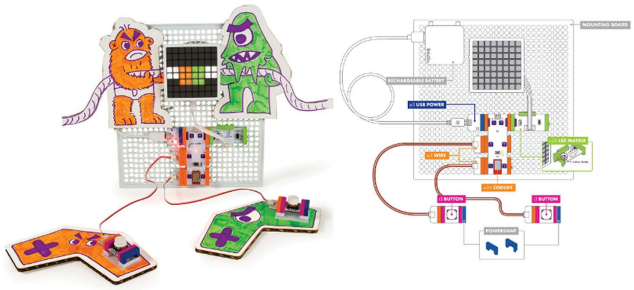

Figure 12: Pictures of a sample project "Tug of War" with the build project on the left side and the wiring diagram on the right side. The images were sourced from the project instructions provided by LittleBits.[5]

4 SELF-CONTAINED KIT

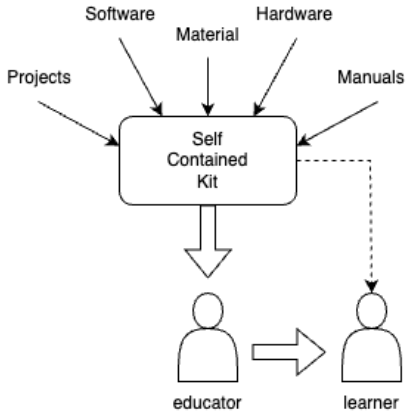

Figure 13: Components of Self Contained Kits that can be used either by learners directly or through educators.

[5]https://littlebits.com/welcome (last accessed: May 31, 2024)
[5]https://classroom.littlebits.com/lessons/invention-4-tug-of-war (last accessed: January 24, 2025)

4.1 Context

Learners, particularly beginners, aim to learn programming using games or kits. This learning takes place either at home in a playful and spontaneous setting or in learning environments such as schools, with educators preparing materials for lessons. Both settings may have limited access to additional resources.

4.2 Problem

For coding kits or games to be fully functional, additional crafting materials, electronic parts, and connectors are required. This leads to learners not being able to use the materials or incurring additional acquisition effort.

4.3 Forces

- **Motivation**: If the need to purchase additional parts is not known in advance, it quickly decreases motivation. In particular in spontaneous use, the desire to experiment is quickly dampened, reducing the likelihood that the kit will be used again.
- **Acquisition Difficulties**: It can be unclear where the required materials can be sourced. Supply shortages can also prevent the timely acquisition of necessary parts or cause prolonged unavailability.
- **Financial Resources**: Financial assessment is challenging if the kit is not usable on its own. Individuals with limited financial means or in regions with difficult delivery conditions may not have access to the required additional parts, significantly restricting accessibility.
- **Project-Dependent Additional Parts**: Different projects may require different additional parts, increasing the complexity and effort for learners and educators.
- **Preparation Time for Educators**: Educators often have limited time to prepare lesson materials. If kits require additional parts or extensive preparation, it increases the workload and reduces the time available for other teaching activities.

4.4 Solution

A kit should be used that contains all necessary components.
Self-containment encompasses both hardware and software. On the hardware side, the kit should include all necessary physical components required for the projects, such as various sensors and actuators, connection materials like cables, connectors, and breadboards, as well as microcontrollers or minicomputers to control the hardware.

On the software side, the components should be easily accessible and usable. This includes pre-installed or easily installable software libraries and frameworks needed for the projects, well-documented example codes to serve as starting points for own projects, comprehensive documentation covering all aspects of using the kit from setup to troubleshooting, and step-by-step instructions and tutorials to support the learning process and ease the entry.

In addition, manuals, project ideas, instructions, and other additional materials should be included in the kit.

4.5 Consequences

Benefits.

- Learners can start programming immediately without waiting for additional parts.
- The ability to start right away increases the motivation and interest of learners and educators.
- Since all necessary components are already included, planning and executing projects becomes simpler and more straightforward for both learners and educators.
- A comprehensive kit offers a coherent and integrated learning experience specifically tailored for beginners.
- Purchasing a complete kit provides upfront knowledge of costs, facilitating financial planning.
- Educators can save preparation time because all necessary materials are already included, allowing them to focus more on teaching and less on material acquisition and preparation.
- All-in-one kits save **sourcing effort** and provide a structured **learning path** with a range of **supported projects**.

Liabilities.

- A comprehensive kit with all necessary additional parts will be more expensive than a basic kit containing only the core programmable component, as it includes all required components.
- Standardizing the included parts may limit the diversity and possibilities of the projects that can be undertaken.
- If a part of the kit is lost or damaged, it may be harder to replace that specific part as the kit is sold as a complete set.
- Educators and learners may become reliant on the specific components provided in the kit, potentially reducing the ability to adapt to or utilize different hardware or materials in the future as well as update to newer versions.
- Sequencing of projects within all-in-one kits needs **careful planning** to ensure a smooth **learning progression**.

4.6 Known Uses

Little Bits. LittleBits[5] is an innovative electronics kit designed to teach children the basics of circuitry and engineering. It includes various electronic building blocks, such as sensors, motors, and LEDs, which snap together magnetically to create different projects and inventions. Figure 14 shows the components included in one of the kits, the Code Kit.

LittleBits is highly complete as a kit, providing all the necessary components to start building right out of the box. Each kit includes a variety of electronic modules, a power supply, and accessories like mounting boards and connector cables. Additionally, the kits often come with instructional booklets and access to online resources, offering project ideas and tutorials to help users get started and expand their skills.

Dash Robot. Dash Robot[6] is an educational robot designed for children, developed by Wonder Workshop. It can be programmed using simple, block-based coding through a tablet or smartphone app, allowing children to engage in interactive and hands-on learning.

[6] https://www.makewonder.com/en/dash/ (last accessed: May 31, 2024)

Figure 14: All contents of the LittleBits Code Kit

The Dash Robot kit is comprehensive, including the robot itself along with necessary accessories like charging cables. It also provides access to a range of free apps for programming and controlling Dash. Many kits include additional accessories like building brick connectors, challenge cards, and activity mats. The robots and one example of expansions are shown in figures 15 and 16. These elements ensure that users have everything they need to start learning and exploring various programming challenges immediately.

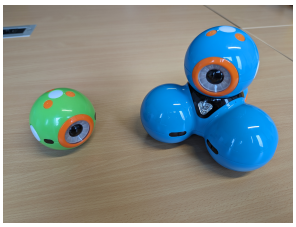

Figure 15: Dot and Dash Robots

Figure 16: Robots with Launcher Equipment

LEGO® Education SPIKE™ Prime. LEGO® Education SPIKE™ Prime[7] is a STEAM learning solution that combines LEGO® building elements, programmable hardware, and a coding platform. It is aimed at middle school students to help them develop critical thinking and problem-solving skills through hands-on learning.

LEGO® Education SPIKE™ Prime is a well-rounded kit, containing a wide range of LEGO® bricks, sensors, motors, and the programmable SPIKE™ Prime Hub, as shown in figure 17. It also includes comprehensive building instructions and lesson plans designed for educators. The kit is designed to be used in a classroom setting, but is equally effective for individual learning, providing all the necessary components to build and program various projects

[7] https://education.lego.com/de-de/products/lego-education-spike-prime-set/45678/ (last accessed: May 31, 2024)

right out of the box. Additionally, it offers access to an online community and resources, enhancing the learning experience with collaborative and extended learning opportunities.

Figure 17: All contents of the LEGO® Education SPIKE™ Prime kit

INTIA. INTIA was a research project that focused on the inclusive development of methods and technologies to help people with disabilities and educational support cope with everyday life. It ran from 2019 to 2023 and included, among other things, a suitcase.

The INTIA suitcase[8] is intended to provide initial access to technology and consists of an experience module and a design module. The ”Experience” module contains an escape game, a technology fan that acts as an encyclopedia, and various components in the form of sensors. The ”Design” module consists of various sets of cards, an idea cube that encourages creativity, and programming tiles that provide an easy introduction to a programming language. The suitcase and its contents are shown in figure 18.

Figure 18: INTIA suitcase and contents[8]

MOXD IoT Kit. The MoxdLab at Cologne University of Applied Sciences has developed an IoT kit[9] that can primarily be used for prototyping. Among other things, it is used in the introductory module ”Introduction to Media Informatics” and is intended to serve as a first introduction to programming. The kit can be used completely without code, but still offers the possibility to do so.

The IoT kit contains a Raspberry Pie in combination with a Groveshield, the Groveshield connectors, and various sensors and actuators, as shown in figure 19. It also includes a cheatsheet that lists a description of the nodes and useful combinations of nodes for various use cases. A Node RED instance runs on the Raspberry Pie, which can be used to work with the components of the kit.

Figure 19: All contents of the IoT Kit

5 PATTERNS IN ACTION

The three patterns presented form a coherent framework that covers the stages of understanding abstract concepts, interactive problem solving, and application of skills. While there are known uses for each pattern, it is important to practice them in combination to validate the framework.

In the ongoing research project ”Coding Culture in Oberberg” these patterns have been applied in practice. Below we describe the workshops we did so far and discuss the patterns simultaneously. At the end of each workshop, we asked the students for feedback using an evaluation form. The evaluation form contains several types of questions about the overall experience in the workshop, as well as detailed questions about learning to code. There are also some free text options for individual feedback. An example of the evaluation form questions is attached in appendix A.

The first half-day workshop was designed for students aged 11 to 13 and focused on teaching the bubble sort algorithm and basic programming concepts. Utilizing the *Abstract Guided Instruction Through Storytelling* pattern, students first explored different options of sorting something by giving each other precise instructions following a storytelling framework. They later used *Block-Based Programming* to construct their own story, developing a simple game in Scratch[10].

The evaluation results show that the workshop was overall perceived as very good with an average rating of 1.5 on a scale of 1 (best) to 6 (worst). The following results are reported on a scale of 1 (best) to 5 (worst). The participants were able to learn what programming is (avg 1.38) can use a block-based coding editor (avg. 1.62) and know what an algorithm is (avg. 1.54). This shows that the combination of *Abstract Guided Instruction Through Storytelling* and *Block-Based Programming* resulted in workshop participants perceiving that they had acquired programming skills.

[8] https://intia.de/intia-koffer/ (last accessed: January 21, 2025)
[9] https://moxd.io/iotkit (last accessed: January 21, 2025)
[10] https://scratch.mit.edu/ (last accessed: July 27, 2024)

The second half-day workshop was designed for a group of students between the ages of 12 and 16 with prior knowledge of abstract algorithmic thinking. The workshop aimed to teach basic programming principles and problem solving skills through algorithm development. Students had to write code for a robot to navigate a maze (see figure 20).

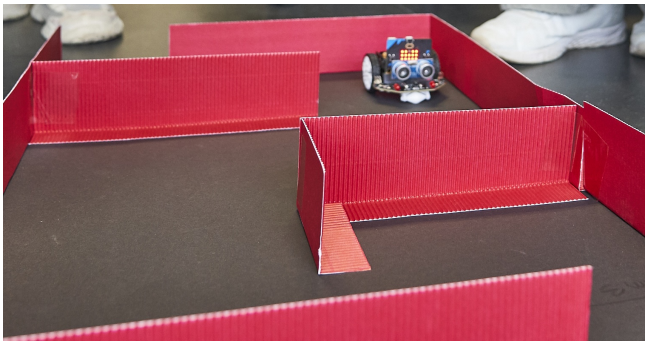

Figure 20: Maqueen robot navigating through a cardboard maze

We used the *Self-Contained Kit* Pattern by integrating a micro:bit controller with the Maqueen extension to provide a robot that could move and had a distance sensor. With the micro:bit microcontroller, the students were able to write code using the MakeCode editor that could be executed and make the robot move in a way that they had coded. We also provided craft supplies so students could make their own mazes. By using the MakeCode editor, we utilized the *Block-Based Programming* pattern, giving students a way to code their robot without having to learn coding syntax. At the beginning of the workshop, we found a solution to traverse the maze interactively with all the students by creating a program flowchart. Then the students created their own mazes and began to apply what we had conceptualized in the flowchart to block-based code. The students seemed to grasp the basics of the editor very quickly and were able to translate their ideas into code. They were able to see how the code worked when they had the robot move through the maze and figure out where it still had problems.

The evaluation results indicated that the workshop was perceived as good overall, with an average rating of 2.25 on a scale of 1 (best) to 6 (worst). The following results are reported on a scale of 1 (best) to 5 (worst). Students rated that they had acquired an understanding of the concept of programming with an avg. of 1.33 and how to use a block-based coding editor with an avg. of 1.75. However, when it comes to finding and solving problems in the code (avg. 2.42) and manipulating the code to make the robot do what the students want (avg. 2.67), there is a noticeable decrease in the ratings. This decline could be explained by the fact that an introduction to the code editor was given, but not all features were explained. For instance, the MakeCode editor incorporates a debugging feature that enables students to observe the code's operation step by step. Explaining this feature could have helped students with finding and solving problems in their code. Overall those results support the usefulness of the *Block-Based Programming* pattern. However, they also reveal the necessity of designing instructions to support individual learning processes of students.

The third workshop was designed for a group of 11-12 year olds and lasted two half days. In this workshop, we tested the concept of providing several complete projects for the students to choose from. At the end of the workshop, each group of students presented the project they liked best. The idea was to allow the students to choose the project they were most interested in and to learn about the other projects of their peers. The *Self-Contained Kit* pattern was utilized by providing each project including all the materials needed to complete it. This typically included instructions on how to execute the project, a microcontroller to run the code, crafting materials, or extensions for the microcontroller. Within each project, a block-based code editor was recommended to write the code (*Block-Based Programming*).

An interesting finding from the evaluation is that the workshop was well received overall, with an average rating of 1.5 on a scale of 1 (best) to 6 (worst). The following results are reported on a scale of 1 (best) to 5 (worst). Students rated knowing how to make a program do what they want it to do with an average of 2.54. This rather low rating could be explained by the structure of the workshop. Students were given instructions that did not include further explanation of individual blocks in the coding editor. However, students were fairly confident that they had learned how to use block-based programming (avg. 1.69) and understood basic programming concepts (avg. 1.62). This supports the usefulness of the *Block-Based Programming* pattern for understanding and applying programming concepts, but also shows the need for instructions to support optional additional explanations of certain aspects.

The workshop concept described above was also used for a weekly 1.5 hour workshop with 10 to 12 year old students. Because we used the *Self-Contained Kit* Pattern, we were able to reuse the projects we had already developed and set up the workshop quickly.

The practical application of these patterns has demonstrated their efficacy and their applicability in real-world settings.

6 CONCLUSION

In this paper, we presented three patterns: *Abstract Guided Instruction Through Storytelling*, *Block-Based Programming*, and *Self-Contained Kit*. These patterns form a framework that can be used in combination or alone to enhance learners' learning experiences and aid teachers in designing complete learning journeys. These patterns can be used while taking into account the prior knowledge of the target learners and the preconditions that teachers have to work with. The framework using the interaction and abstraction framework we provide (as illustrated in figure 1) is easily extensible by placing new patterns within these dimensions. An example of extension is the concept within "Osmo Coding Awbie" (described in section 2.6). It leverages a hands-on approach to abstraction, where physical coding blocks represent different programming commands. This would be arranged within the provided framework as intermediate to high interaction and high abstraction.

As we have shown while discussing the workshops, the patterns can help learners gain confidence in their ability to learn more abstract concepts like algorithmic thinking and apply it using

Block-Based Programming. The high level of interaction provided by Block-Based Programming can help to motivate and explore. Especially for creating learning experiences, having something that can be used out of the box is helpful and eases the process of creating learning experiences.

Future work may identify other patterns that are useful in programming education, as the field is constantly evolving. We plan to address the specific challenges that emerged from the workshop evaluations to expand the patterns and refine the framework proposed in this paper.

ACKNOWLEDGMENTS

The project "Coding Culture Oberberg" is funded by the Hans Hermann Voss Stiftung.

We are very grateful for the valuable feedback of our shepherd YC Cheng. His expertise in software engineering education and pattern writing greatly contributed to the improvement of this paper. We appreciate his insightful input, which has improved the overall quality of our work. Additionally, we are thankful for the valuable feedback we received during the Writer's Workshop at PLoP 2024.

REFERENCES

[1] Christopher Alexander. 1977. *A pattern language: towns, buildings, construction.* Oxford university press.

[2] Mary McCullum Baldasaro, Nancy Maldonado, and Beate Baltes. 2014. Storytelling to teach cultural awareness: The right story at the right time. *LEARNing Landscapes* 7, 2 (2014), 219–232.

[3] Anja Bertels and Dominik Deimel. 2024. Storytelling in the Context of Education: A Pattern Language. In *Proceedings of the 28th European Conference on Pattern Languages of Programs* (Irsee, Germany) *(EuroPLoP '23)*. Association for Computing Machinery, New York, NY, USA, Article 22, 16 pages. https://doi.org/10.1145/3628034.3628056

[4] Cornelia Connolly, Eamonn Murphy, and Sarah Moore. 2008. Programming anxiety amongst computing students—A key in the retention debate? *IEEE Transactions on Education* 52, 1 (2008), 52–56.

[5] Yue Hu, Cheng-Huan Chen, and Chien-Yuan Su. 2021. Exploring the Effectiveness and Moderators of Block-Based Visual Programming on Student Learning: A Meta-Analysis. *Journal of Educational Computing Research* 58, 8 (2021), 1467–1493. https://doi.org/10.1177/0735633120945935 arXiv:https://doi.org/10.1177/0735633120945935

[6] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin* 41, 3 (2009), 161–165.

[7] Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. 2018. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education* 62, 2 (2018), 77–90.

[8] Keith Nolan and Susan Bergin. 2016. The role of anxiety when learning to program: a systematic review of the literature. In *Proceedings of the 16th koli calling international conference on computing education research*. 61–70.

[9] Richard M Ryan and Edward L Deci. 2000. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist* 55, 1 (2000), 68.

[10] Álvaro Santos, Anabela Gomes, and António Mendes. 2013. A taxonomy of exercises to support individual learning paths in initial programming learning. In *2013 IEEE Frontiers in Education Conference (FIE)*. IEEE, 87–93.

[11] Jeroen JG Van Merrienboer and John Sweller. 2005. Cognitive load theory and complex learning: Recent developments and future directions. *Educational psychology review* 17 (2005), 147–177.

A EVALUATION FORM QUESTIONS

Example of the evaluation form questions we use to evaluate each workshop. The evaluation form varies for each workshop depending on the content. In particular, the items under item 3 and the free text option vary. The example below shows items for the multi-project workshop design.

(1) How would you rate your experience during the workshop? Give your answer in a school grade, i.e. 1 means very good and 6 means unsatisfactory.
(2) Please rate whether you agree with the following statements (fAgree wholeheartedly, Agree, Partially, Disagree, Do not agree at all)
   - I know what programming means
   - I can do block-based programming
   - I can make sure that a program does what I want it to do
   - I know basic programming concepts
(3) How would you rate the following parts of the workshop? Give your answer in a school grade, i.e. 1 means very good and 6 means unsatisfactory.
   - Try out the projects
   - Present projects
   - See projects from others
   - Explanations
   - Try things out on your own
(4) Which of the projects did you work on?
(5) Which of the projects did you like best and why?
(6) Which of the projects did you like the least and why?
(7) Do you have any other comments on the workshop, what you liked / didn't like?