Case Study: Design Strategies for Enabling Visual Application Blocks of Bluetooth Library

TAI-LIANG CHEN1, (Student Member, IEEE), I-AN SU1, AND JENQ-KUEN LEE1
1Department of Computer Science, National Tsing Hua University, Hsinchu 300044, Taiwan
Corresponding author: Tai-Liang Chen (e-mail: tlchen@pllab.cs.nthu.edu.tw).
This work was sponsored in part by Airoha Technology Corp., Hsinchu, Taiwan, and supported by the Ministry of Science and Technology in Taiwan under Grant MOST 108-2221-E-007-037-MY3.

ABSTRACT Block-based tools can make it easier for beginners to learn programming by arranging blocks. Their block concept and extensible characteristics make block-based designs suitable for introductory programming. However, block-based tools are easy to use but difficult to extend to include the complicated applications of vendor libraries or domain-specific libraries. This paper proposes a solution to block design strategies that enable designers to redefine and design domain-specific libraries to generate visual application blocks. This research uses the Bluetooth library as a case study and constructs the BlocklyTooth integrated development environment (IDE) using the Google Blockly tool. A feasibility evaluation of this study produced the following findings: (1) for the strategies, the usability of the usability metric for user experience (UMUX) and the practicality of the customized theme received satisfactory responses; and (2) for the IDE, the usability of UMUX and the value implication of the customized theme received positive reviews. In addition, the experiment on resource impact revealed that the strategies are effective in improving resource utilization, in which the quantity of code is reduced by 70.2% and 89% in terms of the block count and lines of code, respectively, and the system memory is reduced by 51.8% in terms of the dynamic code size. The results indicate that the proposed solution can be a design reference guideline for block-based tools and can effectively abstract the complex Bluetooth library into blocks.

INDEX TERMS Block design strategy, Blockly, Bluetooth, embedded development, graphical user interface (GUI), integrated development environment (IDE), internet of things (IoT), visual programming language.

I. INTRODUCTION
Block-based tools can make it easier for beginners to learn programming by arranging the blocks in a spatial manner, which has increased the attention on the use of such tools in visual programming languages (VPLs). In the development of a block-based VPL, the abstract block concept is the core aspect of block design. The Scratch language [1], developed in 2003 as a good example, has interactive elements of abstract graphical symbols and is a representative block-based tool for teaching children about VPLs [2], [3].

Because the abstract block concept of block-based tools can reduce the learning threshold, this concept also extends to providing training on introductory programming. Roque [4] proposed the OpenBlocks framework in 2007, which was designed based on the Java library to focus on language application design through its extensible features. The visual development environment App Inventor1, based on the extensible framework of OpenBlocks, was subsequently developed in 2010 and can be used to build apps. In 2012, Blockly2, released by Google with the aforementioned abstract block elements and extensibility, is currently one of the most popular block-based tools used in application development.

In addition, Blockly can also support converting text-based programming languages into block-based editors [5]–[9]. This tool can also be used to develop embedded program contents for Internet of Things (IoT) environments [10]–[13] or to create a platform editor with domain-specific program-

1https://appinventor.mit.edu
2https://developers.google.com/blockly
ming [14]–[22]. A good example of this is BlocklyDuino, which integrates the above scenarios by combining Blockly as the framework with the IoT environment, supporting C language blocks and various custom IoT requirements of Arduino applications. Thus, BlocklyDuino can simplify the development of Arduino IoT environments. In addition, the Blockly tool can be combined with an artificial intelligence (AI) compiler to develop a deep-learning development environment suitable for AI computing [23], [24]. However, such easy-to-use block-based tools are difficult to extend to include the complicated applications of vendor library technologies or domain-specific libraries. In general, block-based tools use imperative programming [25] (procedural programming, such as C language) to design a block version of the programming language for users to learn to program. For users with a programming background, learning the libraries of complicated and professional techniques through these tools will not be more efficient than direct typing programming, nor can it fully exploit the advantages of the abstract block.

The purpose of this paper is to propose a solution to block design strategies that can be used as reference guidelines for block designers that enable them to redefine and design domain-specific libraries to generate visual application blocks (VABs). This research aims to address the problem that block-based tools are difficult to extend to include vendor library technologies or domain-specific libraries. We utilize the industrial Airoha Bluetooth SDK library as a case study for generating Bluetooth VABs using these strategies. Bluetooth VABs are then integrated with a commercially available development environment to construct the BlocklyTooth Integrated Development Environment (IDE). This solution is displayed through the Blockly tool to illustrate the contents of two classifications of the strategies based on the block shape and the domain-specific characteristics: (1) using block shaping, focusing on the design of the block shape, and reshaping optimization. For the current block-based applications constructed with imperative programming, this classification can be improved to support the program structure of the domain-specific library, such as optimizing the block shape, text descriptions, and the number of items in the block stack; and (2) using domain-specific blocks, focusing on the domain definition of the application behavioral design and modularization design and providing a design reference for converting text-based programming into blocks.

To evaluate the feasibility of the strategies, we designed a feasibility evaluation methodology to validate this solution and obtained evaluation results: (1) for the strategies, the usability of the usability metric for user experience (UMUX) [26] and the practicality of the customized theme received satisfactory responses, and (2) for the IDE, the usability of UMUX, and the value implication of the customized theme received positive reviews. In addition, the experiment on resource impact revealed that the strategies are effective in improving resource utilization, in which the quantity of code is reduced by 70.2% and 89% in terms of the block count and lines of code, respectively, and the system memory is reduced by 51.8% in terms of the dynamic code size. The results indicate that the solution can be a reference guideline for block-based tools and effectively abstract the complex Bluetooth library into blocks.

The innovation of this research is that block design strategies can help block-based tools expand the use of complex domain-specific libraries. A block-based tool that contains a domain-specific library enables block users with a programming background but no relevant domain-specific knowledge (e.g., Bluetooth SDK library) to focus more on learning domain-specific technologies through its block stack. These strategies can also improve employee training in industry to meet practical requirements.

This study focuses on providing block design strategies for block designers in industry. Block designers are developers who are responsible for developing block-based tools and have a programming background and can use block design strategies as a reference to design complicated domain-specific libraries or improve the naive block design caused by imperative programming. These strategies can be applied to domain-specific libraries and modules with a restricted scope and clear goal, which is conducive to new employee orientation and tool development.

The remainder of this paper is organized as follows: Section II describes the related research. Section III uses an example to explain the motivation for distinguishing between block design with imperative programming and VABs. Section III describes the proposed block design approach for block design strategies. Section V presents a design guideline and the VABs with the industrial Airoha Bluetooth SDK library, as well as illustrating the BlocklyTooth IDE and its use cases. Section VI introduces the evaluation methodology for block design strategies and the BlocklyTooth IDE. Finally, Section VII presents the evaluation results, including the strategy and IDE parts. Section VIII discusses the findings, and conclusions are drawn in Section IX.

II. RELATED WORK

Block-based programming is a visual expression of VPLs that uses block-stack behavior to construct program content. The Scratch language developed in 2003 is a good example; it is currently one of the most widely used VPLs in children’s learning software and has been used in the development of many learning programs and applications [27]–[35]. Another representative block-based tool is Blockly (shown in Fig. 1), which is a web-based VPL editor with imperative programming and is an open-source approach proposed by Google in 2012. Its block mode can be converted into an actual programming language mode (red box 1 in Fig. 1). It is different from the Scratch language in that it can support the design or extension of self-defined module blocks for programming languages or domain-specific development.

3https://github.com/BlocklyDuino/BlocklyDuino
4https://www.arduino.cc/
environments. The block design modules contain eight basic syntax categories (red box 2 in Fig. 1), which allow users to stack the block contents of applications. Blockly is used extensively in the development of training programs and applications [7], [36]–[44].

A. BLOCK-BASED PROGRAMMING

One commercial application is in LEGO developing the Mindstorms EV3 environment, which mainly provides software and hardware development for its machine building block equipment and is currently one of the most successful cases of using the block-based concept in a business application. The well-known open-source development software includes App Inventor, which is an environment specialized in developing Android apps; Code.org, which is a nonprofit website with open learning resources; Visualino, which provides an Arduino visual programming environment; and Wyliodrin, which provides IoT solutions for use in industry. BlocklyDuino is an example that modifies the web-based Blockly framework and applies it to an Arduino embedded environment that can convert the blocks into C language contents and support general Arduino embedded modules. Its features not only support a professional programming language but also extend self-defined block modules for domain-specific technology and libraries. However, these open-source tools were originally designed for beginners without a programming background and focused on basic programming training. The main difference between this study and others is that the group of block design strategies proposed by this research can assist block designers in developing understandable blocks for professional domain-specific technologies in the form of a case study. We designed each strategy according to the principles of the high-level language structure defined by programming languages [46], which improves the block structure design based on imperative programming.

Ashrov et al. [47] proposed incorporating the visual programming tool of Google Blockly into a single-threaded implementation of behavioral programming (BP) and the design pattern application for developing reaction systems. BP means that these independent modules modularize the event requirements in a decentralized manner, which execute interactively at runtime. Each module itself represents a behavioral scenario of event sequences that have multiple objects in the system. Their BP implementation uses the B-thread (executed as an ordinary Java thread) block of the pattern application to present interactive application behavior using the concept of a cross-cutting scenario. However, this is still a way of stacking block content with imperative programming, which is a burden for users with a programming background. From the perspective of program structure, the application of design patterns can be further discussed through the definition of block design. The main difference between this paper and those studies is that the support of block design strategies helps to present domain-specific procedural or scenario content in a visual block form.

Tomlein and Grønbæk [48] proposed GIMLE (Graphical Installation Modeling Language for IoT Ecosystem), which uses visual programming to provide a domain-specific approach by expressing ontological requirements. In addition to the cooperation of industry partners and consultations with domain experts, their research further analyzed actual use cases of water movement and cooling technologies to guide visual language design. Weintrop et al. [16] developed a programming tool named Robot Blockly, which is a block design for robot programming of a one-armed industrial robot with professional technology. This tool can make it easy for novices to develop and effectively solve human resources in industrial application development. In addition, their research helped novices achieve basic robot behaviors without any programming experience. These two studies both proposed visual expression methods and applied them to the use of industry-specific technology. The main difference between this paper and the previous two studies is that this research proposes a group of block design strategies for block design-

B. DOMAIN-SPECIFIC TECHNOLOGIES

This paper proposes a group of block design strategies and discusses the feasibility of applying these strategies to industrial domain-specific technologies in the form of a case study. We designed each strategy according to the principles of the high-level language structure defined by programming languages [46], which improves the block structure design based on imperative programming.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
ers as a design guideline, rather than only discussing visual expressions for industry-specific knowledge in the form of a case study. Meanwhile, users with a programming background can focus on learning the principles and concepts of domain-specific technologies through a well-designed block IDE based on block expressions.

Bajzek et al. [49] proposed the low-cost MUzECS (Marquette University Exploring Computer Science) teaching curriculum based on Blockly and BlocklyDuino combined with the Arduino IDE. The design of MUzECS replaces the previously used block curriculum, which cannot convert blocks into text content. Blanchard et al. [50] proposed an Amphibian tool that has a dual-modality IDE plug-in, which is a block design based on the program structure. Even if the text has changed, their dual-modality IDE can still provide correct switching between blocks and text. However, the block-based development environment proposed in the above research still has an imperative programming block design. When beginners use such IDE in an attempt to obtain good basic programming concepts and transition to professional programming, these environments may impede their continued use. The main difference between this paper and those studies is that the IDE platform that integrates the commercially available development environment designed in this research allows users with a programming background to learn the knowledge of domain-specific technologies through blocks generated by block design strategies. Meanwhile, the IDE platform can be converted into a text-based mode to use an industrial programming environment.

Pratomo and Perdana [51] proposed the Arduviz visual programming IDE, which combines Google’s Blockly and Arduino IDE and is bundled with avr-gcc and avrdude as the compiler and uploader. The Arduviz IDE integrates most of the advantages of Ardublock [10] (incorporation of the Arduino IDE into OpenBlock) and miniBlock [11] (incorporation of the self-developed IDE and blocks), such as instant code generation and standalone development environments. However, the previous studies and open-source tools still focus on the block design of program generation and lack blocks of expertise to provide domain-specific technologies. The main difference between this paper and those studies is that this study proposes an IDE that supports block expressions generated by block design strategies for users with a programming background to study professional knowledge. The text-based programming function of this platform provides users with the capability to advance the development of professional content.

C. BLUETOOTH LOW ENERGY

Because the Bluetooth SDK library of Airoha Technology has features of a distinct domain-specific protocol structure, a complete and small scope of code content, and important transmission technology in the IoT embedded development, it is suitable for the case study of this research. We further provide brief information about the industrial Airoha Bluetooth SDK library used in the present case study. Bluetooth low energy (BLE) technology is a subset of Bluetooth 4.0 for low-power applications and mobile IoT environments. Ongoing research on the BLE protocol stack continues to bring technological innovation to the development of the IoT [52–55]. As shown in Fig. 2, the BLE protocol stack architecture includes three parts: applications, host, and controller. Detailed information can be obtained from the Bluetooth specification [12] website, whereas below, we define each part of the stack and briefly introduce the relationships.

First, the application part of the BLE protocol stack includes an application layer that uses the data transmission services provided by the host layer and interacts with the host part only. Second, the host part is a software protocol comprising five layers: (1) the generic access profile (GAP) addresses access modes and procedures, including finding devices, connection establishment, connection suspension, enabling security functions, and device settings; (2) the generic attribute profile (GATT) establishes the connection and data transmission between the two devices set as the master and slave; (3) the attribute protocol (ATT) is responsible for retrieving the transmission of all data-containing services; (4) the security manager (SM) implements secure connections and encryption; and (5) the logical link control and adaptation protocol (L2CAP) provides data packet services to the upper layers for connection-oriented and connectionless services, as well as multichannel operations. Finally, the controller part comprises the following three layers: (1) the host controller interface (HCI) provides communication between the host and controller through an interface, such as the software application programming interface (API), universal asynchronous receiver transmitter (UART), serial peripheral interface (SPI), and universal serial bus (USB) interface; (2) the link layer (LL) controls the hardware to be in one of five

![FIGURE 2. Overview of the BLE protocol stack.](https://www.bluetooth.com/specifications/specs/)

---

10 [http://blog.ardublock.com](http://blog.ardublock.com)/
11 [http://blog.minibloq.org](http://blog.minibloq.org)/
states (standby, broadcast, scan, initialization, or connection); and (3) the physical layer (PHY) is a technique for hardware radio communication.

### III. INTUITIVE MOTIVATION EXAMPLE

We use Bluetooth data transmission (TRSPX), which is specifically used to detect Bluetooth modules and provide data transmission, as a motivation example using an intuitive manner to explain the difference between block design with imperative programming and our VAB solution (see Fig. 3). In general, we split the program content first and then define the split strings to the corresponding block shape during the design block process. In this way, it can guarantee that the result of stacking blocks can restore the program content. Fig. 3-A-(a) shows the example code of a C program for TRSPX from the Industrial Airoha Bluetooth SDK library. Fig. 3-A-(b) is the stacking block result using BlocklyDuino of imperative programming. In other words, Fig. 3-A-(b) is the stacking result of a user using 24 naive blocks and taking 5 to 10 min. This result is the same as the program structure of Fig. 3-A-(a). This situation will greatly increase the complexity of the development time and block space and be operationally confusing for users with a programming background. Meanwhile, this block stack is not more effective than the form of direct typing.

With the above findings, the solution of this paper can improve the above situation. This solution has two aspects: one is to reorganize the program structure. The block design strategies and a domain-specific technique (such as the Bluetooth protocol stack layer in Fig. 2) provided in this paper are the basis for our reorganization. The second is to find and split appropriate structural features according to the block design strategies for the reorganized program content and design corresponding blocks. We then design abstract names that conform to the domain knowledge for these blocks.

Fig. 3-B-(a) shows the reorganized TRSPX program consisting of a header file and calling function APIs. The actual source code of these APIs, which include all Bluetooth behaviors, IoT-related applications, and modules, are included in the header file (Line 1 in Fig. 3-B-(a), blockly_tooth.h).
The advantage of designing the header file is that the block user only needs to stack and arrange the blocks according to the concepts of domain knowledge, without compilation errors due to the wrong order of the block structure. Finally, we split the program content of Fig. 3-B-(a) and design the corresponding block shape.

Fig. 3-B-(b) is the stacking result using our designed block-based tool. We find that the names of these blocks present abstract descriptions, which are significantly different from the block names in Fig. 3-A-(b), given in an imperative programming manner. This difference is because we define the abstract name of each block according to these strategies combined with the domain knowledge (Bluetooth protocol stack and system execution flow). However, the split strings, APIs, and substructure fragments set inside those blocks can still be restored to full program content (such as Fig. 3-B-(a)) after stacking. The content in Fig. 3-B-(b) shows that only four VABs are needed to express the content of TRSPX, and this process takes only 1 to 2 min to complete. The four VABs generated by these strategies can greatly reduce the complexity of the development time and block space compared with Fig. 3-A-(b).

### IV. BLOCK-BASED DESIGN APPROACH

This section describes the block design strategies. As shown in Fig. 4, we propose two classifications of block-shaping and domain-specific blocks. They have a visual design and domain-specific implications. These classifications are based on two aspects in block design: the first is to propose a suitable block shape corresponding to the procedural content, name text expressions, multiple function calls, etc. and to adjust the block shape. The second is that blocks can be designed based on domain-specific behaviors or APIs.

#### A. BLOCK SHAPING

The first aspect is related to the shape of the block, called block shaping, which focuses on block shape design and shaping optimization and includes six strategies: direct forwarding, polymorphism intersection, parameter overloading, input listing, parameter division, and command comprehension. For the current block-based applications constructed with imperative programming, the strategies of this classification can be improved to support the program structure of the domain-specific library, such as optimizing the block shape, name text expressions, and the number of items in the block stack. The six strategies are described below; a program fragment from the Industrial Airoha Bluetooth SDK library is presented as an example, and its block shape is shown.

1) **Direct Forwarding**

The principle of direct forwarding is that the nested procedure is forwarded to the final procedure directly. It can reduce the number of call layers generated by the nested procedure in the block design, thereby reducing the number of block stacks and improving the block set that is difficult to read visually. Fig. 5 provides an example of this strategy. Fig. 5-(a) is a C program fragment of the Bluetooth initial setting. The AB_BSP_initEx() function in red box 1 needs to pass in the structure parameter bsp_init_parm from red box 2. Red box 2 covers three structural parameters (red box 2-1: clock_parm, printf_parm, and BT_parm), which are from red box 3. The blocks are stacked according to the program function logic in Fig. 6, in which red box 1 indicates AB_BSP_initEx() and requires a value input BSP_init_parm, and red box 2
indicates BSP_init_parm and requires three inline parameters as input (clock_parm, print_parm, and BT_parm). Because this naive block design makes the blocks appear lengthy, the entire block set is difficult to read visually. In this case, the function call can be directly forwarded to the parameters of the final function call. AB_BSP_initEx() can directly use the three inline parameters more concisely, without the need for this additional BSP_init_parm block. Therefore, we can use this strategy to readjust the block for this program fragment. The final block shape is illustrated in Fig. 5-(b).

2) Polymorphism Intersection
The principle of polymorphism intersection is to intersect the names of the homogeneity types into a unified expression and use an option to identify the different parts. This simplifies the repeated design of homogeneous blocks in the block design, thereby reducing the number of block expressions. Fig. 7 shows an example of such a strategy. Fig. 7-(a) is a C program fragment, which is coded for two functions that have the same prefix and the same arguments. This example focuses on the two red boxes (red boxes 1 and 2) in Fig. 7-(a), which represent the characteristic notify and characteristic indicate of the GATT layer in the Bluetooth BLE power-saving mode. These two red boxes have the same BLE_gatt_charact_prefix (red box 3) to be combined, and the different parts (notify and indicate) can be represented by the list in the block (red box 4). Therefore, we can use this strategy to design a new block for this program fragment. The final shape of the block is illustrated in Fig. 7-(b).

3) Parameter Overloading
The principle of parameter overloading is to share the same interface expression by receiving different types or numbers of parameter lists. The block design is more flexible than programming languages with overloading concepts. When the parameter types belong to these four types (arbitrary value, list, function pointer, or structure), this strategy is used to design a shared interface, thereby simplifying the complexity of block shapes and reducing the number of block stacks. Fig. 8 provides an example of this strategy. Fig. 8-(a) is a C program fragment of the advertising setting in the BLE power-saving mode of Bluetooth. In red box 1, parameter adv_parm declares the BLE_gap_adv_params structure type and contains three variable members (ad_type, self_addr_type, and channel_mask; red box 2), which can be repeatedly given different parameter values. In this case, it meets the structure type definition of this strategy; therefore, we can use the strategy to design a new block for this program fragment. The final block shape is illustrated in Fig. 8-(b).

4) Input Listing
The principle of input listing is to present a certain number of items with clear goals as a list of options. It can reduce the repeated design of multiple fixed options for a specific function in the block design, thereby improving the visual representation of block expressions and reducing the number of blocks. Fig. 9 provides an example of this strategy. Fig. 9-(a) shows a naive block design named BT_Log_Enable, whose input value comes from one of the five control logging levels. In this case, the area and the number of items in this naive block stack will increase, making it difficult to read visually. Hence, we can combine the five control logging levels into a menu list according to the strategy. The final block shape is illustrated in Fig. 9-(b).
5) Parameter Division
The principle of parameter division is to summarize the same types of features from the complex parameters of the specific functions and divide them into independent interfaces. It can simplify the complexity of multiple input parameters of the block itself in the block design, thereby reducing the area of the block itself and further shaping to improve modularization. An example of this strategy is illustrated in Fig. 10. Fig. 10-(a) shows a naive block design named Create Service, which has five value inputs: serv num, serv UUID, char UUID, char property, and char data. Because this naive block contains two kinds of value input groups, Serv and Char, the value input design is not sufficiently modular, which reduces the readability of that block. Therefore, we can use this strategy to split the naive block. Furthermore, the shape of the block could be adjusted after splitting. For the three value inputs, char UUID, char property, and char data, with the same type of char as an example, this strategy can be used to reshape them to generate an inline input. The final block shapes are illustrated in Fig. 10-(b) and 10-(c). The block in Fig. 10-(c) can be an input value of the char type in Fig. 10-(b).

6) Command Comprehension
The principle of command comprehension is to construct expressions that are easy for humans to read and comprehend. It can improve the incomprehensible literal expressions of the block in the block design, thereby reshaping, renaming, and rearranging the block shape and its literal expressions. Fig. 11 provides an example of this strategy. Fig. 11-(a) shows a naive block named AB_UART_, which can turn on the UART channel, but this block is difficult to understand its literal meaning immediately. The naming convention is a character sequence of the identifier that can be abbreviations or short titles consisting only of letters, digits, and underscores. In general, this naming convention is used for variables, types, and functions in the source code and documents. Redefining the content of this naive block design using this strategy produces Fig. 11-(b), with the channel name added and the content of the modified menu list. However, these changes still do not provide an intuitive arrangement; thus, the block can show the final shape through rearrangement and reshaping, as illustrated in Fig. 11-(c), which allows users to rapidly understand its function.

B. DOMAIN-SPECIFIC BLOCKS
The second aspect is related to the expressions of domain-specific programs, called domain-specific blocks, which focus on a domain-specific definition of the behaviors and modularization with the application and include four strategies: virtual handling, behavior module, function return, and indicate repetition. The strategies of this classification can provide a design reference for converting text-based programming into blocks. The four strategies are described below, and the industrial Airoha Bluetooth SDK library is used continually as an example to demonstrate them.

1) Virtual Handling
The principle of virtual handling is to present an abstract drive description by providing a visual expression. When functions such as services, applications, or APIs in a domain-specific technology need to be initialized and executed, using a visual block can present an action that wants to perform them. The strategy can construct a visual block, thereby driving a domain-specific function. Fig. 12 shows an example. Fig. 12-(a) is a C program fragment of the Bluetooth API services without a return value, including the battery, TRSPX, and device information. When using these services, a visual expression is lacking that can use these services’ intentions. We use this strategy to design a visual block that does not need to import any source code for execution. The final block shape is illustrated in Fig. 12-(b), and the three services in Fig. 12-(a) can be one of the block inputs (as indicated by the dashed line).
2) Behavior Module

The principle of the behavior module is to construct a reusable domain-specific process into an independent execution interface. When a program fragment specifically describing a state, behavior, or algorithm in a domain-specific technology can be reused, the strategy can create a block to correspond to it. Fig. 13 shows an example of this strategy. Fig. 13-(a) is a C program fragment for changing the advertising interval behavior for Bluetooth (red box), which is a reusable domain-specific process. This strategy can be used to design a new block for this program fragment. The final block shape is illustrated in Fig. 13-(b).

3) Function Return

The principle of function return is to put the necessary information into a domain-specific interface and return the result. When an API in a domain-specific technology has a return service, the strategy can construct a block to represent the return result after executing the API. Fig. 14 presents an example of this strategy, which shows a function with a return value. Fig. 14-(a) shows a C program fragment that demonstrates the initialization of TRSPX. In the red box of Fig. 14-(a), BLE_get_attribute_length is an API that is used to obtain the length values. In this case, the API meets the definition of this strategy for designing a new block. The final shape of the block is illustrated in Fig. 14-(b).

4) Indicate Repetition

The principle of indicate repetition is to construct a domain-specific structure with a loop state into an interface that can execute various application requirements repeatedly. When a domain-specific technology contains a loop function such as a while loop, this strategy can construct a block to repetitively use applications that require routine execution. Fig. 15 shows an example of this strategy. Fig. 15-(a) is a C program fragment of a Bluetooth device whose default setting includes a hardware watchdog timer mechanism (DRV_WDT_Enable and DRV_WDT_Kick) in the statements of the while loop (red boxes). In this case, the content of the red boxes meets the requirements of this strategy, which can be used to design a new block for this content. The final shape of the block is illustrated in Fig. 15-(b).

V. DESIGN GUIDELINE AND CASE STUDY

To achieve the feasibility of the above-described strategies, we present a design guideline of the standard design flow and use this guideline to design the Airoha Bluetooth SDK library.
and generate VABs. We also provide an overview of an IDE integrated with VABs. Finally, we use VABs to implement the two use cases.

A. DESIGN GUIDELINE
This section presents a flow of the design guideline for block designers to reference and follow.

a: Select Target
The target applicable to the solution in this paper must be constrained in scale, primarily in technologies with unique domain-specific protocol structures and complete and small-scale libraries, such as the Bluetooth library.

b: Refactor Structure
The objects of reorganization include the programs and built-in examples of the technology itself. The reorganization specification is based on the principle of block design strategies and the application of domain-specific knowledge, which reviews the program architecture to reorganize code to create smaller and more modular components without altering the functionality. Its process also involves putting behaviors, call functions, and specific constructs that can be modularized in the program into a header file.

c: Make Blocks
According to the principle of block design strategies, suitable strings, APIs, and substructure fragments are found and split from the reorganized code content to design the corresponding blocks. Then, the abstract name of each block is assigned by these strategies combined with the domain knowledge and system execution flow.

d: Verification
Benchmarking is based on these built-in examples of this technology. Correctness is verified by stacking and compiling the contents of blocks with the same functionality as the built-in examples.

B. VISUAL APPLICATION BLOCKS
This section reports on VABs using the industrial Airoha Bluetooth SDK library as a case study. These results were designed according to the block design strategies and the design guideline and implemented using Google’s Blockly Developer Tools. The Bluetooth library is sourced from the commercially available Airoha SoC AB1600-Bluetooth 4.2 with Bluetooth specification 4.2 produced by Airoha.

### Table: Block Design Strategies

| Classification | Handle | Shape | Strategies | Stack | Description |
|----------------|--------|-------|------------|-------|-------------|
| Bluetooth service kit | Service | | Parameter Overloading, Command Comprehension | GATT | Provides battery service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides device information service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides generic access service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides generic attribute service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides human interface service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides LED control service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides TRSPX service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides link loss service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides immediate alert service |
| | | | Parameter Overloading, Command Comprehension | GATT | Provides TX power service |
| | | | Virtual Handling, Command Comprehension | None | Triggers the service above |
| | | | Function Return, Command Comprehension | GATT | Creates the custom service |

FIGURE 16. Overview of the Bluetooth service kit and the associated statuses.
Technology. In addition, to ensure that the blocks are suitable for industrial domain-specific applications, we also incorporate the stack concept of the BLE protocol stack (see Fig. 2) designed using Bluetooth SIG. Based on the above approach, we designed 30 VABs under three classifications: service, configuration, and application kits. To improve the flexibility of the VABs when stacking, especially for initialization and activation states, trigger services and handle Bluetooth events have a notch at the top of the block. The purpose of this notch is to make a stack flexible (floating), allowing the stack to comprise general embedded IoT program blocks without being restricted by the placement order. These three classifications are explained below.

1) Bluetooth Service Kit

The service kit we defined provides connection establishment and transmits service contents between Bluetooth devices from the industrial Airoha Bluetooth SDK library. Fig. 16 lists the 12 VABs with the block expressions in this kit, with every block being consistent with the block design strategies and BLE protocol stack layers, and the details are described in the table. The trigger services VAB is designed by using the strategies of virtual handling and command comprehension and does not belong to any BLE layer. It has the concept of enabling it to express the behavior of enabling the above blocks. A custom self-defined VAB is designed using the strategies of function return and command comprehension and belongs to the GATT layer.

2) Bluetooth Configuration Kit

We designed the configuration kit based on hardware settings, including the retrieval of the hardware attribute information, the setting of hardware radio signal transmission and reception, and the conversion of hardware radio signals into digital packets. Fig. 17 lists the 10 VABs with the block expressions in this kit, with every block being consistent with the block design strategies and BLE protocol stack layers, and their information is described in the table. An initial Bluetooth VAB has two different block shapes: (1) with value input parameters, using the strategies of direct forwarding

| Classification | Handle | Shape | Strategies | Stack | Description |
|----------------|--------|-------|------------|-------|-------------|
| Bluetooth service kit | Generic attribute length | ![Image](image1.png) | Function Return, Command Comprehension | ATT | Sets the attribute's length |
| | Get Attribute Length | ![Image](image2.png) | Function Return, Command Comprehension | ATT | Gets the attribute’s length |
| Hardware information | Initial Bluetooth | ![Image](image3.png) | Direct Forwarding, Command Comprehension | GAP & LL | Initials Bluetooth with parameters |
| | Enable Bluetooth Log | ![Image](image4.png) | Function Return, Command Comprehension, Input Listing | LL | Enables the Bluetooth log with level |
| | Get Number of Bluetooth Pending TX Data | ![Image](image5.png) | Function Return, Command Comprehension | LL | Gets the number of Bluetooth pending TX data |
| | Handle Bluetooth Events | ![Image](image6.png) | Function Return, Command Comprehension | GAP & LL | Initials Bluetooth with default parameters |
| | Indicate Repetition, Command Comprehension | ![Image](image7.png) | LL | Handles Bluetooth events for program |
| | Generic attribute value | ![Image](image8.png) | Polymorphism Intersection, Command Comprehension, Input Listing | GATT | Gets the characteristic indication or notification value from handler |
| | Logical link control & adaptation protocol | ![Image](image9.png) | Function Return, Command Comprehension | L2CAP | Initial L2CAP with given channel ID |
| | Host controller interface | ![Image](image10.png) | Function Return, Command Comprehension | HCI | Initial HCI with given controller count |
and command comprehension and belonging to the GAP and LL layers; and (2) no input parameters, using the strategies of function return and command comprehension and belonging to the GAP and LL layers.

3) Bluetooth Application Kit

The last part is the application kit for building applications based on various IoT scenarios and industrial Bluetooth functions. Fig. 18 lists the eight VABs with the block expressions in this kit, with every block being consistent with the block design strategies and BLE protocol stack layers, and their detailed information is described in the table. They all use the strategies of function return and command comprehension, but the protocol stack layer is supported in a slightly different manner. The stack column in the table provides detailed information.

| Classification          | Handle                        | Shape                           | Strategies                                      | Stack                    | Description                                                                 |
|-------------------------|-------------------------------|---------------------------------|-------------------------------------------------|--------------------------|-----------------------------------------------------------------------------|
| Bluetooth application   | Advertising and scan switch   | Provide Bluetooth Adv Scan      | Function Return, Command Comprehension          | GAP & LL                 | Starts the advertising scan switch feature for Airoha Bluetooth SDK          |
|                         | Initial Adv Scan Application | Function Return, Command        | Function Return, Command Comprehension          | GAP                      | Initiates the application of advertising scan feature                       |
|                         |                               | Comprehension                   |                                                 |                          |                                                                              |
|                         | Updating advertising interval | Provide Bluetooth Proximity     | Function Return, Command Comprehension          | GAP & LL                 | Starts the proximity feature for Airoha Bluetooth SDK                        |
|                         |                               | Recommended Change Adv Interval | Function Return, Command Comprehension          | GAP                      | Changes the advertising interval when time is up                           |
|                         | Security manager              | Provide Bluetooth Security       | Function Return, Command Comprehension          | GAP & SM                 | Starts the security manager feature for Airoha Bluetooth SDK                |
|                         |                               | Manager                         |                                                 |                          |                                                                              |
|                         | Synchronize                   | Provide Bluetooth AirSync       | Function Return, Command Comprehension          | GAP, GATT, & LL          | Starts the air synchronize feature for Airoha Bluetooth SDK                 |
|                         |                               | Initial AirSync Application     | Function Return, Command Comprehension          | GATT                     | Initiates the application of air synchronize feature                        |
|                         | Terminal interaction          | Provide Bluetooth Commander      | Function Return, Command Comprehension          | GAP & LL                 | Starts the commander feature for Airoha Bluetooth SDK                        |

FIGURE 18. Overview of the Bluetooth application kit and the associated statuses.

C. BLOCKLYTOOTH IDE

Hall et al. [56] considered that program optimization requires an interactive environment created by incorporating a compiler into the IDE. The IDE can further provide benefits in program compilation, optimization, and debugging. Therefore, we integrate the VABs designed using block design strategies into an IDE with industrial Airoha Bluetooth SoC and SDK. Fig. 19 shows a screenshot of the BlocklyTooth IDE. The IDE is based on the Andes IDE software with a compiler and toolchain, which supports compilation, optimization, and debugging. Fig. 19-(a) shows the entire development environment and the BlocklyTooth tab page (red box 1) generated by modifying BlocklyDuino. The tab page was launched by pressing the trigger button (red box 15).

15http://www.andestech.com/en/products-solutions/andesight-ide/
D. USE CASES

We now present two use cases of Bluetooth applications designed using the BlocklyTooth IDE. Use case I realizes a controlled LED switching behavior for IoT devices through the block stack, and Bluetooth is used to control the IoT LED switch. Use case II realizes the coupon link acquisition for IoT devices using the block stack, and Bluetooth is used to transfer a coupon service. Both of these use cases employ VABs with the industrial Airoha Bluetooth SDK library, as described in Section V-B. Although the functions of these two use cases are simple, the use of VABs designed by block design strategies can more intuitively present the initialization and internal settings of Bluetooth. Users with a programming background can focus more on the entire architecture of the VABs or convert these blocks into text-based programming through the BlocklyTooth IDE for advanced development. We describe the scenarios for these two cases below.

1) Use Case I: Using Bluetooth to Control an IoT LED Switch

Use case I involves a cell phone Android app that provides Bluetooth detection and connects to an IoT device and then controls the LED switching of this device. Fig. 21-(a) shows the content of the block stack, and Fig. 21-(b) shows the C program converted from this content.

(a) Use Case I: VABs of BlocklyTooth

(b) Use Case I: C Program of BlocklyTooth

FIGURE 21. Use case I: the contents of the VABs and their corresponding source code for an IoT LED switch.

2) Use Case II: Using Bluetooth to Acquire a Coupon Link

Use case II involves an Android app that provides Bluetooth scanning and connects to an IoT device that transfers a coupon service. Fig. 22 shows a screenshot of the app interface. The content of the block stack and the C program can be saved and debugged in the IDE using the saving and debugging functions, respectively. The IDE saving function can be used to save the content of the main file, and the debugging function can be used to set breakpoints. The IDE compilation function can generate binary files. Therefore, Blockly's text mode cannot be modified directly and can provide advanced text-based programming after converting the content of the block to the source code.

FIGURE 22. Use case I: the Android app interface used to control an IoT LED switch.
FIGURE 23. Use case II: the contents of the VABs and their corresponding source code for a coupon service.

(a) Use Case II: VABs of BlocklyTooth (b) Use Case II: C Program of BlocklyTooth

FIGURE 24. Use case II: the Android app interface used to receive a coupon service.

box 1 indicates the initial Bluetooth that is responsible for initiating the Bluetooth function of the IoT device. Red box 2 indicates the handle Bluetooth events VAB that contains the text content comprising a while loop() function and watchdog timer, the purpose of which is to ensure that the IoT device maintains a running state; in particular, the watchdog timer can protect against device crashing and being unable to restart (using countdown and kicking functions). These two VABs correspond to the C program indicated by red boxes 1 and 2 in Fig. 21-(b). Developers designing the IoT device to perform application services continuously using a while loop() function can use these two VABs.

Red box 3 in Fig. 21-(a) indicates how to use trigger services and LED control services, both of which come from the configuration kit of the VABs. These two VABs are responsible for starting the LED function of the IoT device and correspond to the C program indicated by red box 3 in Fig. 21-(b). The remaining blocks not included in the red boxes are related to the hardware functions of IoT devices and generated by splitting their code according to the block design strategies. The strings or program fragments defined inside these blocks are the same processes as the VAB design of the Bluetooth SDK library in this paper.

This content is compiled into a binary file using an IDE and loaded onto the IoT device. Fig. 22 shows a screenshot of the Android app that we developed to control the IoT device. Fig. 22-(a) shows that the Android app detects the Bluetooth connection from the IoT device, whereas Fig. 22-(b) indicates that the LED switching behavior in the IoT device can be controlled by pressing the button (red box).

The VABs related to the Bluetooth SDK library in the above use case provide intuitive stacked content. In contrast, if BlocklyDuino is used to generate naive blocks, the block count will change. This block count comparison can be found in the experiment in Section VII-E1.

2) Use Case II: Using Bluetooth to Transfer a Coupon Service

Use case II involves a cell phone Android app that provides Bluetooth detection and connects to an IoT device and then obtains the HTTP address links for downloadable coupons of this device. Fig. 23-(a) shows the block stack content, and Fig. 23-(b) shows the C program converted from this content.

Red boxes 1 and 2 in Fig. 23-(a) indicate the same two VABs as those employed in use case I and correspond to the C program for red boxes 1 and 2 in Fig. 23-(b). Red box 3 in Fig. 23-(a) indicates how custom self-defined from the service kit of the VABs is used to customize services
related to HTTP address links. This VAB corresponds to the C program indicated by red box 3 in Fig. 23-(b). Red box 4 in Fig. 23-(a) shows how the set attribute length from the configuration kit of the VABs is used to correct the default value of the customizing service. This VAB corresponds to the C program indicated by red box 4 in Fig. 23-(b). The remaining blocks not included in the red boxes are generated in the same way as the blocks used in the previous case I for the hardware functions of IoT devices.

This content is compiled into a binary file using an IDE and loaded onto the IoT device. Fig. 24 shows screenshots of the Android app. Fig. 24-(a) shows the Android app detecting the Bluetooth connection of the IoT device and obtaining the coupon service (red box 1) from the device. Fig. 24-(b) shows the received coupon address links (red box 3) obtained after pressing the button (red box 2) in Fig. 24-(a), and Fig. 24-(c) shows the result.

Similarly, the VABs related to the Bluetooth SDK library in the above use case provide intuitive content. In contrast, if BlocklyDuino is used to generate naive blocks, the block count will change. This block count comparison can also be found in the experiment in Section VII-E1.

VI. METHODOLOGY

This section proposes a method for validating the feasibility of block design strategies, including research questions, evaluation scenarios, and descriptions of the respondents’ backgrounds.

The following research questions were designed, which include usability, practicality, value in learning, and resource usage, for the feasibility evaluation performed in this study; the evaluation results of these questions are further described in Section VII:

- What is the usability of the block design strategies?
- What is the practicality of the block design strategies?
- What is the usability of the BlocklyTooth IDE?
- Is there value in learning using the BlocklyTooth IDE?
- What is the resource usage of the BlocklyTooth IDE?

A. EVALUATION SCENARIO

We designed two 120-minute procedures to evaluate the feasibility, which comprised strategy and IDE parts. Respondents were required to prepare a laptop to complete the evaluation. The strategy part, which involved the block design feasibility of using block design strategies and was evaluated by respondents who are computer science (CS) domain experts, comprised five phases (a to e): preoperation, core concept, tutorial, postoperation, and validation. The respondents first used their laptops to surf the internet to experience Blockly’s stack behavior. We then introduced the principle of the strategies and the operation of BlocklyTooth containing VABs (not integrated with the IDE) and its Bluetooth domain knowledge. The respondents were asked to operate BlocklyTooth and complete a basic example of Bluetooth initialization and connectivity. When the respondents encountered technical problems, they were able to ask for guidance.

The IDE part involved evaluating the feasibility of learning the Bluetooth library development of the BlocklyTooth IDE by respondents with a CS background and those with a non-CS background, and it comprised three phases (a to c): tutorial, operation, and validation. We first provide an overview of the IDE regarding its design concept and the Bluetooth protocol and demonstrate its operation. We then asked the respondents to use the IDE to complete a basic Bluetooth initialization example and execute the associated compilation flow. When the respondents encountered technical problems in this part, they were able to ask for guidance. We additionally provided the same flow for the IDE part for independent respondents, comprising an introduction video, online operations, and answering questions.

The design contents for each phase of the strategy and IDE parts are described below.

1) Strategy Part

a: Preoperation

The purpose of this phase was to enable respondents to experience the imperative programming of block-based tools (e.g., BlocklyDuino). We designed a 20-minute introduction and operation session, which included a basic overview and prompted the respondents to operate the block-stack behavior of the tool based on imperative programming. The arrangement ensured that the respondents had experience in this phase as a baseline before investigating the designed block design strategies.

b: Core Concept

The purpose of this phase was to ensure that the respondents understood the stack concepts and design principles of the strategies. We provided a 40-minute introduction that covered the design principles and the applicable conditions. The strategies form the key part, and introducing them allowed the respondents to understand the differences from the baseline.

c: Tutorial

The purpose of this phase was to help respondents understand the blocks with domain knowledge. We provided a 20-minute tutorial on BlocklyTooth (not integrated with IDE) and its domain knowledge. Since the VABs of BlocklyTooth were generated using block design strategies, we provided the respondents with detailed explanations and a Q&A session on the strategies and VAB cases.

d: Postoperation

The purpose of this phase was to enable respondents to experience differences in design concepts between BlocklyTooth and the baseline. We provided a 30-minute operation session and asked them to use BlocklyTooth to complete the Bluetooth initialization and connection example using
TABLE 1. The four UMUX questions for the strategy part.

| UMUX Questions                                                                 | 1 to 7, Strongly Disagree to Strongly Agree |
|--------------------------------------------------------------------------------|---------------------------------------------|
| Q1 The block design strategies’ capabilities meet my requirements of block design. |                                            |
| Q2 Using the block design strategies is a frustrating experience for block design. |                                            |
| Q3 The block design strategies are easy to use for block design.                  |                                            |
| Q4 I have to spend too much time correcting things that design blocks with the block design strategies. |

TABLE 2. The three customized theme questions for the strategy part.

| Customized Theme Questions                                                                 | 1 to 7, Strongly Disagree to Strongly Agree |
|-------------------------------------------------------------------------------------------|---------------------------------------------|
| Q5 For the block design strategies, which can simplify complex libraries to generate abstract blocks, its design concept has a positive value. |                                            |
| Q6 The block design strategies have practical significance for developers to support IDE tools with abstract block design. |                                            |
| Q7 Feedback. (Users free to text)                                                          |                                            |

TABLE 3. The four UMUX questions for the IDE part.

| UMUX Questions                                                                 | 1 to 7, Strongly Disagree to Strongly Agree |
|--------------------------------------------------------------------------------|---------------------------------------------|
| Q1 The block capabilities of BlocklyTooth IDE meet my requirements of learning Bluetooth library development. |                                            |
| Q2 Using the BlocklyTooth IDE is a frustrating experience for learning Bluetooth library development. |                                            |
| Q3 The BlocklyTooth IDE is easy to use with learning Bluetooth library development. |                                            |
| Q4 I have to spend too much time correcting things that learn Bluetooth library development with the BlocklyTooth IDE. |                                            |

TABLE 4. The five customized theme questions for the IDE part.

| Customized Theme Questions                                                                 | 1 to 7, Strongly Disagree to Strongly Agree |
|-------------------------------------------------------------------------------------------|---------------------------------------------|
| Q3 The BlocklyTooth IDE can be used to implement the Bluetooth library applications even if users with basic programming concepts. |                                            |
| Q6 Compared with the BlocklyTooth IDE, the experience of using a library in the past is less intuitive. |                                            |
| Q7 The BlocklyTooth IDE integrates the Blockly-based Bluetooth library and IDE functions, which helps to comprehend how to use the library. |                                            |
| Q8 It can be useful for industrial applications to incorporate Blockly into industrial IDE. |                                            |
| Q9 Feedback. (Users free to text)                                                          |                                            |

VABs. The respondents could ask for guidance when they encountered technical problems.

e: Validation
A 10-minute validation questionnaire containing seven questions divided into two groups was used to validate the feasibility of the strategy part for the method. The first group was a UMUX questionnaire with four questions as proposed by Finstad [26] and modified by this research to evaluate the usability of an application, as detailed in Table 1. The UMUX was scored on a 7-point Likert scale, which corresponds to an empirical evaluation adjective rating [57] on the system usability scale (SUS) [58]. The results can further evaluate the usability of block design strategies to design blocks. The second group was asked three customized theme questions with the purpose of evaluating the respondents’ perspectives on the practicality of using the strategies in industrial applications. All questions, except for the feedback question, were answered using a 7-point Likert scale, as detailed in Table 2.

2) IDE Part

a: Tutorial
The purpose of this phase was to ensure that the respondents understood the operation of the BlocklyTooth IDE. We provided a 60-minute introduction that included the design reasons for this IDE, operation environment, Bluetooth protocol concepts, and a demonstration of the VAB stack of this IDE.

b: Operation
The purpose of this phase was to enable respondents to experience the functions and operations of the IDE. We provided a 50-minute operation session so that they could operate the IDE and stack VABs. We also asked them to complete a topic on Bluetooth initialization and perform the IDE compilation.
function.

c: Validation
This phase asked the respondents to complete a validation questionnaire, which was 10 minutes long and comprised two sections. The first section involved a nine-question questionnaire divided into two groups. The first group was a four-question UMUX questionnaire as mentioned above, as detailed in Table 3, to further evaluate the usability of using the BlocklyTooth IDE to learn Bluetooth library development. The second group involved five customized theme questions, with the purpose of obtaining the respondents’ perspectives on the value of using this IDE-based learning method. Except for the feedback question, all other questions used a 7-point Likert scale, as detailed in Table 4. The second section involved a resource assessment of the quantity of code and system memory for using VABs.

B. RESPONDENTS
Sixty respondents participated in the design feasibility evaluation of the strategy and IDE parts. The strategy part was evaluated in the CS540400 (2020 Advanced Compiler) course of the Department of Computer Science, National Tsing Hua University, Taiwan. Since the default users are block designers with a programming background and are responsible for developing tools, 20 students majoring in CS (defined as respondents with a programming background) participated in this evaluation. Since the block design strategies integrated concepts of the language structure, these 20 respondents believe that the block design strategies provide good knowledge of system software, library design, and programming skills met the requirements for this evaluation.

The IDE part was evaluated in the CS540300 (2020 Advanced Programming Language) course of the Department of Computer Science, National Tsing Hua University, Taiwan. Because the default users are block users with a programming background but no domain-specific knowledge, 26 students participated in this evaluation: 20 majoring in CS and 6 majoring in industrial engineering or management. In addition, we also used publicity from the included students to further recruit 14 volunteers who majored in engineering or business (respondents with a non-CS background) to complete this evaluation by watching a video and online operations. Because the BlocklyTooth IDE showed the feasibility of block design strategies with the basic concepts of logic, functionality, and graphical user interface (GUI) operation, these 40 respondents who had web design experience and an understanding of programming language also met the requirements for this evaluation.

VII. UMUX RESULTS AND RESOURCE EVALUATIONS
This section reports the UMUX questionnaire results of our design method, addresses the research questions proposed in Section VI, and presents the experimental results on resource usage.

A. UMUX USABILITY OF BLOCK DESIGN STRATEGIES
Table 5 reports the statistical results from the UMUX questionnaire of the usability evaluation of using block design strategies to design blocks, with Q1 and Q3 scored as (score − 1) and Q2 and Q4 scored as (7 − score). The maximum sum score for the four questions is 24. To ensure that our scores corresponded to the range of 0–100 for the adjective rating of the SUS [57], the scores were divided by 24 and multiplied by 100 to obtain the final UMUX scores.

The 20 respondents of CS domain experts with good knowledge of system software, library design, and programming skills had an overall UMUX score of 78.75±14.68 (mean±standard deviation). Choosing a 95% confidence level (i.e., α=0.05) yielded a confidence-interval half-width of 6.43 (range=72.32–85.18). Based on the adjective rating of the SUS, our mean score corresponded to a “good” result. From our analysis of the results, we can understand that the respondents believe that the block design strategies provide the criteria that block design can follow to meet their needs for designing blocks. In addition, the strategies can be used in block design to match the domain knowledge of domain-specific technologies, thereby reducing the time complexity of the design process. Therefore, the usability of the strategy theory for designing blocks is considered satisfactory.
TABLE 7. Results for UMUX questions Q1–Q4 for the IDE part.

| Questions (Likert scale 1 to 7 points) | Non-CS (n = 20) | CS (n = 20) |
|--------------------------------------|----------------|-------------|
|                                      | Mean | S.D. | C.I. | Mean | S.D. | C.I. |
| Q1 The BlocklyTooth IDE’s capabilities meet my requirements of learning Bluetooth library development. | 75.63 | 16.85 | (68.25–83.01) | 68.33 | 17.65 | (60.60–76.06) |
| Q2 Using the BlocklyTooth IDE is a frustrating experience for learning Bluetooth library development. | 5.70 | 1.42 | | 5.50 | 1.40 | |
| Q3 The BlocklyTooth IDE is easy to use with learning Bluetooth library development. | 5.40 | 1.31 | 5.36 | 1.39 |
| Q4 I have to spend too much time correcting things that learn Bluetooth library development with the BlocklyTooth IDE. | 5.70 | 1.31 | 5.50 | 1.40 |

TABLE 8. Results for customized theme questions Q5–Q8 for the IDE part.

| Questions (Likert scale 1 to 7 points) | Non-CS (n = 20) | CS (n = 20) |
|--------------------------------------|----------------|-------------|
|                                      | Mean | S.D. | Mean | S.D. |
| Q5 The BlocklyTooth IDE can be used to implement the Bluetooth library applications even if users with basic programming concepts. | 5.40 | 1.31 | 5.36 | 1.39 |
| Q6 Compared with the BlocklyTooth IDE, the experience of using a library in the past is less intuitive. | 5.50 | 1.40 | 4.85 | 1.57 |
| Q7 The BlocklyTooth IDE integrates the Blockly-based Bluetooth library and IDE functions, which helps to understand how to use the library. | 5.70 | 1.42 | 5.70 | 1.17 |
| Q8 It can be useful for industrial applications to incorporate Blockly into industrial IDE. | 5.85 | 1.09 | 5.70 | 1.03 |

B. PRACTICALITY OF BLOCK DESIGN STRATEGIES

Table 6 reports the results for the customized theme questions (Q5 and Q6) in the value evaluation of the block design strategies for respondents of CS domain experts. For questions Q5 and Q6, the discussion method is different from UMUX, but we evaluated each question independently and calculated its mean. The last feedback question allowed the respondents to freely input their opinions.

Table 6 reveals that the mean scores were quite close. This indicates that the respondents considered that the block design strategies were practical to use, which means that the VABs generated by using the strategy design not only improve the imperative programming of the block (e.g., BlocklyDuino) but also help users understand the overview of domain-specific libraries. The strategies have the significance of being positive and practical in the process of generating blocks. Regarding the last feedback question, the comments of the respondents were positive and provided further suggestions.

- “The use of blocks can help beginners understand easily through abstraction and integration.”
- “It is easy to navigate even for an IoT outsider.”
- “Provide a guide or help for better operation.”
- “Used in a university course, new employee orientation, or employee training.”

C. UMUX USABILITY OF THE BLOCKLYTOOTH IDE

Table 7 reports the results from the UMUX (Q1–Q4) of the usability evaluation of using the BlocklyTooth IDE to learn Bluetooth library development. The UMUX scores were calculated in the same way as described in Section VII-A. The overall UMUX score for the 20 respondents with a non-CS background was 75.63±16.85, with a confidence-interval half-width of 7.38 (range=68.25–83.01). The overall UMUX score for the 20 respondents with a CS background was 68.33±17.65, with a confidence-interval half-width of 7.73 (range=60.60–76.06). Based on the adjective rating of the SUS, the mean score maps to “good” for those with a non-CS background and to “OK” for those with a CS background.

Since the BlocklyTooth IDE, which has block expressions, is a research platform prototype that we defined, the respondents with a non-CS background could meet the requirements of rapidly learning the industrial Airoha Bluetooth SDK library by using the VPL from this prototype. However, they encountered operational errors caused by unfamiliar interfaces when using this professional IDE tool. This barrier could be eliminated within a short time with appropriate guidance. In addition, they also encountered block-stack errors when operating the VABs on the IDE platform, which was caused by their unfamiliarity with the Bluetooth protocol. After we explained this protocol again, they quickly resolved the errors and completed the operation. The results of using this platform for the usability evaluation of learning Bluetooth library development for the respondents with a non-CS background were in line with expectations.

For the respondents with a CS background who are familiar with general programming development tools, there is no barrier to operating this type of IDE platform. Their ability in computer architecture makes it easy to accept the domain knowledge of the Bluetooth protocol architecture. Therefore, for respondents with a CS background, the usability of using this platform to learn Bluetooth library development was acceptable.

D. VALUE IN LEARNING WITH THE BLOCKLYTOOTH IDE

Table 8 reports the results for the customized theme questions (Q5 to Q9) in the value evaluation of the BlocklyTooth
IDE. We independently calculated the averages for each of the questions Q5–Q8, whereas the last feedback question allowed free input. The mean scores for each question were similar for respondents with a non-CS or CS background, indicating that their perspectives were consistent. These results indicate that learning with the BlocklyTooth IDE has value implications in that its intuitive design based on block expressions can help respondents improve their learning for self-training and development capabilities for industrial applications. On the last feedback question, the comments of respondents with a non-CS background were positive and emphasized depth and intuitiveness.

- “This tool is powerful in combination with the IDE.”
- “This is a meaningful topic.”
- “Blocks based on Bluetooth have intuitive features.”

Meanwhile, respondents with a CS background provided further suggestions that focused on software development.

- “Innovative development features.”
- “This can help simplify the AI operation process.”
- “This may help to design a large library in the future.”

### E. RESOURCE USAGE OF THE BLOCKLYTOOTH IDE

This section presents the results of experiments on the quantity of code and system memory to evaluate whether the BlocklyTooth IDE implemented using block design strategies has an impact on resource usage. We first present and discuss the evaluations of the block count and lines of code for the two use cases mentioned in Section V-D. The block count and lines of code were then evaluated based on the contents of the main program of the Bluetooth test samples. Finally, the code usage of flash memory and RAM is evaluated for the system memory of IoT devices using Bluetooth test samples, and the results are used to illustrate the impact of resource usage on the device.

Our experimental environment ran the BlocklyTooth IDE based on the Andes AndeSight IDE (compiler version: nds32le-elf-gcc 4.4.4) as a testbed. Moreover, we used the following seven Bluetooth test samples from the sample projects of the industrial Airoha Bluetooth SDK: TRSPX, Adv_scan_switch, Airsync, BT_LED, BT_init, BT_proximity, and BT_sm. Table 9 provides full descriptions of these test samples, and Table 10 lists the various strategies used in the VAB stack. The baseline was the block design implementation of the naïve blocks from these seven test samples generated by BlocklyDuino (imperative programming of open-source), as described in Section I. We evaluated these samples and the IoT device on an Airoha AB1600 development board with an embedded 32-bit MCU with a clock rate of 16/48 MHz, 512 kB embedded flash memory, and 32 kB of RAM.

Our experimental results indicate that the VABs using the BlocklyTooth IDE based on block expressions can effectively improve the usage of resources, including reducing the quantity of code (both block count and lines of code) and saving system memory (especially RAM). The detailed experimental results are presented below.

#### 1) Findings for the Use Cases

Section V-D introduced two use cases created using the BlocklyTooth IDE. In this section, we report the experimental results for the block count and lines of code for these two use cases. The targets of the experiments have two types of block structures: (1) blocks generated by BlocklyDuino (open-source) and (2) VABs generated by the block design strategies proposed in this paper. The experimental results are used to further discuss the resource usage of the VABs of this IDE.

For use case I, we compare its BlocklyDuino and VABs separately through the block count and lines of code.

| Test Sample | Description |
|-------------|-------------|
| TRSPX       | The test attempts to detect the Bluetooth module and provide data transmission. |
| Adv_scan_switch | The test attempts to examine the hardware button to switch between the Bluetooth for advertising and scanning mode. |
| Airsync     | The test attempts to detect the air synchronization of Bluetooth to establish the data broadcasts. |
| BT_LED      | The test attempts to control the LED module of a remote IoT device via the Bluetooth transmission. |
| BT_init     | The test attempts to examine the Bluetooth initialization process, through setting the random address, public address, and advertising data. |
| BT_proximity| The test attempts to examine by LEDs display mode that the Bluetooth disconnection and the Bluetooth advertising interval modification. |
| BT_sm       | The test attempts to examine the encryption supporting of Bluetooth data. |

| Test Sample | Strategy |
|-------------|----------|
| TRSPX       | Virtual Handling, Function Return, Parameter Overloading, Indicate Repetition. |
| Adv_scan_switch | Function Return, Indicate Repetition. |
| Airsync     | Function Return, Indicate Repetition. |
| BT_LED      | Virtual Handling, Function Return, Parameter Overloading, Indicate Repetition. |
| BT_init     | Function Return. |
| BT_proximity| Virtual Handling, Behavior Module, Function Return, Parameter Overloading, Indicate Repetition. |
| BT_sm       | Function Return. |
Figs. 25-(a) and 25-(b) present the statistical results of this use case in terms of the block count and lines of code, respectively. Fig. 25-(a) shows that the block count decreased by 62.8% for the VABs compared with BlocklyDuino. This indicates that the VABs designed using the block design strategies, including the BLE protocol, have a better effect for this use case.

In Fig. 25-(b), the lines of code were calculated for the main program (i.e., main.c) converted from the block stack. The lines of code decreased by 57.5% for VABs compared to BlocklyDuino. This better effect is due to the code represented by VABs being analyzed and reorganized to enhance the modularity of the code behavior. In other words, the block stack of this use case produces modularized function calls for the main program.

Figs. 26-(a) and 26-(b) present the statistical results for the block count and lines of code, respectively, for use case II. Fig. 26-(a) shows that the block count decreased by only 17.7% for the VABs compared with BlocklyDuino. Because this use case does not have a complicated program structure, which only includes the API call functions and independent API behavior, the block count did not differ significantly between the VABs and BlocklyDuino.

Fig. 26-(b) shows that the lines of code decreased by only 4.7% for the VABs compared with BlocklyDuino. Both the BlocklyDuino and VABs of the main program had a simple structure and large numbers of API call functions, resulting in a smaller difference between the two types of block structures. Together, these results indicate that the effect of VABs is greater for a certain scale of the program structure.

2) Findings for the Main Program

We used the main program of the seven test samples listed in Table 9 to quantify the block counts and lines of code in the evaluations with BlocklyDuino and VABs. Fig. 27 shows that the block count was decreased for VABs compared with BlocklyDuino, by at least 55.6% (for Adv_scan_switch and Airsync) and up to 83.3% (for TRSPX and BT_init); the overall average decreased by 70.2%, indicating the efficacy of VABs designed using our strategies. Because the application of TRASPX and BT_init is almost entirely based on the industrial Airoha Bluetooth SDK library, these two test samples were stacked through the VABs that included the BLE protocol, API analysis, and block-reshaping optimization, showing the most effective results. On the other hand, because the module scale is functionally simple and light for Adv_scan_switch and Airsync, the reduction for the VABs was small for these two test samples.

Fig. 28 shows that the number of lines of code of the main program was decreased for VABs compared with BlocklyDuino by at least 74% (for Airsync) and up to 96.3% (for Adv_scan_switch), and the overall average decreased by 89%. Because these test samples have a high degree of modularization, VABs are highly effective at minimizing the lines of code of their main program. If the program has a simple function and fewer lines of code, it retains the basic program structure even when using VABs. In other words,
the efficacy of using VABs is greater for a certain scale of the program structure.

3) Findings for the System Memory

We also compared the system memory for the seven test samples listed in Table 9, including code memory usage (i.e., the static code size of the native code and the compiled binary file) and RAM usage (i.e., the dynamic code size of the native code and the total amount of memory allocated).

During our design of VABs, all Bluetooth behaviors, IoT-related applications, and modules must be included in the header file. This design avoids errors in compiling the program caused by stacking blocks in the wrong order. We use the compiler provided by the vendor, whose default optimization option is `-Os` (optimized for size). During the compilation process, the compiler analyzes the content of this header file and program and then links the entire static library into the static code by default. As a result, these test samples’ static code size increases, and the dynamic code size also needs an allocation of a large amount of RAM space.

Therefore, our solution optimizes the results of static and dynamic code size by adding two types of GCC compiler options: one is `-ffunction-sections`, `-fdata-sections`. A separate section for each function and variable would be set. The second is link options, `-Wl, -gc-sections`. The `-Wl` was the passed option (as `-gc-sections`) as an option to the linker. The `-gc-sections`, during the link time, would instruct the linker to remove unused sections. Ultimately, these options can remove unused parts of the static SDK libraries, reducing the size of static and dynamic code.

Fig. 29 compares the code memory usage between BlocklyDuino (using the vendor’s default optimization option) and VABs (using our solution’s optimization options). On average, it decreased by 51.8% for the VABs compared with Blockly-Duino. Because the VABs only need to reserve the necessary storage space of the hardware for variable and program content access, they are effective at reducing the utilization rate of the system memory. In other words, the VABs that include this header file do not increase the overhead of the amount of memory required during the execution of applications in hardware and can effectively reduce the memory needed.

VIII. DISCUSSION

The results of the feasibility evaluation indicate that the BlocklyTooth IDE based on block design strategies can have positive effects in vendor library technologies or domain-specific libraries while significantly overcoming the three shortcomings mentioned by Weintrop and Wilensky [59]. The new strengths of this paper are as follows:

- Greater flexibility: this platform supports IDEs and a complete domain-specific library.
- Low complexity: the target is small-scale industrial domain-specific applications, and the strategies support block content based on block expressions.
- Authenticity: in addition to being an introductory learning tool, the platform can also support professional programming environments supported by general commercially available IDEs.

In addition, we discuss the solution of this paper from three aspects: industrial domain expert feedback, use of strategies, and use of the IDE platform.

A. INDUSTRIAL DOMAIN EXPERT FEEDBACK

Because this paper proposed block design strategies and used the industrial Bluetooth SDK library as a case study to design blocks, complete feedback from domain experts is positive for this solution. As mentioned above, we conducted a feasibility evaluation from CS domain experts and obtained satisfactory results. Furthermore, we also brought in a third-party industrial partner to provide actual feedback from industrial domain experts to understand the status of promoting the strategies to industry.
We invited domain experts from an industrial partner, Richtek Technology, with experience in GUI design and the development of fundamental tools. Since we conducted project work with our partner, four employees related to the project participated in this session. The research and development (R&D) experiences of these experts in the company provided direct and useful feedback for our research. The titles of experts who participated in this session were the senior manager (research and development center), manager (product support), deputy manager (new platform development), and senior R&D engineer (new platform development). To make these experts fully understand our research, we reported the principle of block design strategies and demonstrated the entire operation process of BlocklyTooth.

The domain experts expressed positive feedback in this study. The manager believes that the advantage for users is that they can quickly establish a basic structure without text-based programming. Conversely, when the blocks cannot meet special requirements, text-based programming is still required to complete the requirements. The deputy manager held the same view. A well-planned block development environment can bring users a program structure that is intuitive and easy to understand. However, the lack of universal framework blocks may cause applications to be independent, and the blocks cannot be integrated into various tool platforms. They believe that if the blocks can integrate into the infrastructure and establish a complete ecosystem, the user experience will be improved. In addition, the senior R&D engineer raised concerns related to the cost of the block stack, which may be much higher than the cost of users learning text-based programming. This situation must be considered when choosing a suitable domain knowledge application to import the block design using block design strategies. To improve abstraction learning, user experiences should be taken into consideration when using block design strategies.

In terms of application suggestions, the domain experts proposed views that can further discuss the company's internal technique applications through this research based on past development experiences. The senior manager mentioned the application of audio algorithm design. In general, filters can transform audio frequencies and have a modular design for application development. Developers only need to pay attention to how to adjust the parameters of filters, which filters are combined to form a complete algorithm, and determine the order of algorithm content according to their needs. Therefore, this application has an opportunity for developers to generate the block contents of the algorithm according to the block design strategies of this research to achieve their purpose. In addition, the manager proposed an automatic test equipment (ATE) test case evaluation. ATE is a test performed after the electronic components and systems are fabricated and is widely used in electronics manufacturing, semiconductors, and industrial industries. Therefore, this test case evaluation is also suitable for testers to quickly develop their desired test case items or test scenarios through the already built block modules generated by block design strategies. The biggest feature of the abovementioned two applications is that they both have domain-specific libraries, modules, and components, which is in line with the objects of the block design strategies used in this research. However, there is a difference in that ATE can also be operated by users who do not have a programming background, which makes the block expressions generated by the approach of the strategies even more important.

B. USE OF STRATEGIES

Generally, the imperative concept is used to design and generate block programming structures, which ensures that a computer performs a particular function. This text-based programming design method may bring flexibility to the blocks, but it also increases the operational confusion for users with a programming background. Weintrop and Wilensky [60], [61] described that the text expressions in a block are similar to the syntax or keywords of text-based programming language, which hinders the ability of users to achieve a better understanding and operating experience. In contrast, the block design strategies proposed in this study are a design approach for domain-specific libraries. Although this approach may reduce the flexibility of block combinations brought about by block programming based on the imperative concepts, it will require the computer to achieve domain-specific technologies more clearly. The results from the evaluation experiments of the quantity of code (see Sections VII-E1 and VII-E2) reveal that the effect of this approach is greater for domain-specific libraries, modules, and components.

Based on the drawbacks of the block-based programming approach proposed by Weintrop and Wilensky [59], using this strategy approach to design large system software may still result in the block space being much larger than the visual range of the computer screen, which may worsen the experience of the block users. The results of the present feasibility evaluation indicated that the strategy approach should be applied to technologies with a distinct domain-specific protocol structure and a complete and small scope of code content, such as Bluetooth, Wi-Fi, learning models for AI, test automation, robotic applications, and specific industrial IoT applications, for which the usability and practicality are appropriate. Compared with some of the related studies described in Section II [16], [47], [48], the solution is more feasible and can serve as a reference guideline for block designers constructing block-based tools.

C. USE OF THE IDE PLATFORM

Weintrop and Wilensky [59] proposed the following drawback of block-based programming: general users regard block-based tools suitable for only introductory programming. Therefore, we chose a commercially available development environment with complete editing, compiling, and debugging functions as the case integration for this study. The industrial Andes AndeSight IDE used in this study as an example of a commercially available development environment...
is based on the Eclipse IDE and can integrate VAB functions using a plug-in package. For users who want to engage in professional development, BlocklyTooth IDE not only avoids the impression of participating in introductory programming but is also suitable for providing professional development training. However, Meerbaum-Salant et al. [62] and Armoni et al. [63] mentioned that the transition from block-based programming tools to professional programming would affect the learning ability of users. Hence, for block users who do not have a programming background, using this IDE platform will hinder the learning effect and will not help their basic training in programming. In contrast, block users with a programming background using this IDE platform only need to learn the principles and concepts of vendor library technologies or domain-specific libraries through blocks. Even after the basic program framework is generated by the block stack, the development can continue in a text-based editing environment.

Finally, the results of the feasibility evaluation indicated that users with a programming background believed this IDE platform to be usable and valuable for improving their professional knowledge via its block stack. The user feedback revealed that respondents with a CS background suggested integrating more professional areas such as AI into this IDE platform, and respondents with a non-CS background expressed positive views. Compared with the related research reported in Section II [49]–[51], our IDE platform can indeed become a design reference template for block designers integrating block-based tools with IDE. In addition, it can be used as a professional development tool for block users with a programming background, allowing them to choose VABs to understand the needs of domain-specific technologies and practical development.

IX. CONCLUSIONS
Block-based tools have features of abstraction concept and extensibility and are easy to use; however, they are difficult to extend to include the complicated applications of vendor libraries or domain-specific libraries. Therefore, this paper proposes a solution to block design strategies for the block design of domain-specific libraries. The strategies allowed block designers to construct block-based tools with design references, enabling them to redefine or improve the block expression design of complex domain-specific contents. The newly defined blocks based on the block expressions allowed the construction of expression contents and could further improve the programming method of imperative programming of block-based tools. This research used the industrial Airoha Bluetooth SDK library as a case study for the strategies and to generate VABs. The VABs were further integrated with a commercially available development environment to construct the BlocklyTooth IDE. This IDE not only provided blocks but also had a compilation environment, allowing the block users to choose the VABs to comprehend the needs of the domain-specific libraries and practical development.

To examine the feasibility of the strategies, we designed a feasibility evaluation for validating and obtaining positive evaluation results. The block design feasibility evaluation of the strategy part involved respondents of CS domain experts in designing a case study of the industrial Airoha Bluetooth SDK library using block design strategies. The scores on the UMUX questionnaire indicated that the usability evaluation of the block design using the strategies received satisfactory responses. The customized theme survey indicated that the respondents believed that using the strategies in the design process not only improved the block-stack behavior of imperative programming but also helped the block users to comprehend the professional knowledge of domain-specific libraries. The feasibility evaluation of learning Bluetooth library development of the IDE part involved respondents with a programming background but no domain-specific knowledge evaluating the BlocklyTooth IDE produced by the case study. The scores on the UMUX questionnaire indicated that the respondents considered this IDE to have satisfactory usability of learning Bluetooth library development, whereas the customized theme survey revealed that the respondents believed that learning with the BlocklyTooth IDE was of value. This means that the intuitive design of the VABs of this IDE could help improve their self-training and ability to develop industrial applications. The experiment on resource impact revealed that the strategies are effective in improving resource utilization, in which the quantity of code was reduced by 70.2% and 89% in terms of the block count and lines of code, respectively, and the system memory was reduced by 51.8% in terms of the dynamic code size. The results indicate that this research can be a design reference guideline for block-based tools and can effectively abstract the complex Bluetooth library into blocks.

ACKNOWLEDGMENT
The authors would like to thank the team in Software RD Department / Digital Design & Software Division / RD of Richtek Technology, Hsinchu, Taiwan, for their domain expertise, industrial feedback, and suggestions.

REFERENCES
[1] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, “The Scratch programming language and environment,” ACM Trans. Comput. Educ., vol. 10, no. 4, pp. 16:1–16:15, Nov. 2010. [Online]. Available: https://doi.org/10.1145/1866256.1866363
[2] E. Aivaloglou and F. Hermans, “How kids code and how we know: An exploratory study on the Scratch repository,” in Proc. ACM ICER, Melbourne, VIC, Australia, Aug. 2016, pp. 53–61. [Online]. Available: https://doi.org/10.1145/2960310.2960325
[3] D. Weintrop and U. Wilensky, “Comparing block-based and text-based programming in high school computer science classrooms,” ACM Trans. Comput. Educ., vol. 18, no. 1, pp. 3:1–3:25, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3089799
[4] R. V. Roque, “Openblocks: An extendable framework for graphical block programming systems,” Master’s thesis, Dept. of Elect. Eng. and Comput. Sci., M.I.T., Cambridge, MA, USA, 2007. [Online]. Available: https://dspace.mit.edu/handle/1721.1/414550
[5] R. Holwerda and F. Hermans, “A usability analysis of blocks-based programming editors using cognitive dimensions,” in Proc. IEEE VL/HCC, Lisbon, Portugal, Oct. 2018, pp. 217–225. [Online]. Available: https://doi.org/10.1109/VLHCC.2018.8506483

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
T.-L. Chen et al.: Case Study: Design Strategies for Enabling Visual Application Blocks of Bluetooth Library

Programming Environments with Bayesian Hierarchical Models,” in Proc. ACM UMAP, Genoa, Italy, July 2020, pp. 62–70. [Online]. Available: https://doi.org/10.1145/3346361.3349453

[42] M. Verano Merino and T. van der Storm, “Block-based syntax from context-free grammars,” in Proc. ACM SLE, Virtual, USA, Nov. 2020, p. 283–295. [Online]. Available: https://doi.org/10.1145/3426425.3426948

[43] F. Morshed Fahid, X. Tian, A. Emerson, J. B. Wiggins, D. Bouajjaj, A. Smith, E. Wiebe, B. Mott, K. Elizabeth Boyer, and J. Lester, “Progression Trajectory-Based Student Modeling for Novice Block-Based Programming,” in Proc. ACM UMAP, Utrecht, Netherlands, June 2021, p. 189–200. [Online]. Available: https://doi.org/10.1145/3450603.3456833

[44] E. Bagley, J. Shunmugan, and J. Edwards, “Second-Grade Students’ Use of Visual Programming to Learn Multiplication: Leveraging the Concept of Iteration,” in Proc. ACE, Virtual Event, Australia, Feb. 2022, p. 76–84. [Online]. Available: https://doi.org/10.1145/3511861.3511870

[45] D. Mason and K. Dave, “Block-based versus flow-based programming for naive programmers,” in Proc. IEEE B&B, Raleigh, NC, USA, Oct. 2017, pp. 25–28. [Online]. Available: https://doi.org/10.1109/BLOCKS.2017.8120405

[46] R. W. Sebesta, Concepts of programming languages, 12th ed. Pearson, 2019, ISBN: 9780134997186.

[47] A. Ashrov, A. Mannon, G. Weiss, and G. Wiener, “A use-case for behavioral programming: an architecture in JavaScript and Blockly for interactive applications with cross-cutting scenarios,” Sci. Comput. Program., vol. 98, pp. 268–292, Mar. 2015. [Online]. Available: https://doi.org/10.1016/j.scico.2014.01.017

[48] M. Tomleim and K. Grunbæk, “A visual programming approach based on domain ontologies for configuring industrial IoT installations,” in Proc. IoT, Linz, Austria, Oct. 2017, pp. 9–1–9. [Online]. Available: https://doi.org/10.1145/3131542.3131552

[49] M. Bajzek, H. Bort, O. Humphain, L. Mivshek, T. Much, and C. O’Hare, “MUeECS: Embedded blocks for exploring computer science,” in Proc. IEEE B&B, Atlanta, GA, USA, Oct. 2015, pp. 127–132. [Online]. Available: https://doi.org/10.1109/BLOCKS.2015.7369021

[50] J. Blanchard, C. Gardner-McCune, and L. Anthony, “Amphibian: Dual-modality representation in integrated development environments,” in Proc. IEEE B&B, Memphis, TN, USA, Oct. 2019, pp. 83–85. [Online]. Available: https://doi.org/10.1109/BB8857.2019.8941213

[51] A. B. Pratomo and R. S. Perdana, “Arduviz, a visual programming IDE for arduino,” in Proc. ICODSE, Palembang, Indonesia, Nov. 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1145/ICODSE.2017.8285871

[52] A. Balogh and S. Imre, “Simulation and analysis of concurrent BLE link layer state machines running within the same physical device,” in Proc. SOFTCOM, Split, Croatia, Sep. 2017, pp. 1–8. [Online]. Available: https://doi.org/10.23919/SOFTCOM.2017.8115512

[53] J. Campos, S. Colteryanh, and K. Gagneja, “IPV6 transmission over BLE using Raspberry PI 3,” in Proc. ICNC, Maui, HI, USA, Mar. 2018, pp. 200–204. [Online]. Available: https://doi.org/10.1109/ICNC.2018.8390350

[54] G. Margelis, X. Fafoutis, G. Oikonomou, R. Piechocki, T. Tryfonas, and V.-T. Chen, “Physical layer secret-key generation with discreet cosine transform for the Internet of Things,” in Proc. IEEE ICC, Paris, France, May 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC.2017.7997419

[55] M. Spörk, C. A. Boano, M. Zimmerling, and K. Römer, “B4Each: Exploiting the full potential of IPv6 over BLE in constrained embedded IoT devices,” in Proc. ACM SenSys, Delft, Netherlands, Nov. 2017, pp. 2:1–2:14. [Online]. Available: https://doi.org/10.1145/3131672.3131687

[56] M. Hall, D. Padua, and K. Pingali, “Compiler research: The next 50 years,” Commun. ACM, vol. 52, no. 2, pp. 60–67, Feb. 2009. [Online]. Available: https://doi.org/10.1145/1461928.1461946

[57] A. Bangor, P. T. Cortum, and J. T. Miller, “An empirical evaluation of the system usability scale,” Int. J. Hum.-Comput. Interact., vol. 24, no. 6, pp. 574–594, 2008. [Online]. Available: https://doi.org/10.1080/10447310802025776

[58] J. Brooke, “SUS-A quick and dirty usability scale,” Usability Eval. Indus., pp. 189–194, 1996. [Online]. Available: https://doi.org/10.1201/9781498704111

[59] D. Weintrop and U. Wilensky, “To block or not to block, that is the question: Students’ perceptions of blocks-based programming,” in Proc. IDEC, Boston, MA, USA, June 2015, pp. 199–208. [Online]. Available: https://doi.org/10.1109/2771839.2771860

[60] D. Weintrop and U. Wilensky, “The challenges of studying blocks-based programming environments,” in Proc. IEEE B&B, Atlanta, GA, USA, Oct. 2015, pp. 5–7. [Online]. Available: https://doi.org/10.1109/BLOCKS.2015.7368989

[61] D. Weintrop and U. Wilensky, “Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs,” in Proc. IER, Omaha, NE, USA, Aug. 2015, pp. 101–110. [Online]. Available: https://doi.org/10.1109/2787622.2787721

[62] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, “Habits of programming in Scratch,” in Proc. ITiCSE, Darmstadt, Germany, June 2011, p. 168–172. [Online]. Available: https://doi.org/10.1145/1999747.1999796

[63] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, “From Scratch to ‘real’ programming,” ACM Trans. Comput. Educ., vol. 14, no. 4, pp. 25:1–25:15, Feb. 2015. [Online]. Available: https://doi.org/10.1145/2677087

T.-L. Chen (S’01) received the B.B.A. degree in information management from Shih Chien University, Taipei, Taiwan, in 2002 and the M.S. degree in computer science and information engineering form Asia University, Taichung, Taiwan, in 2005. He is currently pursuing the Ph.D. degree in computer science at National Tsing Hua University, Hsinchu, Taiwan. From 2007 to 2009, he was a Engineer, ZyXEL Communication Corp. Hsinchu, Taiwan. His research interest includes visual language, compiler optimization, and parallel computing.

JENQ-KUEN LEE received the B.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2019. He is currently a software engineer. His research interest includes software engineering, system software, and web application development.

I-AN SU received the B.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2019. He is currently a software engineer. His research interest includes software engineering, system software, and web application development.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/