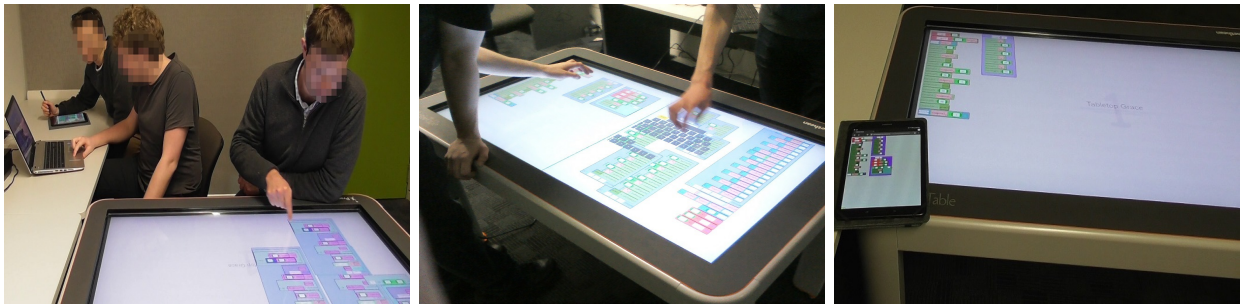
Co-located Collaborative Block-Based Programming

Ben Selwyn-Smith, Craig Anslow, Michael Homer
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email: {selwynbenj,craig,mwh}@ecs.vuw.ac.nz

James R. Wallace
School of Public Health and Health Systems
University of Waterloo
Waterloo, Ontario, Canada
Email: james.wallace@uwaterloo.ca



(a) Multiple users interacting on three devices separately: tablet, laptop, & tabletop.

(b) Multiple users interacting on Tabletop Grace in different workspaces.

(c) Mobile Grace on a tablet and Tabletop Grace with blocks being shared.

Fig. 1: *Multi-Device Grace*: multiple novice programmers simultaneously working together on different devices (laptop, mobile tablet, and digital tabletop) within different independent workspaces to develop a new blocks-based program.

***Abstract*—With the increasing need to teach programming to novices using collaborative methods like pair programming, it is important to understand how different input devices can help support collaborative learning. In this paper we present *Multi-Device Grace*, the first application to explore block-based programming in a cross-device environment consisting of digital tabletops, mobile tablets, and laptops. We conducted a user study ($n = 18$) to explore how cross-device environments can support co-located collaborative block-based programming. The study used Tiled Grace, an existing block-based programming language, and our extensions: Tabletop Grace (designed for tabletops) and Mobile Grace (designed for tablets). Our results show that the majority of participants felt they were able to collaborate quickly and easily, and the cross device interaction would be particularly beneficial in an education setting.**

***Index Terms*—Block-Based Programming, Collaboration, Cross Device Interaction, Tabletop, Mobile**

I. Introduction

Many of today's classrooms teach programming using block-based languages (e.g. Scratch [43]), where programs are composed of visual blocks that can be built, shared, and experimented with easily. Block-based languages are effective in encouraging engagement, storytelling, and exploratory programming [18], [38]. They are also highly collaborative, and instructors largely take advantage of "pair programming" approaches to learning. Advances in these programming languages have not been accompanied by more collaborative hardware. Block-based programming languages are typically taught on PCs (mouse and keyboard) or laptops, but provide little support for building, sharing, and experimenting with blocks in a co-located collaborative environment.

Cross-device techniques (with multiple devices) offer a compelling solution to support co-located collaborative environments [30]. Digital tabletops have been shown to foster collaboration between multiple users [14], who can work on different tasks on different parts of the table simultaneously, or work side by side on the same task together, split out, bring together, and share elements in ad-hoc collaboration. Personal laptops and tablets provide a safe environment where individuals can work on problems before sharing their solutions with others. Block-based languages also lend themselves to touch-based interaction, due to their visual and physical nature.

Our research goal is to explore how cross-device interaction can support students learning to program in a collaborative environment with block-based programming languages, using tabletops, tablets, and laptop computers. We describe *Multi-Device Grace* our block-based programming environment which involves new techniques we designed, *Tabletop Grace* for digital tabletops [45], *Mobile Grace* for mobile tablets, and the original Tiled Grace [25], [26], [27] for desktops (see

This is the author's version of this paper, published in the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), October 14–18 2019, Memphis, TN, USA. The final publication is available in IEEE Xplore via https://doi.org/10.1109/VLHCC.2019.8818895

Figure 1). Tabletop Grace and Mobile Grace extended Tiled Grace to support different interaction modes (touch and pen). We discuss the differences in the design of the tabletop and mobile versions, outline the methods used for cross device interaction between Tabletop Grace, Mobile Grace, and Tiled Grace, and evaluate the cross device functionality via a user study. Figure 1 shows novice users collaborating both across the three device types, and simultaneously on the tabletop.

The remainder of the paper is outlined as follows. Section II addresses related work. Section III presents Multi-Device Grace and addresses various points in the design space we have explored. Section IV describes our user study design and Section V presents the results. Section VI discusses the main contributions of our work and Section VII concludes and summarizes our contributions.

II. Related Work

Most block programming systems are designed for use in traditional keyboard-and-mouse environments and run in a web browser. Users construct programs by dragging blocks with the mouse from a toolbox area onto the workspace. Squeak Etoys [34], Scratch [43], Blockly [8], Pencil Code [4], GP [40], Calico Jigsaw [7], and Tiled Grace [25], [26], [27] based on the textual Grace programming language [5], [6] all have drag-and-drop as a fundamental element of their interaction model, with screen elements designed for traditional Windows, Icons, Menus, and Pointer (WIMP) interfaces.

Drag-and-drop is a problematic paradigm on traditional keyboard-and-mouse systems, both in general [20], [31], [37] and for block programming [26]. On a mobile touch-screen device, dragging appears more seamless than dragging with a mouse, but the additional occlusion of the finger and hand, and the poor resolution of a fingertip (i.e., 'fat fingers'), lead to inaccurate interactions if the touch targets are not large and unambiguous. On a tabletop, however, a drag "across the screen" requires a whole-arm movement or more. For rare interactions, this may be acceptable, but repeated large movements, or requiring finesse and precision are not viable [48].

Some block-based languages have targeted touch-screen devices, primarily tablets, including Hopscotch [28], TouchDevelop [29], Blockly [8], App Inventor [50], Pocket Code [46], and Snap! [21]. TouchDevelop has touch-screen programming as its main interaction model, providing a point-and-tap structured editor with menus and large touchable areas. Hopscotch allows coding on iPad tablets only, and has a similar interaction model to TouchDevelop, but aimed at children. App Inventor deploys only onto touch-screen devices, but users create programs on a mouse and keyboard system. Pocket Code uses a single stack-like collection of screen width blocks that is better suited to the smaller screen space of smartphones.

Most block-based languages support some form of asynchronous project sharing, which takes the form of sharing previously created programs with other users who can then make their own changes. A few languages have added support for synchronous collaboration, including AliCe-VilLagE [1], NetsBlox [10], [11], and an App Inventor plugin [16], [17]. AliCe-VilLagE, an extension of Alice, allows users to remotely contribute to the same project, with built-in remote chat and video conferencing options. NetsBlox, an extension of Snap!, implements the Google Docs collaboration style and additionally provides networking features for created applications. The App Inventor plugin enables synchronous collaboration in a similar manner to NetsBlox.

Tabletop devices and interactive surfaces are well known to enhance collaboration in many domains [2], such as visualization [32], [35] and software development [3], [9], [19], [36], [49]. Isenberg et al. [32] highlighted the use of tabletops in open-ended information foraging tasks, where multiple users could examine, annotate, and share documents with similarity linking to other users, while Kim et al. [35] described similar techniques for mixed physical-virtual collaboration between many users. SourceVis [3] used a tabletop where users could work together, separately, and come back together while working on the same broad task of analysing software through visualizations. Wang et al. [49] and Ghanam et al. [19] explored the potential for digital tabletops to replace typical agile programming meetings, using planning software designed specifically for large tabletop displays. aWall [36] extends these designs to large vertical display walls. CodeSpace [9] adapted the CodeBubbles IDE to touch devices to support software development team meetings, but used a vertical touch screen. These studies show the benefits of tabletops as a single, shared surface around which groups can often collaborate more efficiently and effectively than a traditional desktop PC.

Cross device interaction has been gaining popularity steadily in recent years, where individuals take advantage of both shared (i.e., tabletop, wall displays [30]) and personal devices, such as tablets, mobiles, and devices with larger screens when collaborating. Some noteworthy applications of cross device interaction include: HuddleLamp [42], which allows users to combine tablets for collaboration across a physical table, in an ad-hoc fashion, using a lamp-mounted camera to track devices and hands. CurationSpace [13] allows users to change their interaction method within a shared space through the use of smart watches. SurfaceConstellations [39] is a tool that facilitates multi-monitor and multi-surface environments. Users can adjust the placement of their screens or devices within the software, and then 3D print brackets that will allow the screens or devices to be placed in the user defined positions. VisPorter [15] is a tool designed for collaborative exploration of text documents using multiple devices, and spatially related gestures for collaboration.

To understand the benefits of both shared and personal devices, the research community has performed analyses of group work patterns, called coupling styles [47], during collaborative work. It has been shown that technologies that support a variety of coupling styles, including Discussion, use of Personal Device, and collaboration around a Shared Device, ultimately reduce the effort required to collaborate, and improve collaborative outcomes. In this work, we show that these benefits translate directly to cross-device, co-located collaborative block-based programming.

III. Collaborative Multi-Device Grace

To facilitate cross device block-based programming we have developed *Multi-Device Grace*, which is a block programming application for use on a large tabletop called Tabletop Grace [45] which extends Tiled Grace [25], [26], [27], and Mobile Grace a modification of Tabletop Grace, designed for tablets. Multi-Device Grace is illustrated in Figure 1, with multiple users engaged in editing block-based programs on the same digital tabletop, and across multiple devices. Tabletop Grace has been deployed on a Promethean ActivTable (1920×1080 resolution, 46-inch display, supporting 32 simultaneous touches), and Mobile Grace has been deployed on a Samsung Galaxy Tab S3 (9.7 inch). This paper is based upon some preliminary work on Tabletop Grace that was presented in a non-peer reviewed workshop [45]. We now discuss the design features of each of the Multi-Device Grace components.

A. Cross Device Collaboration

In Multi-Device Grace, users can send tiles from their own device to another device connected to the same device group. In this way users are able to share any of their tiles, from small snippets, to complete programs. Multi-Device Grace does not support any real time collaboration (e.g. Google Docs style shared workspace), but instead allows users to work on their own project and share as needed. We believe this functionality allows for greater flexibility in how Multi-Device Grace can be used, and this design decision was based on our vision of Multi-Device Grace usage in an educational setting. Such a setting would allow students to focus individually on a given task, using a single user device, whilst being able to share snippets for bug fixing, idea demonstration or other purposes with their peers. Students would also be able to work together more closely by using the tabletop version, where multiple students can code in the same area, whilst still allowing individuals to break away and work on smaller tasks on their own single user device as needed. Lastly, the ability to send complete programs to other devices would allow for students to demonstrate their completed program by sending their work to a tabletop or large wall display visible by all involved.

B. Tabletop Grace

Tabletop Grace enables multiple users to work simultaneously, with both in-person and computerised interaction. To avoid contention for the same area, each user can create their own independent, re-orientable, resizeable workspace, depicted in Figure 2. Users may work on different parts of the program in a separate area of the table (for example, one end each) and then combine, or share the programs with each other. Users may also work in the same workspace, as long as they remain on the same edge of the table, as orientation to each of the table edges is only adjustable per workspace. Users can reorient workspaces in any direction to share their current code with any other user at the table. The workspaces are separate Grace modules [23] that can be run and tested individually, but any block, or group of blocks, can be sent to another workspace through the menu that is displayed by tapping that block, including to a workspace that is not currently displayed, and code reuse (or inheritance [33]) is possible between workspaces, by sharing previously defined methods and objects. A complete program can be built in parts and assembled within a shared workspace. Each workspace is loaded through a semi-circle shaped blocks window menu displayed from the table edge. In Tabletop Grace we opted to restrict workspace positioning to the four edges of the tabletop shown in Figure 2, with active workspaces equally sharing all of the available space. This is a trade-off between maximal usage of screen space versus increased flexibility in user positioning. Workspaces have a fixed scale, but are individually scrollable to an infinite area. Full or partial zooming is possible, though scaling items to be feasible touch targets can be difficult.

C. Floating Pie Menus

Menus in Tabletop Grace and Mobile Grace use a different system compared to Tiled Grace, due to using touch input instead. For the Tabletop, the use of statically located UI elements is not feasible due to the average distance a user needs to reach [48], and the possibility of physical contention in a multi-user scenario, while for Mobile Grace, screen space is a premium and much of the screen space would be lost to UI elements using the interface of Tiled Grace. For these reasons we implemented a multi-layer pie menu, seen in Figure 2 (denoted as label i), which can be revealed in-situ via a short press. Like Tiled Grace, the currently available tiles in this tile pie menu change based on the current dialect [24]. Multiple tiles can be created by dragging a block out of the pie menu (rather than tapping), leaving the menu open to be used again. A long press on a block creates a pie menu with deletion, copy, and workspace transfer operations. Each operation applies to an entire nested block structure. A pie menu for global program options, such as running, saving, changing dialects, switching to text view, and overlaying errors [26], is available with a two-finger tap and seen in the second (upright) workspace of Figure 2 (denoted as label ii).

D. Manipulating Blocks

Tiled Grace relies on dragging blocks to join them together or separate them. Frequent or long drags of blocks on tabletops is problematic. To improve the accuracy of tile manipulation using touch, we ensured that every tile is at least as large as a finger-tip. For Multi-Device Grace, we added the ability for users to select multiple tiles at once by using a lasso gesture which works by pressing and dragging on the background of a workspace and lassoing the tiles. When selected in this manner tiles can be grouped together (vertically stacked), or collapsed down to the topmost block of a stack for reduced screen clutter. Subgroups of blocks in a larger block can also be selected.

E. Text Input

Some parts of programs require text to be written by the user, such as defining the name of a variable or method, or the value of a string. In Mobile Grace this is facilitated by using

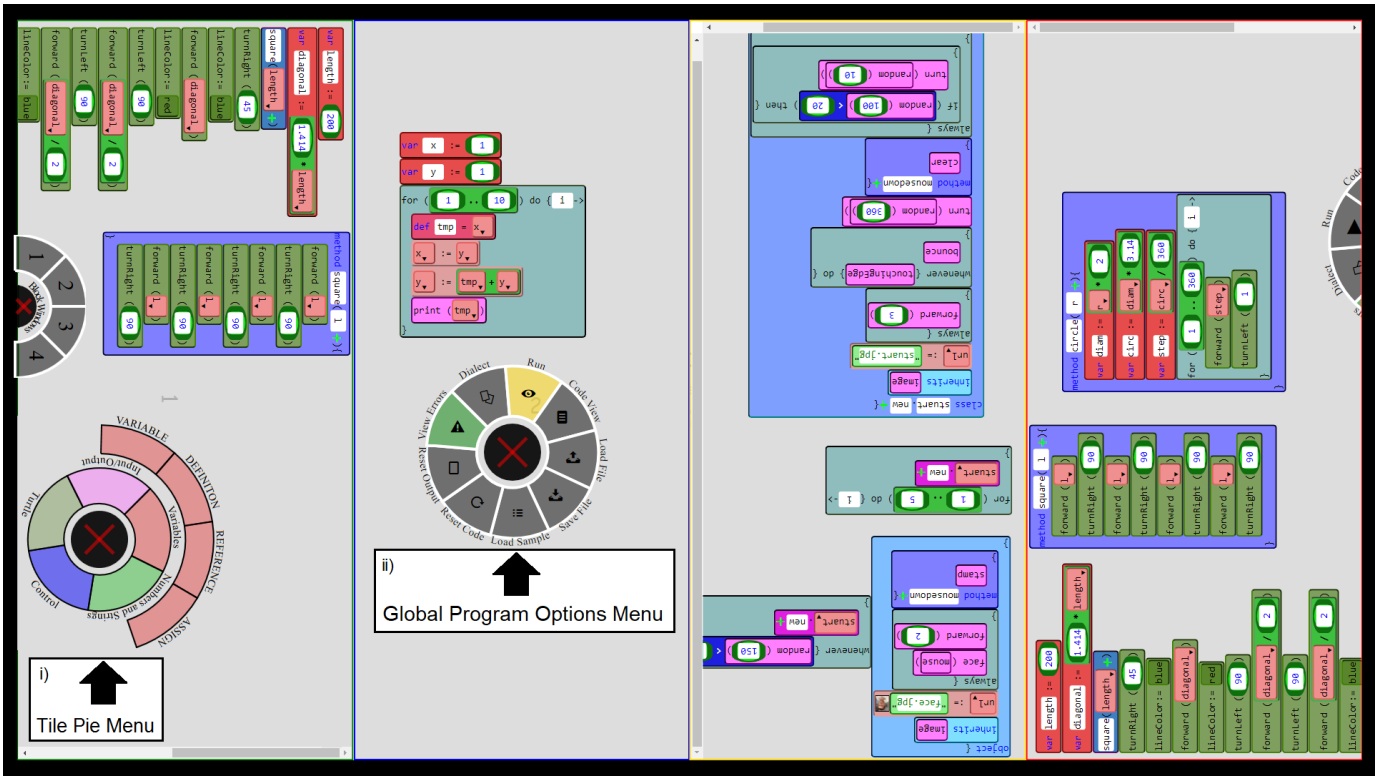


Fig. 2: Multi-Device Grace tile features. Four active workspaces displayed in different orientations showing different Tabletop Grace programs. From left to right: 90° clockwise, 0°, 180°, 90° counter-clockwise. Each workspace is loaded by the semi-circle shaped Blocks Window menu as displayed in the left most workspace. Label i) demonstrates the tile selection pie menu. Label ii) highlights the Global Program Options Menu. Only Tabletop Grace supports multiples workspaces.

the built-in soft keyboard. In Tabletop Grace we implemented our own multiple soft keyboards system, where each field can have a separate keyboard. The operating system soft keyboard can also be used but is only focused on a single field.

F. Mobile Grace

Mobile Grace shares the same interaction method as Tabletop Grace, but the smaller screen size required some changes. Multi finger operations were infeasible, largely due to the increased pointer size when using fingers. Instead we redesigned operations that rely on multi finger input, to require only a single finger. This lead to two-finger presses becoming a press and hold operation, and two-finger drag operations becoming a single finger drag with a UI button for toggling command type. The addition of a stylus improved the interaction accuracy.

G. Cross Device Communication

Cross device interaction is achieved using a Node JavaScript server, which acts as an intermediary for instances of the software. Communication between clients and the server is done using the WebSockets protocol. When clients are connected to the server, they can send updates to other devices, allowing every instance to remain up to date. To send tiles to another device, the desired tiles must be selected, and then the appropriate target device(s) (single or all) is selected through the global menu. Tiles can be sent with or without acceptance to another device, and a visual notification is given regardless. Tiles can be sent to specific active workspaces on the Tabletop. Tiles appear in the same XY position as where they are sent from. To stop communication a device needs to end the connection and stop broadcasting.

IV. User Study

To evaluate the effectiveness of Multi-Device Grace (comprising of Tabletop Grace, Mobile Grace, and Tiled Grace) when used for cross device collaborative programming we conducted a user study. We received human ethics approval.

A. Design

To guide the design of our user study, we started off with a few key aspects that we wanted to consider. We wanted to observe how participants made use of the cross device features of Multi-Device Grace, and to record their subjective opinions. We wanted to measure the usability of Multi-Device Grace on a tabletop and mobile, and compare it with the baseline Tiled Grace on a laptop. To measure cross device collaboration we recorded the coupling styles participants used, where coupling styles refer to the way in which participants work together, based on the coupling styles defined by Tang et al. [47] but applied to a multi-device setting. From these aspects we decided upon three research questions:

RQ1 How effective is each device in a co-located cross device collaborative programming setting?

RQ2 How do participants collaborate when using Multi-Device Grace?

RQ3 How would the cross device functionality be effective in an education scenario?

B. Participants

We recruited a total of 18 participants (M = 16, F = 2) from our university, working in groups of three, for a total of 6 individual sessions. Participants that completed the user study were given a $10 gift voucher honorarium. All participants had experience with textual programming languages, while 72% had experience with block-based programming languages. The textual and block languages that participants had the most prior experience with were mainly Java and Scratch (see Figures 3a and 3b). The minimum age of participants was 21, maximum 28, and mean 22.6.

C. Procedure

User studies took place in a room containing the tabletop next to a desk, with space for a laptop (including external mouse and keyboard) for running Tiled Grace, and a tablet (with stylus) for running Mobile Grace. All three devices ran the software in the Google Chrome Web browser.

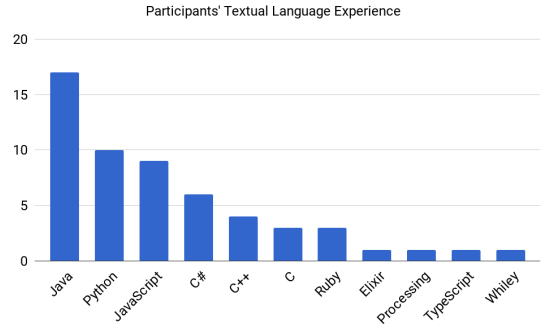
Participants were given information sheets and asked to sign consent forms before participating. Before starting the study, after using each device, and at the end of the study, participants were asked to fill in parts of a questionnaire. Participants were given a short tutorial using a simple sample program, to explain how to use each of the three systems before starting the tasks. We used within subject testing [41], with each participant being assigned a different device for each task and switching device for subsequent tasks. Participants were given 30 minutes maximum to complete all tasks. After completing all the tasks the final sections of the questionnaire were completed by participants.

We used four coupling styles designed to measure loose and tightly coupled scenarios based on those defined by Tang et al. [47]: Discussion (DISC), where participants engage in discussion about the task; Personal Device (PD), where each participant is working on the device they have been assigned to; Same Device 2 participants (SD2), where two participants are engaged on the same device and the third is working on a different device; Same Device 3 participants (SD3), where all participants are working together on the same device. PD, SD2 and SD3 represent a scale for coupling categories, with PD equating to the most loosely coupled, SD2 being more tightly coupled, and SD3 being the most tightly coupled category.

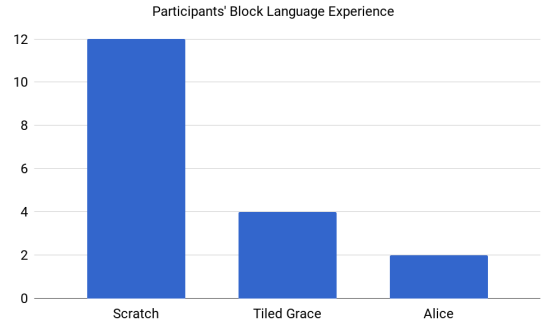
D. Study Tasks

Participants were given three tasks to complete (Appendix D). The tasks relied upon a Grace dialect and sample code that allows for line and circle drawing. Participants were advised to split the workload of each task evenly between themselves. When a task was completed or time limit reached, participants were instructed to send the results to the tabletop, combine the individual parts, and run the final version. Participants were limited to 10 minutes for each of the three tasks. Participants that finished early moved onto the next task. The set of tasks participants were asked to complete were:

1) Correctly draw shapes that match a provided template. Participants had different starting blocks, and had to send tiles to match up methods, and fix intentional bugs.
2) Draw the letters for ”Hello World” using a provided line drawing method. Participants were given a faded template to draw over, and provided with positions and dimensions.
3) Design a simple program, distribute the workload, and create the program.



(a) Textual programming languages.



(b) Block-based programming languages.

Fig. 3: Programming language experience (textual and block-based) as reported by participants.

E. Data Collection

Each participant completed a questionnaire containing questions about their background in terms of relevant knowledge (Appendix A). To measure opinions on system usage we used the System Usability Scale (SUS) [12] for all three systems (Appendix B). The final section of the questionnaire contained several free-form questions, which asked participants about the advantages and disadvantages of sending tiles, their opinions on an example usage scenario, and their thoughts on the usefulness of the cross device functionality (Appendix C). Each user study was recorded using a video camera, and screen capturing software on each device.

V. Results

We now report on the findings from the user study by addressing each of the research questions.

A. RQ1 – How effective is each device in a co-located cross device collaborative programming setting?

Participants responded to SUS questions through a 5-point Likert scale ranging from 1 "Strongly Disagree", to 5 "Strongly Agree", for each of the 10 questions. Box plots of the SUS results for each system are shown in Figure 4. Results show that participants favoured the tablet system over the other two devices. Some participants mentioned that they believed the laptop system to be problematic due to the differences between the user interfaces, namely pie menus versus statically located menus. Despite this, the SUS scores for all three systems were very similar, indicating that this was not a significant problem. Participants also reported that the ability to combine and display programs on the larger tabletop screen was particularly valuable, giving the tabletop a clear role.

Responses to the SUS questions highlight a number of points. The laptop system scored highest in terms of likeability, ease of use, and design quality, but participants' responses indicate that the laptop system was also the most inconsistent, hardest to learn, and had the most usability hurdles. The tabletop and tablet systems scored quite similarly across all questions. Two clear differences between these can be seen, firstly in terms of ease of learning, where the tabletop scored highest out of the three platforms, and secondly assistance requirement, where the tablet was reported as being the system that participants would be happiest using without assistance.

Overall, the responses for the SUS questions can be combined into a single value that represents the overall usability of the system as perceived by the user study participants. For our user study the overall scores were 64.3, 64.7 and 65.3 for the tabletop, laptop, and mobile devices respectively. SUS research shows that a score of 68 is considered average for usability [44], indicating that participants felt each system was slightly below average in terms of usability. That all three systems received fairly similar scores indicates that the user interface we designed and implemented for the tabletop and tablet versions is as usable as the user interface of Tiled Grace, with Mobile Grace being the preferred system by a small margin over the others.

The first of the questions in the freeform questions of the questionnaire prompted participants to rank the three systems in terms of effectiveness. Responses for this question were tallied to create an overall score for each system, where ranking a system in first place counted as 2 points, second place 1 point, and third place 0 points. The responses to this question show that overall, participants preferred working on the tablet system with 21 points, then the tabletop with 18 points, and lastly the laptop with 15 points.

Participants were also asked to specify advantages and disadvantages of the ability to send tiles between systems. Responses to this question that were reported by at least two participants can be seen in Figures 5a and 5b. The most

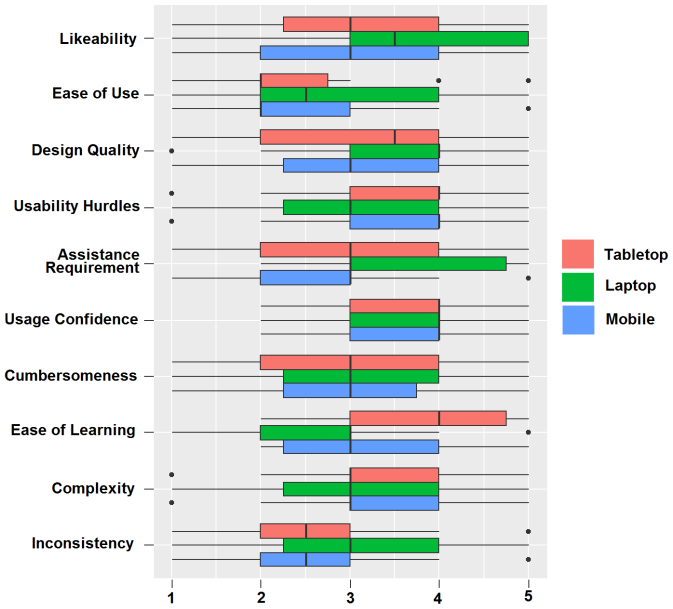


Fig. 4: System Usability Scale data using 5 point Likert scale. Red: Tabletop, Green: Laptop, Blue: Mobile.

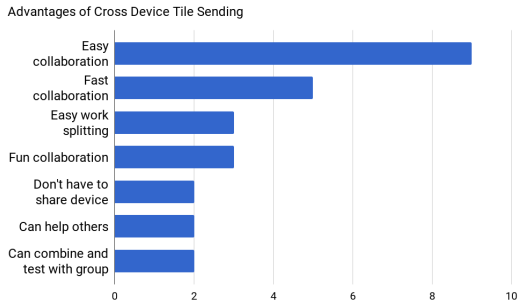
common advantages voiced by the participants were that cross device tile sending makes collaboration easy, fast and fun, as well as allowing the workload to be split easily. The main reported disadvantages were that there was no limit on tile sending, it was hard to remember the names of devices, and that sent tiles can't be updated without re-sending them.

The last question in the freeform questions asked participants if they thought the cross device functionality provided by Multi-Device Graces was useful for collaborative programming. 17 out of 18 (94%) participants answered affirmative for this question. The most commonly reported reasons in order of number of responses were: it was easy to combine parts of a program into a final version and view the results; users of such a system can use whichever device suits them best, based on preference or current activity; work can easily be split up into smaller parts and distributed among a group; and small snippets of code can easily be shared.

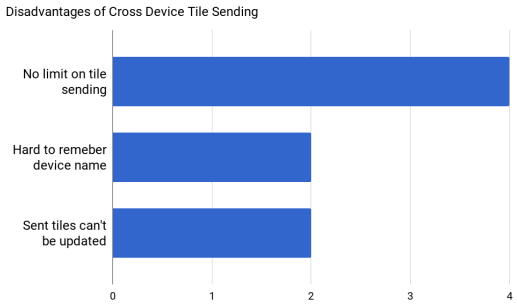
B. RQ2 – How do participants collaborate when using Multi-Device Grace?

During our user study we recorded participants making use of the three devices to work together, split apart, and reconvene as required. We coded the amount of time each group spent in the different coupling styles so we could compare and contrast the different styles. Participants spent the largest portion of their time in the personal device coupling style, the most loosely based coupling style measured. Participants were able to switch to more tightly coupled styles when needed for seeking and providing assistance, discussing progress, and combining individual work into a completed program.

Due to the nature of the tasks provided to participants, all groups spent some time at the start of each task in the discussion coupling style. This involved dividing up the



(a) Advantages of cross device tile sending.



(b) Disadvantages of cross device tile sending.

Fig. 5: Participants' responses from the final questionnaire: opinions on the advantages and disadvantages of cross device tile sending, reported from at least 2 participants.

task into equal parts and deciding who would do which part. The amount of time spent in this coupling style varied between groups and between tasks with task 3 requiring more discussion on average, and task 2 the least amount. Groups then moved onto the Personal Device (PD) coupling style to start work on their part of the group task. During this time groups would often temporarily enter into the Shared Device 2 or 3 (SD2, SD3) coupling style, before reverting back to PD. Reasons for entering into SD2 or SD3 often included: additional task deliberation, and progress reports. Participants that were unsure how to proceed or conversely had finished their part early also made use of SD2 or SD3 in a support requesting or support providing role. A few groups opted to perform more rigorous planning using paper or whiteboards, which was common for task 3, where participants had to implement a program of their own choosing.

Overall, participants spent the largest amount of time (64%) in the PD coupling style, which involved completing their portion of the assigned task. The second most commonly used style was discussion (15%), and then SD2 (13%). The least commonly used coupling style was SD3 (7%). These results are shown in Table I. Breaking down the results by task shows that task 2 required the least amount of communication and also involved no SD3 coupling. This is likely due to the task being easier to split up and groups running out of time to test the final program on the tabletop using the SD3 coupling style. While task 3 involved more discussion as groups had to agree

TABLE I: The amount of time participants spent in each coupling style, grouped by task. Coupling Styles: Discussion, Personal Device, Same Device 2, Same Device 3.

	D	PD	SD2	SD3
Task 1	13.42%	59.14%	13.92%	13.53%
Task 2	6.94%	87.17%	5.89%	0.00%
Task 3	24.94%	46.94%	20.58%	7.53%
Total	15.10%	64.42%	13.46%	7.02%

upon a program to implement before dividing up the task, it also involved considerably less SD3 time than task 1.

C. RQ3 – Would the cross device functionality be useful in an education scenario?

The third question of the final section of the questionnaire, asked participants to provide their opinion of the potential for Multi-Device Grace to be used in a classroom environment for teaching purposes. 16 out of 18 (89%) participants responded that they believe this application would be beneficial for a classroom environment, with the most common reasons being: it made for fun and enjoyable group learning; it provided a tangible, visual, hands on experience; it provides the ability to distribute lessons or examples to other devices; and the application was easy to learn. These reasons favour an educational setting because the potential of Multi-Device Grace provides quick and easy collaboration methods which can help facilitate a fun group learning scenario.

VI. Discussion

We now discuss the more important aspects from our results, most notably cross device collaboration and designing block-based environments for touch surfaces.

A. Block-Based Cross Device Collaboration

The most important finding from our research in terms of block-based collaboration, was that the cross device functionality provided by our software for Tabletop Grace, Mobile Grace, and our modified version of the original Tiled Grace, was considered to be beneficial for collaboration by 94% of the participants. Participants reported that the ability to break apart programming tasks into smaller sub tasks that can be distributed amongst a group quickly and easily, worked on separately in a user's preferred device, and finally recombined into a finished program on a single device, was very useful for collaboration. 89% of participants responded affirmatively about the potential use of Multi-Device Grace in classrooms.

The cross-device functionality also supported a variety of coupling styles, though we found that most of our participants followed a set pattern when completing tasks, where they discussed the problem, shifted to personal device usage with sporadic same device 2 (SD2) phases, and finally completed the task together via same device 3 (SD3). These patterns are consistent with the literature surrounding cross-device collaboration (e.g., Homaeian et. al [22]), and show support for the flexibility required to effectively and efficiently solve complex tasks as a group. For example, participants chose

to make heavier use of SD2 when assisting group members that were stuck. The cross device functionality, affords users a large amount of flexibility in how they approach problems, and support for both individual and collaborative sub-tasks.

Houben et al. [30] describe seven opportunities and challenges based on the proceedings of the Cross-Surface workshops. We address some of these opportunities and challenges. The first of these is *enabling easy configuration of device ecologies*. In terms of discovering other devices, Multi-Device Grace allows all devices to easily connect to a central server and communicate with other connected devices, at the press of a button. While disabled for our user study, Multi-Device Grace can allow users to join user defined groups, preventing communication to non grouped devices. The second point raised is *designing for scale and interoperability*. Each system in the Multi-Device Grace group, runs in a web browser, requiring no extra software installation as long as an up-to-date browser is used. This allows users to share tiles between any set of devices that support a modern web browser. Furthermore, Multi-Device Grace provides software designed for three common device types: Mouse and Keyboard computers, large touch screen devices, and mobile tablet devices. Another issue raised is *addressing cross-device interaction challenges*. The user interface for each system in Multi-Device Grace, differs to match the device it is designed for, however the process of sending tiles between devices, follows the same process on each system: select tiles, go to connection menu, and send to target device(s). The on screen appearance of the connection menu is also kept the same across each system, which helps to keep the cross device interaction process as simple as possible.

B. Designing Touch Enabled Blocks-Based Environments

Overall, based on the responses received for the SUS questions, participants considered the laptop, tablet, and tabletop systems to be almost equivalent in terms of usability. We found this result to be particularly interesting, as all our participants had extensive mouse and keyboard experience, suggesting that the more familiar WIMP style interface would receive a noticeably higher score than the less familiar tablet and tabletop interfaces. In addition, the SUS scores and participant reported device rankings do not match up. While the tablet system was ranked highest in effectiveness and SUS, the tabletop and laptop systems were placed differently in both. We believe that the tabletop provides a novel interface that allows users to have a more enjoyable collaborative programming experience, despite a marginally perceived weaker user interface.

One of the key features of Tiled Grace is the ability to transition from a block-based view to a direct textual equivalent. On the touch surface devices this feature was less useful, due to keyboards being harder to use, with physical keyboards being awkward to position and potentially disrupting other users, and soft keyboards suffering due to the issues with text input and focus. Generally, this points to taking a simpler approach better suited for novices, with more focus on the blocks themselves than the textual representation. The tile pie menu noticeably reduced time spent dragging blocks around the screen, as the menu could be brought up in close proximity to where participants wanted new blocks. During the user study, we noticed a few participants trying to call up a tile menu directly in the hole they wished to place new blocks, which we hadn't considered implementing. This could be a more efficient strategy for creating and adding new blocks.

Limitations. There were some limitations with our study. We conducted our study in a controlled lab and with a convenience sample within the same geographical region. As we conducted a qualitative study with 18 participants we were not concerned with errors made or time taken to perform the tasks. We did not compare with other textual based languages or other block based languages as we see this as future work.

Future Work. Based on feedback we believe that further testing the potential of Multi-Device Grace in the wild in an educational classroom setting would be of value. With introductory programming being introduced earlier and earlier in schools, performing a longitudinal study of the effectiveness of such a learning method, could help guide design decisions and ultimately increase interest in programming for young people. We would also like to do comparative studies to find out how effective collaborative blocks based programming is with multiple devices compared with solely a textual based language, blocks on a single device, and remote collaboration.

VII. Conclusion

As teaching programming to novices becomes more prevalent and widespread, which is often happening in pairs and groups of people, there is a need to support different methods for collaborative programming. In this paper we presented *Multi-Device Grace*, our vision for co-located cross device block-based programming, using a mouse and keyboard computer, mobile tablet, a digital tabletop, and modifications to an existing block-based programming language Tiled Grace [25], [26], [27]. Multi-Device Grace allows multiple users to work together in their own workspace, remotely via cross device communication, and on the device that is most suitable for their current situation.

We evaluated Multi-Device Grace through a user study (n=18, within subjects testing). Results showed that Multi-Device Grace with our block-based interface modifications for a digital tabletop and mobile tablet were equivalently usable as the traditional mouse and keyboard interaction method. Nearly all participants (94%) found the cross device functionality of Multi-Device Grace, to be valuable for collaboration, due to the ability to split tasks apart, work on them separately across multiple devices, and recombine the final program easily. Of the three systems we tested, participants reported that they preferred the tablet version, in terms of effectiveness through a system ranking and usability through the System Usability Scale questions. Our aim is to see introductory programming courses in high schools adopt new styles of interfaces to help aid collaborative programming and we believe Multi-Device Grace is an effective design for this activity.

VIII. Acknowledgements

We thank all the participants who volunteered for the user study. We thank the members from the Grace programming language community for the valuable discussions regarding the design of Multi-Device Grace and user study including James Noble, Andrew Black, Kim Bruce, and Tim Jones.

References

[1] Ahmad Al-Jarrah and Enrico Pontelli. "AliCe-ViLlagE" Alice as a Collaborative Virtual Learning Environment. In *Frontiers in Education Conference (FIE), 2014*, pages 1–9. IEEE, 2014.

[2] Craig Anslow, Pedro Campos, and Joaquim Jorge, editors. *Collaboration Meets Interactive Spaces*. Springer, 2016.

[3] Craig Anslow, Stuart Marshall, James Noble, and Robert Biddle. Sourcevis: Collaborative software visualization for co-located environments. In *Proc. of International Working Conference on Software Visualization (VISSOFT)*, pages 1–10. IEEE, 2013.

[4] David Bau, D. Anthony Bau, Mathew Dawson, and C. Sydney Pickens. Pencil Code: Block code for a text world. In *Proc. of International Conference on Interaction Design and Children (IDC)*, pages 445–448. ACM, 2015.

[5] Andrew P. Black, Kim B. Bruce, Michael Homer, and James Noble. Grace: The absence of (inessential) difficulty. In *Proc. of International Symposium on New Ideas in Programming and Reflections on Software (Onward!)*, pages 85–98. ACM, 2012.

[6] Andrew P. Black, Kim B. Bruce, Michael Homer, James Noble, Amy Ruskin, and Richard Yannow. Seeking Grace: a new object-oriented language for novices. In *Proc. of SIGCSE*, pages 129–134. ACM, 2013.

[7] Douglas Blank, Jennifer S. Kay, James B. Marshall, Keith O'Hara, and Mark Russo. Calico: A multi-programming-language, multi-context framework designed for computer science education. In *Proc. of SIGCSE*, pages 63–68. ACM, 2012.

[8] Blockly Project. Blockly web site. https://code.google.com/p/blockly/, 2011.

[9] Andrew Bragdon, Rob DeLine, Ken Hinckley, and Meredith Ringel Morris. Code space: Touch + air gesture hybrid interactions for supporting developer meetings. In *Proc. of Interactive Tabletops and Surfaces (ITS)*, pages 212–221. ACM, 2011.

[10] Brian Broll and Akos Ledeczi. Distributed Programming with NetsBlox is a Snap! In *Proc. of SIGCSE*, pages 640–640. ACM, 2017.

[11] Brian Broll, Akos Lédeczi, Peter Volgyesi, Janos Sallai, Miklos Maroti, Alexia Carrillo, Stephanie L. Weeden-Wright, Chris Vanags, Joshua D. Swartz, and Melvin Lu. A visual programming environment for learning distributed programming. In *Proc. of SIGCSE*, pages 81–86. ACM, 2017.

[12] John Brooke et al. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[13] Frederik Brudy, Steven Houben, Nicolai Marquardt, and Yvonne Rogers. Curationspace: Cross-device content curation using instrumental interaction. In *Proc. of International Conference on Interactive Surfaces and Spaces (ISS)*, pages 159–168. ACM, 2016.

[14] Stéphanie Buisine, Guillaume Besacier, Améziane Aoussat, and Frédéric Vernier. How do interactive tabletop systems influence collaboration? *Computers in human behavior*, 28(1):49–59, 2012.

[15] Haeyong Chung, Chris North, Jessica Zeitz Self, Sharon Chu, and Francis Quek. Visporter: facilitating information sharing for collaborative sensemaking on multiple displays. *Personal and Ubiquitous Computing*, 18(5):1169–1186, 2014.

[16] Xinyue Deng. Group collaboration with app inventor. Master's thesis, Massachusetts Institute of Technology, 2017.

[17] Xinyue Deng and Evan W Patton. Enabling multi-user computational thinking with collaborative blocks programming in MIT app inventor. *Siu-cheung KONG The Education University of Hong Kong, Hong Kong*, page 168, 2017.

[18] Diana Franklin, Phillip Conrad, Bryce Boe, Katy Nilsen, Charlotte Hill, Michelle Len, Greg Dreschler, Gerardo Aldana, Paulo Almeida-Tanaka, Brynn Kiefer, et al. Assessment of computer science learning in a scratch-based outreach program. In *Proc. of SIGCSE*, pages 371–376. ACM, 2013.

[19] Yaser Ghanam, Xin Wang, and Frank Maurer. Utilizing digital tabletops in collocated agile planning meetings. In *Proc. of AGILE*, pages 51–62. IEEE, 2008.

[20] Douglas J. Gillan, Kritina Holden, Susan Adam, Marianne Rudisill, and Laura Magee. How does Fitts' law fit pointing and dragging? In *Proc. of CHI*, pages 227–234. ACM, 1990.

[21] Brian Harvey and Jens Mönig. Bringing "no ceiling" to Scratch: Can one language serve kids and computer scientists? In *Proc. of Constructionism*, 2010.

[22] Leila Homaeian, Nippun Goyal, James R. Wallace, and Stacey D. Scott. Group vs individual: Impact of touch and tilt cross-device interactions on mixed-focus collaboration. In *Proc. of CHI*, pages 73:1–73:13. ACM, 2018.

[23] Michael Homer, Kim B. Bruce, James Noble, and Andrew P. Black. Modules as gradually-typed objects. In *Proc. of Workshop on Dynamic Languages (DYLA)*. ACM, 2013.

[24] Michael Homer, Timothy Jones, James Noble, Kim B. Bruce, and Andrew P. Black. Graceful dialects. In *Proc. of European Conference on Object-Oriented Programming (ECOOP)*, pages 131–156. Springer, 2014.

[25] Michael Homer and James Noble. A tile-based editor for a textual programming language. In *Proc. of International Working Conference on Software Visualization (VISSOFT)*. IEEE, 2013.

[26] Michael Homer and James Noble. Combining tiled and textual views of code. In *Proc. of International Working Conference on Software Visualization (VISSOFT)*. IEEE, 2014.

[27] Michael Homer and James Noble. Lessons in combining block-based and textual programming. *Journal of Visual Languages and Sentient Systems*, 3, 2017.

[28] Hopscotch Technologies, Inc. Hopscotch - learn to code through creative play. https://www.gethopscotch.com/, 2011.

[29] Nigel Horspool, Judith Bishop, Arjmand Samuel, Nikolai Tillmann, Michał Moskal, Jonathan de Halleux, and Manuel Fähndrich. *TouchDevelop: Programming on the Go*. Microsoft Research, 2013.

[30] Steven Houben, Nicolai Marquardt, Jo Vermeulen, Clemens Klokmose, Johannes Schöning, Harald Reiterer, and Christian Holz. Opportunities and challenges for cross-device interactions in the wild. *Interactions*, 24(5):58–63, 2017.

[31] Kori Inkpen. Drag-and-drop versus point-and-click mouse interaction styles for children. *Transactions on Computer-Human Interaction*, 8(1):1–33, 2001.

[32] Petra Isenberg, Danyel Fisher, Sharoda A Paul, Meredith Ringel Morris, Kori Inkpen, and Mary Czerwinski. Co-located collaborative visual analytics around a tabletop display. *Transactions on Visualization and Computer Graphics*, 18(5):689–702, 2012.

[33] Timothy Jones, Michael Homer, James Noble, and Kim Bruce. Object inheritance without classes. In *Proc. of European Conference on Object-Oriented Programming (ECOOP)*, 2016.

[34] Alan Kay. Squeak Etoys authoring & media. Research note, Viewpoints Research Institute, 2005.

[35] KyungTae Kim, Waqas Javed, Cary Williams, Niklas Elmqvist, and Pourang Irani. Hugin: A framework for awareness and coordination in mixed-presence collaborative information visualization. In *Proc. of Interactive Tabletops and Surfaces (ITS)*, pages 231–240. ACM, 2010.

[36] Martin Kropp, Craig Anslow, and Magdalena Mateescu. Enhancing agile team collaboration through the use of large digital multi-touch cardwalls. In *Proc. of International Conference on Agile Software Development (XP)*. ACM, 2017.

[37] Scott MacKenzie, Abigail Sellen, and William Buxton. A comparison of input devices in element pointing and dragging tasks. In *Proc. of CHI*, pages 161–166. ACM, 1991.

[38] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. Programming by choice: urban youth learning programming with scratch. In *Proc. of SIGCSE*, pages 367–371. ACM, 2008.

[39] Nicolai Marquardt, Frederik Brudy, Can Liu, Ben Bengler, and Christian Holz. Surfaceconstellations: A modular hardware platform for ad-hoc reconfigurable cross-device workspaces. In *Proc. of CHI*, page 354. ACM, 2018.

[40] Jens Mönig, Yoshiki Ohshima, and John Maloney. Blocks at your fingertips: Blurring the line between blocks and text in GP. In *Proc. of Workshop on Lessons and Directions for First Programming Environments (Blocks and Beyond)*, pages 51–53. IEEE, 2015.

[41] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[42] Roman Rädle, Hans-Christian Jetter, Nicolai Marquardt, Harald Reiterer, and Yvonne Rogers. Huddlelamp: Spatially-aware mobile displays for ad-hoc around-the-table collaboration. In *Proc. of International Conference on Interactive Tabletops and Surfaces (ITS)*, pages 45–54. ACM, 2014.

[43] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, November 2009.

[44] J. Sauro. *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices*. CreateSpace Independent Publishing Platform, 2011.

[45] Ben Selwyn-Smith, Michael Homer, and Craig Anslow. Towards collaborative block-based programming on digital tabletops. In *Proc. of Workshop on Lessons and Directions for First Programming Environments (Blocks and Beyond)*. IEEE, 2017.

[46] Wolfgang Slany. Tinkering with Pocket Code, a Scratch-like programming app for your smartphone. *Proc. of Constructionism*, 2014.

[47] Anthony Tang, Melanie Tory, Barry Po, Petra Neumann, and Sheelagh Carpendale. Collaborative coupling over tabletop displays. In *Proc. of CHI*, pages 1181–1190. ACM, 2006.

[48] Aaron Toney and Bruce H Thomas. Considering reach in tangible and table top design. In *Proc. of International Workshop on Horizontal Interactive Human-Computer Systems (Tabletop)*, pages 2–pp. IEEE, 2006.

[49] Xin Wang, Yaser Ghanam, and Frank Maurer. From desktop to tabletop: Migrating the user interface of agileplanner. In *Proc. of International Working Conference on Human-Centered Software Engineering (HCSE)*, pages 263–270. Springer, 2008.

[50] David Wolber, Hal Abelson, Ellen Spertus, and Liz Looney. *App Inventor*. O'Reilly Media, Inc., 2011.

APPENDIX

A. Background Information

1) What is your age?
2) What is your height?
3) How much experience do you have with computers using mouse and keyboard?
4) What is the average numbers of hours per week you spend using mouse and keyboard devices?
5) How much experience do you have with touch based devices?
6) What is the average number of hours per week you spend using touch based devices?
7) How much programming experience do you have?
8) Please list the textual programming languages you use frequently, or have used extensively.
9) How much experience, if any, do you have with block-based programming languages?
10) If you have experience with block-based programming languages, please tick all languages you have used. (Scratch, Blockly, Pocket Code, Pencil Code, Alice, Tiled Grace, Other)

B. System Usability Survey

Answers were given in a 5-point Likert scale, ranging from Strongly Disagree (1) to Strongly Agree (5)

1) I would like to use this interaction method with block based languages frequently.
2) This interaction method was easy to use.
3) The interaction method and user interface were well integrated.
4) I needed to learn a lot of things before I could start completing the tasks.
5) I would be comfortable using this interaction method without assistance in the future.
6) I felt very confident using this interaction method.
7) This interaction method was very cumbersome to use.
8) Most people would learn to use this interaction method easily and quickly.
9) I found this interaction method to be unnecessarily complex.
10) There was too much inconsistency with this interaction method.

C. Freeform Questions

1) Please rank the three devices from best to worst below (e.g. 1 = the device you liked using the most.).
2) In your opinion what are some of the advantages and disadvantages of being able to send tiles to other devices, using the current system?
3) Do you think the current system would be effective for usage in an education setting such as a classroom? Please briefly explain your answer.
4) Overall, do you think the cross device functionality was effective for collaboratively completing programming tasks? Please briefly explain your answer.
5) If you have any other comments about (or suggestions for potential uses of, or possible improvements for) any aspect of the system, please write them below.

D. User Study Tasks

Note: The first two tasks have some tiles located offscreen that are required for the program to work. Please do not modify, delete, or transfer these tiles.

Task 1 - Correctly draw shapes that match a provided template:

1) Match up method and method call (i.e. draw_circles (...) + method draw_circles), by sending the method or method call to the correct person.
2) Fix the errors within the methods.
3) Pass the fixed methods and method calls to the tabletop for demonstrating correctness.

Task 2 - Draw the letters for "Hello World":

1) Use the line.draw class to draw letters out of lines. The letter W is already present as an example.
2) Split up letter drawing between the team, so that each member has roughly an equivalent workload.
3) Send the final letters to the tabletop and run them together.

Task 3 - Design a simple program:

1) Decide upon a program to implement. It is recommended to stick to something fairly simple as you only have limited time to complete this in.
2) Split up workload between the team as desired.
3) Demonstrate the final result on the tabletop.