DressCode: Tools and Activities to Engage Youth in Algorithmic Craft

1st Author Name
Affiliation
Address
e-mail address
Optional phone number

2nd Author Name
Affiliation
Address
e-mail address
Optional phone number

ABSTRACT
Algorithmic craft is the combination of computational design and hand craft for the production of useful, beautiful and personally meaningful physical artifacts. We believe that this combination poses a unique opportunity to engage young people in forms of technological creation in ways that are relevant, pleasurable and intellectually engaging. In this paper we describe our development of DressCode, a software tool that combines programing *and* graphic drawing, enabling the creation of designs that are compatible with handcraft. We evaluated DressCode through a study in which young people created computational designs for screen printing. Our analysis of this study demonstrates useful design principles for combining programing with graphic design and demonstrates the diversity of practices that are possible by engaging people in the creation of objects that are imagined by the mind, designed through computation, and shaped by hand.

Author Keywords
Guides; instructions; author's kit; conference publications; keywords should be separated by a semi-colon.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION
There is a growing emphasis on teaching young people skills in technological production to advance our societal technological progress. A common component of these learning initiatives is that they seek to provide students with skills for *future* careers in technology. As researchers, we believe in the value of working with young people in technological production, but draw upon Dewey's observation that education oriented towards future responsibilities can hinder active participation by students in the present [3]. Instead we support engaging young people in forms of working with technology that are relevant in their *current* lives. We are interested in helping young people learn how to use computation in ways that connect with familiar and creative forms of creation, and enable them to create things that are connected with their present goals and values. Our approach has been to support youth in learning experiences that allow them to design using computation, and then produce a functional artifact based on that design through crafting. We use term *algorithmic craft* to describe the combination of computational design and hand craft.

The combination of computation and hand-craft can occur in multiple forms. In algorithmic craft, we focus on creating digital designs through programming, and translating these designs to physical artifacts using established craft practices. This process raises many difficult questions. What are the important design principles to consider when creating programming environments for physical design? How do we compellingly link programming code with visual designs, and what are appropriate intersection points between textual, visual and physical manipulation? How can we reduce technical challenges in translating code into an object that can be successfully built by hand, while supporting a wide variety of design styles, aesthetics, and approaches? What values are reflected in this form of production, and how are the resulting artifacts relevant to a person's life?

To explore these questions we have developed a software tool called DressCode to help novice programers participate in computational design for physical craft. In this paper, we describe DressCode, and our creation and implementation of craft activities in which high-school students designed forms and patterns with DressCode and applied them to physical objects through screen printing. Through analysis of these experiences, we describe design factors that enable young people to computation and making in ways that emphasize pleasure, exploration, intellectual engagement and utility in the service of personal expression. We conclude with a set of recommendations for the development of tools and activities for future forms of engagement in computational design and making.

CREATIVE PROPERTIES OF ALGORITHMIC CRAFT
Computational design and craft have properties that enable different forms of creation. In computational design, computer programming provides a powerful way of thinking about design, however as a digital medium, computational design lacks direct connections to the material world. Craft on the other hand, is often defined by its material properties, and physical engagement. Algorithmic craft provides a connection between the abstraction of computational design and

the material domain of craft by converting digital concepts to physical forms. Here we describe the properties of computational design and craft and how they are combined in algorithmic craft.

Properties of Computational Design

While often dependent on programming ability, computational design is best represented as a way of applying procedural thinking to a design task [12]. Computational designers author rules that define systems capable of producing many outcomes, enabling the production of multiple design variations that share a set of constraints. Through this systematic abstraction, the following are possible:

- **Precision:** Computation supports high levels of numerical precision.
- **Visual Complexity:** Computational design allows for rapid creation and transformation of complex patterns and structures, allowing for complex manipulations of numerous design elements.
- **Generativity:** Designers create algorithms that allow for the computer to autonomously produce unique and unexpected designs.
- **Parameterization:** Computation allows users to specify a set of degrees of freedom and constraints of a model and then adjust the values of the degrees of freedom while maintaining the constraints.
- **Remixing:** Computationally generated designs are created by a program which can be modified by other designers.

Properties of Craft

We consider craft as a recreational pursuit and artisanal practice. The following properties connect to algorithmic craft:

- **Materiality:** Craft involves working with physical materials by hand. Intuitive decisions are made while physically manipulating the material.
- **Pleasure through physical labor:** Fundamental to traditional conceptions of arts and crafts is the idea of pleasure in working with one's hands [9].
- **Unification of form and function:** Many forms of craft can applied the creation of objects that merge ornamental and useful qualities.
- **Craftsmanship:** Traditional notions of craft are closely connected to the idea craftsmanship: the motivation to produce quality work for its own sake [16].

The Potential of Algorithmic Craft

By bridging computational design and craft, it is possible to blend computational problem solving with intuitive physical engagement. As demonstrated by Zoran's work, the incorporation of craft into a digital process can transform an infinitely reproducible digital form into an object that is unique and connected to a distinct space and time [17]. Algorithmic craft artifacts are shaped by machine processes, the properties of the materials, and the hands of the craftsman. Due to its hybrid nature, algorithmic craft supports diverse design practices and values, and practitioners in this space blur the boundaries between engineer, designer, and artisan.

Challenges in Access and Application

Through its association with hobbyist culture, many consider craft to be accessible. Conversely, computational design is largely limited to experts and professionals, in part due to practical barriers to novice participation. Many of the programming languages used for computational design are difficult for novices to learn and most novice oriented programming environments emphasize producing screen-based work rather than physical objects. Novice-oriented computer-aided-design (CAD) software emphasizes graphic interaction and seldom features computational methods as a key design tool.

Significant perceptual barriers also exist. Many people consider programming to be irrelevant to their interests, and are unmotivated to pursue what they perceive to be a highly specialized and difficult undertaking [14]. There also are perceptions of technology in general which may hinder the integration of craft. Digital technology is often portrayed as reducing the need or desire for manual human labor, rather than supporting it in creative forms. Rosner describes this view:

> *A central element of these and other visions of the future is that craft is done for us: Kitchens tell us what and how to cook, eliminating the creativity and pleasure of cooking from scratch with what's on hand; object printers create flawless prototypes, eliminating messily glued-together chipboard and toothpicks. In this new world, craft becomes fetish [1].*

We chose to address these practical and perceptual challenges through the creation of a tool, called DressCode, for novice-oriented computational design that is compatible with hand craft. Further, we developed of an activity structure to guide young people through in use of DressCode for the creation of wearable artifacts. Prior to describing DressCode, we discuss related works in creative computation and design for young people.

RELATED WORK

Eisenberg and Buechley's research on pervasive fabrication provides a foundation for our objectives in algorithmic craft. Their research encompasses techniques and approaches for incorporating digital fabrication into educational settings, and describes how fabrication enables youth to decorate their environments, allows them to create novel artifacts in the service of personal expression, and stimulates intellectual [4]. The development of DressCode is closely related to research using the Codeable Objects programming library for youth activities in computation, digital fabrication and fashion design [6]. We also examined related software tools: learning oriented programming tools and novel and accessible CAD tools

Learning Oriented Programing Environments

Logo is the seminal novice programming language founded on principles of constructionism and embodiment [10]. Scratch is one of the most successful modern novice-oriented programming environments, and allows users to create interactive projects by combining command blocks [13]. Processing is a language and development environment for cre-

ating complex forms and animations. Described as an entry level programming environment, Processing has also become a successful professional computational design tool[11]. Although neither Logo, Scratch or Processing were explicitly created for compatibility with craft practices, they share features that enable creative programing by novices. They contain a simplified programming syntax, prioritize visual feedback, and are applicable to a diverse range of projects and interests.

Novel and Accessible CAD tools

A number of emerging CAD tools blend novel forms of interaction with accessible design mechanisms. Sketch It, Make It is a 2D CAD tool that allows users to constrain their designs through gestures made using a digital drawing tablet [8]. SketchChair enables the design of a chair by sketching with a computer stylus [15]. FlatCAD connects programming and digital fabrication by allowing users to computationally design construction kits, and fabricate them on a laser cutter[7]. These tools all emphasize various forms of hand engagement in the design process.

In developing DressCode, our goal was to merge the computational functionality of learning-oriented programming environments with the design capabilities of gestural and expressive forms of CAD tools. We did this in a way that allows young people to write code and graphically draw and manipulate forms to produce designs for physical making.

DRESSCODE SOFTWARE DESIGN

DressCode is a 2D computational design tool. The primary objectives in DressCode were to reduce the technical challenges of computational design, to support independent and deliberate design decisions through programming, and to assist in the creation of designs for craft applications. DressCode is general purpose, and can be applied to many forms of creation. The software is comprised of a combined integrated development environment (IDE) and a graphic-user-interface (GUI) design tool and a custom programming language and application programming interface (API) (see Figure 1). Blending the IDE and GUI enabled the development of graphic design tools that work in conjunction with the process of programming. The tools enable graphic drawing and manipulation because the software automatically generates editable code in the IDE that reflects the designer's actions.

Interface

The interface of DressCode is divided into a design panel and a coding panel. The design panel contains a resizable drawing board and pan and zoom tools to facilitate navigation. The print tool opens a dialog that allows the user to export their current design in vector format for output through printing, or 2-axis forms of digital fabrication. The coding panel contains a text editor for writing code and an output console for printing output and error reporting. DressCode also contains a declarative listing view, and a stamp menu, discussed in the graphic tools section. When a program is run with the play button, the resulting design is displayed in the design panel.

Programming Language and Drawing API

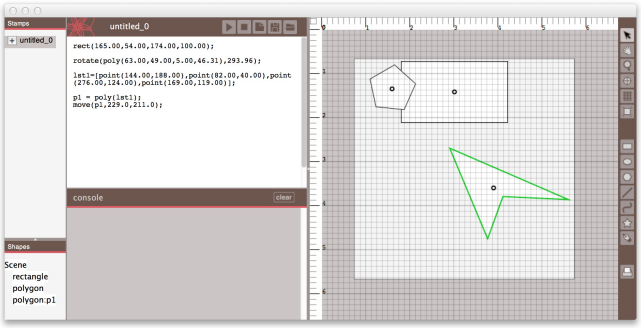

Figure 1. DressCode Software with example of graphically created polygon, rectangle, and irregular polygon, and corresponding automatically-generated code.

The DressCode programming language is interpreted with semantic functionality that is simulated through a Java-based library. For most programs, interpretation is instantaneous; however, complex operations require several seconds to be executed. DressCode features a custom textual programming language with native 2D drawing functionality[1]. The language supports conventional programming datatypes as well as drawing primitives in the API. The language is fully featured with support for loops, conditionals and user-defined functions. Variables in DressCode are dynamically typed in order to reduce syntactic challenges for novices. Identifiers can be assigned to datatypes that differ from their original assignment at any point. The language also has built in math and random methods including random noise generation.

The DressCode drawing API is formulated on an Object Oriented Programming paradigm enabling the users to create geometric primitives (points, lines, curves, polygons etc.), and manipulate them as objects. Primitives are initialized by calling the appropriate method and passing it a set of parameters designating the primitive's location and dimensions. There is no "draw" method for DressCode. All primitives are automatically drawn in the order of their initialization in order to simplify the steps for a design to appear on the screen.

Primitives can be modified through two kinds transformation methods, geometric and stylistic. Geometric transformations allow for primitives to be rotated, scaled, moved or combined with other primitives through polygon boolean operations, (union, intersection, either-or and difference), and are performed relative to the origin of the primitive, unless otherwise specified. Stylistic transformations modify the appearance of a primitive (color and stroke weight). Transformations are performed by assigning an identifier to the primitive, and then calling the transformation method with the identifier. By using the transformation methods to manipulate primitives it is possible to generate complex and generative designs from the repetition and structured distribution of simple forms. As a method of organizing sets of primitives, DressCode contains a group datatype, a specialized list for organizing and performing collective transformations on multiple primitives. Groups

[1] A full specification of the DressCode language and API is available at: `ommittedforannonymity`

facilitate more advanced transformations including clipping masks and collective merge operations.

We created a custom textual programming language for DressCode because we believe textual programming enables transparent representations of computational design algorithms. Because we recognize that textual programming is challenging for novices, we augmented the programming language with visual interaction and feedback in the form of graphic drawing tools.

Graphic Tools

The graphic tools in DressCode allow users to create and modify elements of their design through graphic selection. The tools maintain a direct symmetry with the DressCode programming language because each tool correspond directly to a method in the drawing API. The use of each tool automatically generates a corresponding textual statement in a user's program. This functionality is designed to encourage natural transitions between graphic drawing and textual programming because it enables elements that are created graphically to be manipulated through textual programming and vice-versa.

Figure 2. The graphic drawing and manipulation tools in DressCode (from left to right: selection and move tool, rectangle tool, ellipse tool, regular polygon tool, line tool, curve tool, SVG import tool, pen tool)

The toolset includes regular primitive creation tools and a pen tool for the creation of irregular forms (see Figure 2). In addition to the drawing tools, the selection tool allows for individual primitives and groups to be manually selected and moved. When a primitive is moved for the first time, a textual move statement is inserted into the user's program. For all subsequent moves of that primitive with the move tool, the inserted move statement is updated to reflect the new coordinates of the primitive (see Figure 1).

The declarative view contains a listing of all primitives in the current design. Child primitives are nested within their parent groups. When selected in the declarative view, the primitive is selected and highlighted in the design view, and the line where the primitive was last modified in the text-editor is highlighted (see Figure 3). The declarative view is designed to provide visual feedback on how elements of a design connect to the user's program, and provide a practical selection technique for complex designs.

DressCode contains functionality to help people organize their code in the form of static and dynamic *stamps*: graphically created functions that return shape primitives. Dynamic stamps are created by selecting a portion of code in a user's program and then selecting the dynamic stamp option from the menu. A dynamic stamp will package the selected code in a function with a name specified by the user. Static stamps are created by graphically selecting a single primitive or group with either the selection tool or the declarative view, and selecting the static stamp option. Static stamps translate shapes generated in random positions to explicit primitives, allowing

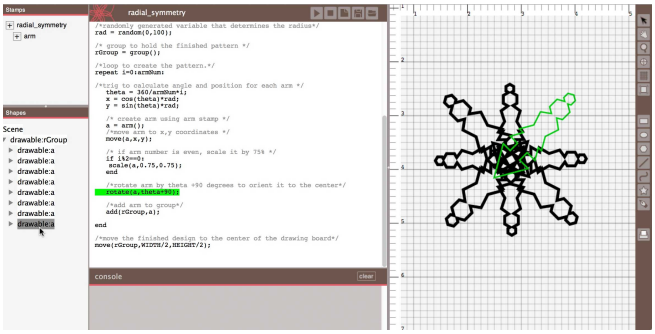

Figure 3. Declarative view with selected primitive

users to save a specific instances of a generative design (see Figure 4). Stamps are listed in the stamp menu and can be added to a user's primary program by selecting the + icon next to each stamp. The code of both static and dynamic stamps can be modified by the user as the code generated is human readable.

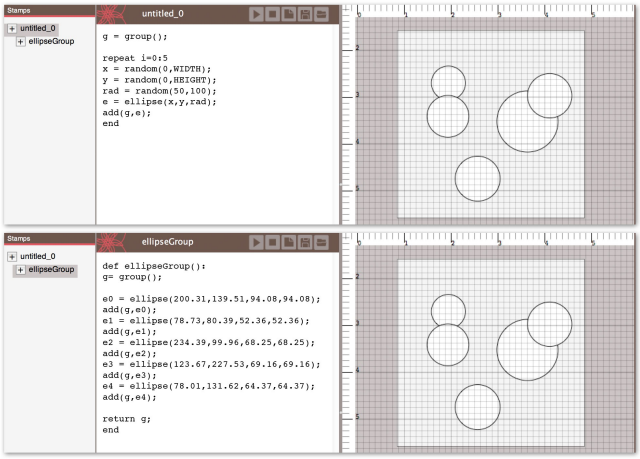

Figure 4. Static stamp functionality. (Top: User defined code which generates five random ellipses. The ellipses' positioning and size will change each time the program is run. Bottom: static stamp created from ellipses which will always return the same design.)

ACTIVITY DESIGN

Domain-specific software is an important component of engaging youth in algorithmic craft; however it is equally important to appropriately design the context in which the software is applied. In this section, we discuss the primary components of our activity design using DressCode: *learning, creation and reflection*. These components are directly influenced by the Brennan and Resnick's dimensions of computational thinking[2], but are specific to algorithmic craft.

Learning

Learning components of activities relate to participants' understanding of the core affordances of computational design and the values of craft. Computational learning in algorithmic craft entails learning programming concepts and understanding how they relate to design objectives. Craft learning involves learning technical craft practices and evaluating designs in respect to these practices.

Creation

Creation in algorithmic craft is comprised of *iteration and repetition*, *remixing*, and *critique*. In computational design, iteration occurs as designs evolve through incremental steps. Craft involves repetition, where successful execution requires practice of manual skills. Remixing plays a key roles both in computational design and craft. In computational design example programs serve as starting points for novices and existing designs are merged to produce something new. In craft people share parts and techniques and produce multiple variations of a design.

Extension

Algorithimic craft emphasizes extension through *reflection, sharing and connection*. Reflection occurs as practitioners examine and evaluate their creations. Sharing extends reflection as practitioners share the process behind their artifacts with friends and family. Connection occurs when the experience of one project generates ideas for future pursuits.

USER STUDY

To better understand young people's practices in algorithmic craft, and evaluate the effectiveness of DressCode, we conducted an algorithmic craft activity using DressCode as the primary tool. The activity consisted of a four-day workshop, divided between two consecutive weekends, in which participants created computational designs in DressCode and screen-printed them onto t-shirts.

Evaluation Methods

The participants' experiences were evaluated through written surveys and group discussions. Pre-workshop surveys focused on participants' previous experience and attitudes towards programming, craft and design. Post-workshop surveys contained attitudinal questions that were matched to the pre-surveys, as well as a range of written questions asking the participants to describe their process and project. Interim surveys were also administered following the first two days, and asked participants about their experiences using the DressCode software. We held three group discussions with the participants which were recorded and transcribed. The discussions examined participants' design process, experience using DressCode in combination with craft practices, and ideas they had for modifying or augmenting future activities and software tools. Survey results, verbal discussion responses and project outcomes were analyzed to identify repeated and notable elements of participants experiences, and how they connected to our original research objective of helping people connect to computation in ways that are relevant, personally meaningful and intellectually engaging.

Demographics of Participants

The screen printing workshop was conducted among 7 young adults, aged 13-17, and one older participant, aged 21. Three participants were male, and five were female. Participants were selected to represent a range of programming, craft and design experience (table 1).

	no experience (1)	2	3	4	expert (5)
art	0	0	4	4	0
craft	1	0	3	4	0
programming	1	1	4	1	1
design	0	2	3	3	0

Table 1. Prior participant experience. (Numerical values in the columns correspond to the number of participants with that response.)

Workshop Progression

Prior to the workshop, we had participants select t-shirts in a size and color of their preference. In the first session of the workshop, we introduced participants to generative design and programming in DressCode by guiding them through writing their own random walk algorithms that produced patterns composed of lines. We followed by explaining the graphic drawing tools, and had participants modify their random walk patterns by incorporating shapes they created graphically. The following session began with an overview of photo emulsion screen printing and transitioned to computational design activities. We demonstrated several other forms of generative design by providing participants with images of sample designs and having them work in groups to write a set of instructions to re-produce the design. As a group we discussed their results, and looked at code examples in DressCode that would produce results similar to the samples. Participants were given the task of creating a computational design to screen print onto a t-shirt they would want to wear, and given 4-5 hours open design time divided between two days. Other portions of the workshop were spent preparing screens for screen printing. Participants laser-printed their finished designs on transparencies, which the instructors exposed on the screens overnight (see Figure 5). On the last day, participants practiced screen printing and printed their final designs onto their shirts.

Figure 5. Screen printing process. (Clockwise from upper-left: Digital design, printed transparency of design, stretching the screen, applying photo emulsion, exposing the screen with the transparency, washing out the screen to reveal the design, printing with the finished screen, a completed print.)

Screen Printing Results

Each participant in the workshop was successfully able to use DressCode to produce a design for their t-shirt (see Figure 6). The design approaches among the participants varied. Two participants created designs that were derived from their random walk algorithms, one participant created a pattern by

combining two of the example patterns and making adjustments, and one participant worked solely by modifying an example. Other participants designed patterns independent of the examples, including a generative landscape, a geometric spiraling pattern and a complex radial pattern comprised of overlapping lines. The screen printing process was extremely popular among the participants. Each person successfully created their own screen through photo-emulsion transfer, and mastered the basics of printing. Several participants not only printed to the provided shirts, but also brought in additional garments to print on for friends and family. The participants requested to keep their screens following the workshop, and stated on the survey that they planned to continue making prints for themselves and others. All participants indicated that they planned to wear their shirts, and two participants contacted us via email following the workshop thanking us for the experience, and requesting tips on how to properly care for their garments.

Figure 6. Some of the completed shirts. (Clockwise from upper-left: generative landscape, clipping mask cloud pattern, geometric spiral, random walk of hand-drawn hearts, dandelion-random walk mashup, generative linear-radial pattern.)

An evaluation of the pre, mid and post-workshop surveys demonstrated that following the screen printing, participants attitudes towards programming, design and craft, changed from what they were after the computational design sessions. In the majority of cases, this change was positive; participants indicated greater interest in learning programming in the future, a stronger belief that programming was a tool that they could create things they would use in their daily life, and greater comfort in programming on their own following the craft activity. Participants were positive about the graphic tools, although they requested that their functionality be extended to incorporate a greater range of transformation methods. All participants said they would be interested in using DressCode for another activity, and all but one indicated they would like to continue using the programming language in particular.

DISCUSSION

In our discussion of the workshops we focus on three primary elements. We evaluate the successes and limits of the DressCode software, with an emphasis on how people used the graphic drawing and manipulation functionality. Second, we examine participant's design practices and discuss how they connect to the affordances of algorithmic craft. Third, we consider what participants accomplished in the workshop by evaluating how the completed artifacts resonated with the participants' personal values and lives.

	str. disagree	disagree	neutral	agree	str. agree
helped create design	0	1	1	3	2
helped understand program	0	0	5	3	0

Table 2. Participant survey responses on the role of graphic tools. Numerical values in the columns correspond to the number of participants with that response.

Combining Graphic Manipulation and Textual Programing

One of our primary research objectives was to develop techniques that were accessible while also supporting personal aesthetics and styles through computational design. Analysis of participants' use of DressCode indicates that for many people, the graphic tools were helpful in this regard (table 2).

Before the graphic tools were formally introduced, several participants independently started experimenting with them on their own. Several people also indicated that the graphic drawing tools also helped them to better understand the process of writing their program. Participants expanded on this idea during the discussion:

> *You find a tool, you draw on the canvas and it shows you the code, that's basically it.*
> Participant Z

> *I think that having the drawing tools and having it also show the code there makes it so that you can see as you're doing it. [By] having a graphical side and having it auto update the code, it can show you that you want to work with the code you're learning as you're using the GUI. So that people could try and say ok, if I can't do something with the graphical tools, let me try and manipulate the code. And then you already have some understanding because you've been using the graphical tools and it's been appearing over there the entire time.*
> Participant B

A visual examination of many of the finished products in the workshop demonstrates the impact the graphic drawing tools on the aesthetics of the designs (see the heart and cloud designs in Figure 6). The pen, and shape generation tools allowed participants to blend their personal drawing styles with computational elements, resulting in computational forms that reflected the drawing techniques of the creator.

The introduction of the graphic tools also resulted in a discussion among participants about future paradigms for graphic design and programming. Participants unanimously agreed on the utility of the drawing tools, but requested that they be developed further to provide more sophisticated forms of drawing, control, and visual feedback. In contrast to the popularity of the drawing tools, several participants had difficulty

with the move tool. From a development perspective, the move tool was challenging to implement because there were numerous ways it could be represented in code. We implemented what we felt to be the simplest option (inserting a move statement for primitives with identifiers, or wrapping the declaration statement of un-identified primitives), but doing so required us to speculate about what would be most useful and intuitive for novice programmers. In practice, participants who had large numbers of primitives in their programs without identifiers found the move behavior confusing because it modified an existing statement in their program, rather than generating a new line of code. This confusion makes it clear that we should re-consider the functionality of the tool, however it also points to a design principle for future tools. The fewer assumptions a tool has to make about the intentions of the user, the less likely it is that it will conflict with the user's coding and design process. We feel the creation of more sophisticated primitive generation tools is a form of low hanging fruit that will provide immediate benefits to the user's design process, whereas transformation tools require greater analysis on how to be implemented effectively.

Participants responses in the discussion also demonstrated ways in which the tools had expanded their comprehension of the computational design process. One participant suggested adding an graphic erasing tool, which resulted in a conversation on how such a tool could be implemented:

> Participant Z: *So if you erase one line, it becomes two lines...so it's going to generate the code for two lines?*
> Participant M: *No like if you drew a line with the sidebar, and then you decided you didn't like, you don't have to go to the code, erase it and press play, you could just take the erase tool, click on it and then it goes away.*
> Participant Z: *No, what I mean is like if you just wanted to erase half?*

This conversation reinforces issues when making assumptions about individuals' design processes in the implementation of graphic tools. Moreover, this discussion shows the participants actively considering how new graphic tools could correspond to writing code. Conversations of this nature demonstrate how graphic tools can stimulate thinking about computational processes for young programmers, rather than making them opaque. During the workshop, a similar discussion occurred around the topic of preserving generative forms. Participants had different ideas for preseving static instances of randomly generated designs. Several participants used the static stamp tool for this purpose, but others requested different forms of control over the randomly generated values. One participant wondered if he could have the software store the randomly created values for a specific program, and allow the user to specify the point at which new values would be generated. Building on this idea, we asked participants if they would have preferred to specify random noise generation through a form of graphic input similar to the graphic dialog tools, or if they felt creating random methods in the programming environment was useful. Two participants said they would prefer that as an option, however others disagreed, stating that they felt you would lose some of the functionality of the program in the process.

> *Well that would kind of take away from the programming part of it, because you'd be trying to turn everything into a UI. The important thing I really feel about DressCode is you can make things random, these are drawn (indicating a hand drawn graphic) and these are from the programming part (indicating the random repetition of the graphic).*
> Participant Z

Understanding the creative potential of one's tools is essential for effective design. It is encouraging therefore, that many participants not only saw the value of the graphic tools, but also understood and embraced the applications of textual programming in combination with these tools.

The introduction of graphic manipulation tools in conjunction with textual programming demonstrated several conclusions. Drawing tools proved to be largely intuitive and easy to use and they allowed participants to blend their personal drawing style with computational forms. The incorporation of drawing styles is highly relevant to our objectives in algorithmic craft, because it allows people to create artifacts that are personally unique and conform to individual stylistic. Furthermore, the discussion of the software demonstrated that the tools assisted in communicating basic computational principles as evidenced by participants description of how they would want additional tools to function. Lastly, despite participant's requests for more sophisticated forms of transformation tools, we feel their implementation should be tempered by careful consideration of a successful design strategy for integrating them with diverse approaches in programming.

Algorithmic Crafting in Practice

If the graphic tools assisted with some of the technical challenges of algorithmic craft, we are still left with questions regarding the actual practice of algorithmic craft. How did the experience affect participants perceptions towards computation and craft? Were individuals able to take advantages of computational design affordances in ways that connected to their personal design goals? What effect did feedback mechanisms have on the success of people's projects? We begin our discussion of these components by describing changes in participants' perception.

Following the workshop, participants displayed an increased awareness of the design applications of programming.

> *I never thought of using programming as a tool to help create a design. There are so many things you can design using programming languages.*
> Participant R
>
> *When I got the hang of [DressCode], I loved how much you could do with it. There's such a broad range of design possibilities, and with every new one you create, you learn more about the various tools and become more familiarized with DressCode.*
> Participant E

More importantly, the participants demonstrated a nuanced understanding of the process of design, and carefully considered how computation could play a role:

> *Before I thought that design was relatively simple, but now I see that a lot of thought has to go into what you're making and the process of how to make it.*
> Participant E

> *I've learned new controls like the random code and it's really just small steps and things I learn through DressCode that allow me to fully express my creative side and expand my knowledge of design.*
> Participant J

A general perception of the applicability of programming to design is a fundamental step to participating in algorithmic craft. It is challenging however for people to understand the specific affordances of computational design and incorporate them in their practice. The participants performed better than we hoped in this respect. Combined evaluation of their end products, and their statements from discussions and surveys point to the clear presence of the principles of computational design. When asked how using DressCode had changed his creative process, one participant responded that he was trying out ideas that were similar to work he had created before the workshop, but now he had the opportunity to execute them with greater *precision*. Another participant described her design process this way:

> *It's very different- I draw everything, and I never could've created my design by hand.*
> Participant J

Her description of her process, like her design, revolves around the idea of the *visual complexity* that is enabled through computational iteration and repetition (see the random-walk dandelion mashup in Figure 6). Her design is also an example of *remixing* because she combined portions of the code from an example with a program she had written, making adjustments to both components to produce a novel composition. Another participant borrowed the generative landscape creators' code for stars and added it to his design. The landscape design provided a perfect example of the benefits of parameterization, when in the last five minutes of the design session, the participant decided that a horizontal composition would look better on his t-shirt than a vertical one. Since he had defined variables for the random distribution of clouds, mountains and stars that corresponded to the width and height of the drawing board, when he resized his drawing board, the design was automatically re-generated to correspond to the new size. The participant stated in the survey that:

> *[DressCode] made changes much easier to change, rather than redoing the entire design*
> Participant R

Generativity played a role in almost all designs, as participants experimented with adding some form of noise in to their code. Below, a participant with no prior programming experience describes her experience in using gaussian noise in her design:

> *The random thing.. if you made it random by hand then it wouldn't really be random, because you would just put it in a special place.. so the computer just has to choose where to put it.*
> Participant M

The participant who created the randomized radial pattern demonstrated the most elaborate use of generative design in the workshop. During the second critique, he presented 17 unique patterns. This astonished the other participants who were surprised to see that such variety could be produced by one algorithm. He described his process this way:

> *[DressCode] allowed me to see random patterns such as the one I made. I may have been able to [create them] otherwise, but I probably wouldn't have thought to.*
> Participant B

Aside from implementing specific affordances of computational design, participants also ran in to challenges that are particular to computational design. One participant used gaussian noise to distribute elements of her design, but wanted to deviate from this structure at several key instances for emphasis.She struggled in determining a way to have her algorithm deviate from random placement at arbitrary points. Her struggles touch upon the larger challenge in computational design of creating singularities. Because computational design is governed by a systematized ruleset, the methods of breaking these rules in arbitrary fashions are often unclear or tedious to implement. The participant who created 17 instances of his design was challenged when it was time to choose a single instance to screen print with. Computational design gives the designer the ability to produce extremely large numbers of solutions to a single design problem. While this is useful in situations where multiple solutions are required, when a single design must be chosen, the process of deciding on a solution is difficult, especially if the decision is based on aesthetic criteria.

The challenges participants encountered in the design process were addressed primarily through two forms, instructor assistance and group critiques. Although instructor assistance was important to the success of the workshop, in many ways it mirrored instructor help in a general programming context and consisted of helping to correct syntax errors and pointing participants towards relevant programming methods. Here, we focus on the group critiques as the most interesting and novel form of design feedback that occurred in the workshop. As we explained to participants, in providing design critiques, the role of the group is to understand the creative goal of the designer, and offer advice on how they can reach that goal. Prior to the workshop, most participants had not participated in design critiques, and they were hesitant to provide feedback. Through successive sessions participants gradually became comfortable in voicing opinions. By the end of the workshop, several participants stated that the critiques had helped to improve their design:

I started with a design I was ok with, but group critique and getting others' opinions helped me create a design I loved.
Participant E

During the critiques, designs were collectively "debugged" by the group. Feedback was simultaneously provided in the form of stylistic suggestions and technical tips on how modify one's code to achieve the stylistic goal. In one instance, a participant noticed that the linear spiral pattern we were critiquing came close to producing a Moir effect. The designer said that she had noticed the effect too, and wished to make it more apparent. As she demonstrated the code for the pattern, other participants began suggesting values she could modify to change the density of the lines and produce a more intense effect. Debugging often occurred in a craft context as well. When critiquing a design, participants considered the material limitations of screen printing and offered suggestions about how to make a design correspond to these factors.

Group critiques also applied well to the challenge of generative design selection. The critique provided the participant with 17 design instances the opportunity to receive group suggestions on what his final design should be, and in the process, stimulated a discussion on the challenges of generatively among the participants. Critiques in the workshop ended up serving a purpose beyond optimizing the designs of the participants. In the process of critiquing others' work, participants *learned* from their peers about the computational approaches behind their designs, and *thought about* the challenges and opportunities of this form of making.

As a whole, the workshop changed participants' perceptions of the creative affordances of programming by allowing them to produce their own computational designs. Through the production of these designs, participants demonstrated both the application and understanding of key computational design affordances. Finally, group critiques helped people to improve their designs, enabled them to learn from peers about computational processes, and provided the opportunity for intellectual discussion about the advantages and challenges of combining computational design and craft.

Physical and Digital Notions of Value

An analysis of participants use of the DressCode tool and design practices in the workshop reveal that participants learned about computational concepts and applied them in creative and intellectually engaging ways. In this section, we focus on what participants felt they accomplished during the workshop, what they enjoyed, and how the combination of computational design and craft resonated with the participants personal values.

The participants varied individually on elements they found most difficult and most enjoyable. Several found the process of coding most difficult, others found the overall process of design challenging, while others described the screen printing itself as the most challenging:

The hardest part for me was actually putting the design from my screen onto any material. I had to make like 10 copies before I put it on the shirt.
Participant M

I enjoyed learning and understanding the process of programming. There were some math concepts that I have yet learned (i.e trigonometry) so that made some parts of the workshop difficult to understand. But overall, I did enjoy it!
Participant L

I enjoyed being able to do things myself and get help when I needed. I found it frustrating how my screen printing kept on fading
Participant R

Many of the female participants in particular, found the process of screen printing to be highly rewarding:

I loved learning to screen print, and getting to learn how DressCode and screen printing were both incorporated into craft was interesting.
Participant E

Participants also remarked on the differences between difficulty in screen printing versus computational design:

I feel as though maybe like when you're programming stuff, you're on the computer but you're not using any physical work, but when you're screen printing there's the manual labor involved, so when you actually get something right it's I guess more gratifying, since you put hard work [into it]... Well you do put hard work into programming, but it's more a thinking thing, not actual physical labor.
Participant L

We followed up by asking participants if the physical labor in craft changed how they felt about what they were making:

Participant L: *After all this hard work, you have this (pointing to the t-shirt she is wearing)*
Participant R: *It lets you take pride in it.*

These responses demonstrate that participants experienced enjoyment and personal satisfaction as a result of working with their hands. In addition, despite the difficulty of screen printing, participants made efforts to produce a well-crafted artifact. Furthermore, participants indicated that after screen printing, they were more interested in computational design, and that the process produced additional ideas for future programs they could create in DressCode. The pleasure, pride and craftsmanship demonstrated by participants indicate how the values of craft provide positive forms of motivation and engagement for computational learning and expression.

In addition to the expression of these values, participants emerged from the workshop with artifacts of their own design and creation. The fact that all participants planned to wear their t-shirts and also naturally began making copies for friends and family indicate that screen printing possesses a strong cultural relevance for both young men and women.

There were also questions around what the final artifacts consisted of. Participants were excited about their shirts, but many were even more enthusiastic about keeping and re-using

their screens. This excitement led to a discussion of ways in which the screen could be used: for the creation of additional copies, as a means to apply new computational designs, or for display as an art object itself. Several participants even brought up ideas concerning entrepreneurship and discussed ways one might sell artifacts created in this manner. These reactions demonstrate the diversity of ways that craft practices can resonate with the principles of computational design and connect to present and future goals of young people.

DESIGN RECOMMENDATIONS

Based on our evaluation of the workshop, we conclude with a set of recommendations for future work in developing activities and tools for algorithmic craft.

- **Tools should support design techniques that demonstrate the benefits of computational design.** Tools for algorithmic crafting should effectively communicate the affordances of computational design in ways that are applicable to a range of experience levels and design styles.
- **Programming languages should be augmented with visual feedback and intuitive drawing methods to support multiple forms of creation.** Algorithmic crafting tools should feature multiple access points for creating designs, including graphic drawing tools that are familiar and accessible to novice programers.
- **Activities should encourage discussion and group critiques of work, and when possible, use in-person facilitation and guidance.** Algorithmic craft benefits from discussion among peers, critique, and reflection.
- **Materials matter.** The use of rich, interesting materials greatly enriches the experience of algorithmic craft and contributes to the success of the finished artifacts. Care should also be taken use construction techniques that will hold up over time.
- **Consider the cultural implications of algorithmic craft experiences.** Because algorithmic craft can provide a form of self-expression and communication, both craft activities and computational tools should aim to be mindful of the gender, age, cultural background, and social norms for participants.

CONCLUSION

As computers grow in complexity, and ubiquity, it is possible to regard computation as outside the reach of most individuals, and incompatible with familiar forms of making. In our experience, the opposite is true; computation can enhance established forms of creation. As a way of making, algorithmic craft gives people the opportunity to experience the pleasure that is possible when using programming, and one's hands to build something entirely new.

REFERENCES

1. Bean, J., and Rosner, D. Old hat: Craft versus design. In *Interactions Magazine* (2012).
2. Brennan, K., and Resnick, M. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the Annual American Educational Research Association* (Vancouver, Canada, 2012).
3. Dewey, J. *Experience and Education.* Touchstone, New York, 1938.
4. Eisenberg, M., and L., B. Pervasive fabrication: Making construction ubiquitous in education. In *Computing and Communications Workshops, IEEE* (White Plains, New York, 2007).
5. Hooper, P., and Freed, N. The spiro inquiry: exploring the application of an inquiry structure for learning science content to teaching with a digital design fabrication tool. In *IDC, AMC* (New York, 2013).
6. Jacobs, J., and Buechley, L. Codeable objects: Computational design and digital fabrication for novice programmers. In *CHI, ACM,* (Paris, France, 2013).
7. Johnson, G. Flatcad and flatlang: Kits by code. In *VL/HCC, IEEE,* (Herrsching am Ammersee, Germany, 2008).
8. Johnson, G. Sketch it, make it. `http://sketchitmakeit.com.`, 2012. (Accessed: 06/3/2013).
9. McCullough, M. *Abstracting Craft: The Practiced Digital Hand*. The MIT Press, Cambridge, Massachusetts, 1996.
10. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas.* BasicBooks, New York, 1980.
11. Processing. `http://www.processing.cc.`, 2001. (Accessed: 06/3/2013).
12. Reas, N., McWilliams, C., and LUST. *Form and Code: In Design, Art and Architecture, A Guide to Computational Aesthetics*. Princeton Architectural Press, 2010.
13. Resnick, M., Maloney, J., Monroy-Hernndez, A., Rusk, N., et al. Scratch: programming for all. In *Communications of the ACM* (2009).
14. Resnick, M., and Silverman, B. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 Conference on Interaction Design and Children* (2005).
15. Saul, G., Lau, M., Mitani, J., and Igarashi, T. Sketchchair: an all-in-one chair design system for end users. In *TEI, ACM* (Funchal, Portugal, 2011).
16. Sennett, R. *The Craftsman*. Yale University Press, New Haven and London, 2008.
17. Zoran, A., and Buechley, L. Hybrid reassemblage: An exploration of craft, digital fabrication and artifact uniqueness. In *LEONARDO*, vol. 46 (2013).