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

3rd Author Name
Affiliation
Address
e-mail address
Optional phone number

ABSTRACT

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 technological progress in the future. A common component of these learning initiatives is that they seek to provide students with skills for *future* careers in technology. As researchers, we feel strongly that working with young people in making and technological production presents great opportunities for learning. We believe however in the value of in engaging young people in forms of working with technology that are relevant in their current lives, as opposed to preparing them for future applications. We are interested in helping young people learn how to use computation in ways that connect with familiar and creative forms 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* as a way of describing the combination of computational design and hand craft.

The combination of these disciplines raises many difficult questions. What are the important design principles to consider when creating programming environments for physical design? How do we compellingly link textual code with visual designs, and what are the appropriate intersection points between textual manipulation and visual manipulation? What support is required to help people move back and forth from programming to building real objects in a way that is comfortable, expressive, and pleasurable? How can we remove the technical challenges involved in translating code into an object that can be successfully built by hand, but still support a wide variety of design styles, aesthetics, and approaches? What values and concepts can people gain from creating artifacts in this manner, and how do they apply to other parts of a person's life? To explore these questions we developed a software tool called DressCode, to enable novice programers to participate in computational design. We designed and implemented of a set of craft activities in which high-school students designed artifacts in DressCode and then realized them in physical form through a combination of digital fabrication and hand craft. Throughout our description and analysis of these experiences, we discuss design factors that enable young people to use computation in ways that emphasize pleasure, exploration, and utility in the service of personal expression. We then describe how participating algorithmic craft creatively engages young people in intellectually during the activity, as well as providing motivation and inspiration for future forms of learning and creation. 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
As individual disciplines, both computational design and physical craft have specific properties that support distinct forms of creation. In computational design, the abstract qualities of computer programming provide a powerful way of thinking about design. Although individual applications of computational design often consider material properties that are relevant to a specific domain, computational design does not possess an inherent connection to the material world. Conversely, hand craft is closely linked to materiality and the physical world. Algorithimic craft provides a connection between the abstraction of computational design and the material domain of craft by converting digital concepts to physical forms. In the following section we briefly describe the properties of computational design and craft in greater detail and describe some forms of how digital fabrication can serve as a connection between them.

Properties of Computational Design
Although computational design can embody a range of practices, our use of the term refers the process of using computational processes to create visual forms and patterns. Because computational design usually requires the designer to write

programs, it is possible to mistake the practice of computational design as a technical skill rather than a way of thinking [8]. While it is dependent on programing ability, computational design is better represented as a way of applying procedural thinking to a design task. Rather than producing specific design representations, the designer authors a set of rules that define a system capable of producing many outcomes. Designs are presented in abstract terms, resulting in the potential to create multiple variations that share a set of constraints. Through this abstraction, the following properties are possible:

- **Precision:** Computation supports high levels of numerical precision.
- **Visual Complexity:** Computational design allows for the rapid creation and transformation of complex patterns and structures, allowing for the combination and manipulation of large numbers of simple elements in a structured manner.
- **Generativity and Self-Similar Forms** Computation allows for the programmer to 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 of the original model.
- **Documentation and remixing:** Computationally generated designs are created by a program which can be modified by other designers and serve as documentation of the design process.

Properties of Craft

The cultural connotations of craft have varied throughout history, however forms of craft have endured, both as a recreational pursuit, and as an artisanal practice. Here we describe the specific aspects of craft that are integral to algorithmic craft.

- **Materiality:** Craft involves the manipulation of physical materials by hand, which is often an intuitive physical process. The decisions made in the craft process are altered by the feel of working with the material.
- **Pleasure:** Fundamental to traditional conceptions of arts and crafts is the idea of pleasure in working with one's hands [6]. This emphasis on pleasure is retained in conceptions of professional and amateur craft practices today.
- **Unification of form and function:** Although not all craft is functional, many forms of traditional craft can applied the creation of useful objects. In addition craft often emphasizes the importance of beauty in the form and ornamentation of objects. Well crafted artifacts frequently demonstrate the successful unification of aesthetics and utility.
- **Craftsmanship:** Traditional notions of craft are closely connected to the idea craftsmanship: quality for its own sake [11]. Craftsmanship is dependent on manual skill and experience, but can also be described as a form of intrinsic motivation.

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 hybrid-reAssemblages, 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 [12]. Algorithmic craft artifacts are shaped by the machine, 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 people consider certain forms craft to be accessible, and open to amateurs. Conversely, the combined use of computational design and digital fabrication is largely limited to experts and professionals. There are many practical barriers to novice participation in this domain. Most prominently, many of the programming languages used for computational design are difficult for novices to learn. Although novice-oriented compter-aided-design (CAD) software exists, most of this software does not support computational design. Similarly, most novice oriented programming environments cannot produce designs that are suitable for fabrication.

Significant perceptual barriers to participation also exist. There persists among the general public a limited perception of the applications of programming. Many people consider programming to be irrelevant to their interests, and as a result, they lack motivation to pursue what they perceive to be a highly specialized and difficult undertaking [9]. 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—the proudly displayed collection of vinyl records shelved alongside an iPod and digital files [1].*

We chose to address these practical and perceptual challenges through the creation of a novice-oriented computational design tool that is directly compatible with a range of hand crafts. Furthermore, we developed of an activity structure to guide young people through the use of this tool for the creation of a wearable artifact. In the following section we

discuss related software tools we evaluated in the process of developing of DressCode.

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 create novel artifacts in the service of personal expression, and stimulates intellectual activity [**?**]. In developing the DressCode software, we focused on two forms of related tools: learning oriented programing tools and novel and accessible CAD tools

Learning Oriented Programing Environments

Logo, a text-based computational drawing program, is the seminal novice programming language founded on principles of constructionism and embodiment [**?**]. The Scratch visual programming language is one of the most successful modern novice-oriented programing environments, and allows users to create interactive projects by combining command blocks [**?**]. Finally, Processing is a Java-based programming language and development environment created for creating complex forms and animations. Processing is described as an entry level programming environment but is also an extremely successful professional computational design tool[7]. Although neither Logo, Scratch or Processing were explicitly created for compatibility with craft practices, they are excellent examples of novice oriented programing tools that support creative expression. They all contain a simplified programing syntax, prioritize visual feedback, and are applicable to a diverse range of projects and interests.

Novel and Accessible CAD tools

There are a number of emerging CAD tools that blend novel forms of interaction with accessible mechanisms for creating digital designs. Sketch It, Make It is a 2D CAD tool that allows users to constrain their designs through gestures made using a digital drawing tablet [5]. SketchChair allows users to design their own chair by sketching with a computer stylus [10]. The resultant design can then be cut on a CNC milling machine and assembled into a 3D object. FlatCAD seeks to connect programming and digital fabrication, and allows users to build customized construction kits with a laser cutter by programming in FlatLang, a novice-oriented programming language modeled on Logo [4]. Spirogator is a tool that allows users to digitally customize a set of hypotrochoid geared-drawing tools and then view a simulation of those tools in action. The user then has the option of either exporting the resulting design generated by the digital gears and fabricating it directly, or exporting the paths for the gears themselves, and then fabricating them to serve as physical drawing tools [3]. These tools all emphasize forms of hand engagement in the design process, either by drawing with a stylus, or by producing physical components that can be manipulated manually.

In developing our own software, our goal was to merge the computational functionality of learning-oriented programing environments with the design capabilites of gestural and expressive forms of CAD tools, in a way that allows young people both write code, and graphically draw and manipulate forms in the interest of producing designs that could be applied to physical making.

DRESSCODE SOFTWARE DESIGN

DressCode is a 2D computational design tool aimed at young people. The primary objectives in the design of DressCode were to reduce the technical challenges of computational design, to support independent and deliberate design decisions through programing, and to assist in the creation of designs for craft applications. Although the name of the tool reflects an emphasis on wearables and fashion, as an application DressCode is general purpose, and can be applied to many forms of creation. The software is comprised of the following elements: a combined integrated development environment (IDE) and a graphic-user-interface (GUI) design tool and A a custom programing language and drawing application programing interface (API). By blending the IDE and GUI we were able to develop graphic design tools that work in conjunction with the process of programing. The tools enable users to gesturally draw and manipulate forms, because the software automatically generates human-editable code in the IDE that reflects their actions.

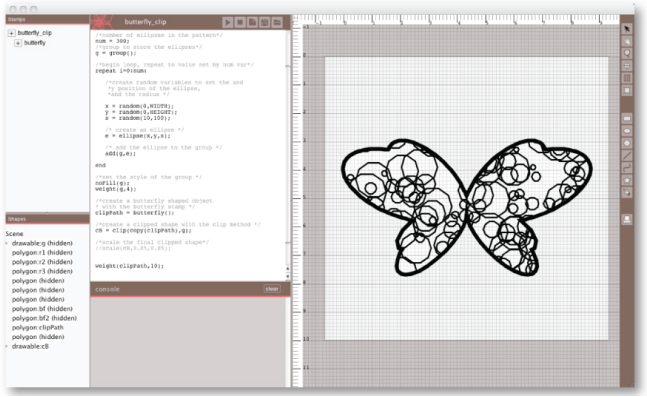

Figure 1. The DressCode software

Interface

The interface of DressCode is divided into two sections: a design panel on the left and a coding panel on the right. The design panel contains features standard to digital graphic design software, with rulers, a grid, a drawing board, and a graphic tool menu. The drawing board defines the coordinate system referenced by the drawing API with the upper left hand corner corresponding to (0,0) in cartesian coordinates. Users can resize the drawing board and set the units to inches or millimeters by selecting the drawing board icon in the tool menu. The pan and zoom tools allow the user to navigate around the drawing board, and the print tool opens a dialog that allows the user to export their current design in vector format for output through digital fabrication.

The coding panel contains a text editor for writing code and an output console for print output and error reporting. It also contains two additional panels, a declarative listing view, and

the stamp menu, discussed in the graphic tools section. A toolbar on top of the coding panel allows the user to save their existing program, open a program, and run their current program. When a program is run, the resulting design is displayed in the design panel.

Programming Language and Drawing API

The DressCode programming language is interpreted with semantic functionality that is simulated through a Java-based library. For most user-defined programs, the interpretation process is instantaneous; however, some programs with complex operations require several seconds to be executed. DressCode features a custom textual programing language with native 2D drawing functionality[1]. The language supports conventional programing datatypes, (numbers, strings, booleans, and lists), as well as drawing primitive datatypes, included in the drawing API. The language supports basic expressions, conditionals, loops, user-defined functions and variable assignment. Variables in DressCode are dynamically typed in order to reduce syntactic challenges for people new to textual programing and identifiers can be assigned to datatypes that differ from their original assignment at any point. The language also has built in math and random functions, including trigonometric functions, a Euclidean distance function, and three random noise generation functions: uniform, gaussian and Perlin.

The DressCode drawing API is formulated on an Object Oriented Programming (OOP) paradigm which enables users to create geometric primitives (points, lines, curves, polygons, ellipses, rectangles and imported SVG primitives), 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. Each primitive is drawn in the design panel relative to this central origin point. 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.

All primitives in DressCode 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. A specialized boolean operation, *expand*, enables the conversion of strokes to filled polygon paths, enabling the translation of line art to a form that will maintain its appearance when fabricated on a laser cutter, vinyl cutter or CNC milling machine. Stylistic transformations modify the appearance of a primitive by setting the fill and stroke color and weight, or prevent the primitive from being drawn. 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 automatically maintain an origin that is the average of all their children's origins. Groups also facilitate more advanced transformations; it is possible to clip a group of primitives within the bounds of a single primitive to serve as a form of clipping mask, and a union can automatically be performed on all objects in a group with the merge method. We chose to create a textual programing language for DressCode because we believe textual programing is well suited to transparent representations of computational design algorithms, however we recognize that textual programing can be challenging for novices. In recognition these challenges and in a desire to extend the types of design which were possible with computational tools, we augmented the programing language with visual interaction and feedback in the form of graphic drawing tools.

Graphic Tools

The drawing and manipulation tools in DressCode are designed to allow users to create and modify elements of their design through graphic selection. Collectively, the drawing functionality of the tools is similar to the functionality of many existing forms of 2D vector graphics software. The tools are distinguished from other graphics software tools because they maintain a direct symmetry with the DressCode programming language. Each tool correspond directly to a method in the drawing API. More importantly, the use of each tool automatically generates a corresponding textual statement in a user's program. This enables elements that are created graphically, to be immediately manipulated through textual programing, and is designed to encourage a natural flow between graphic drawing and textual programing throughout the design process.

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 (ellipse, rectangle, polygon, line curve), and an SVG import tool. There is also a pen tool that allows for the creation of irregular forms (figure:2). Use of the pen tool generates a list of points in the text program, and statement initializing a polygon with the list. Because of the symmetry between the DressCode graphic tools and the DressCode language, there is no functionality hidden from the user in the process of modifying elements graphically. In addition to the primitive generation tools, there is also a selection tool, which allows for individual primitives and groups to be manually selected with the cursor, and moved to different points on the drawing board. When a primitive is moved for the first time, a textual move statement is inserted into the user's program, with the identifier for the moved primitive as the first argument. If the

[1] A full specification of the DressCode language and API is available at: `http://jacobsj.scripts.mit.edu/dresscode/index.php?title=Drawing_API_Reference`

moved primitive does not have an identifier, the primitive declaration is wrapped by a move statement. 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 (figure:3).

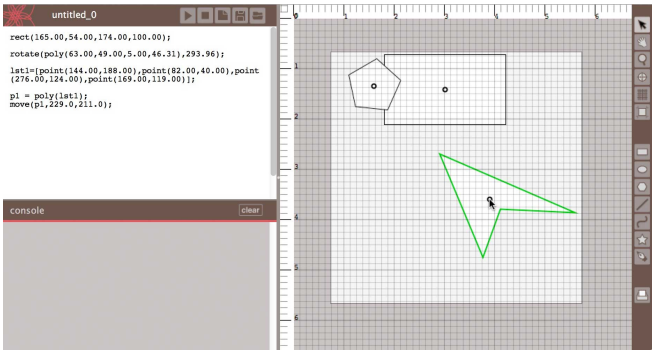

Figure 3. Graphically created polygon, rectangle, and irregular polygon, and corresponding automatically-generated code. (The irregular polygon has just been moved with the selection tool.)

The declarative view contains a listing of all primitives in the current design and their type and identifier. Child primitives of groups are shown as nested elements below the group. When a primitive is selected in the declarative view, the primitive is simultaneously selected and highlighted in the design view, and the line where the primitive was last modified in the text-editor is highlighted (figure:**??**). The declarative 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 selecting and modifying elements within a complex design.

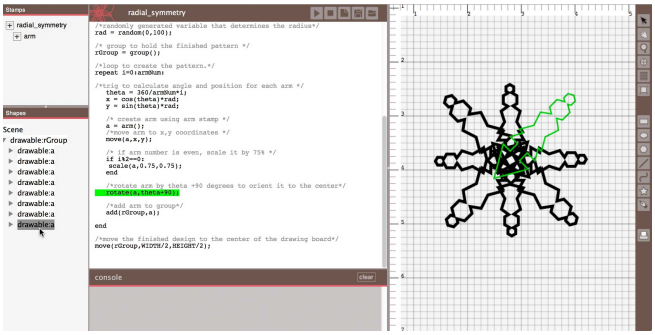

Figure 4. Declarative view with selected primitive

DressCode contains functionality to help people organize their code in the form of *stamps*: graphically created functions that return shape primitives. There are two forms of stamps: dynamic and static. 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, and automatically return any primitives that are generated by the code in the last line of the function. 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 will translate shapes generated in random positions to explicit primitives, allowing users to save a specific instances of a generative design (figure:5). Stamps are listed 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, by selecting the stamp from the menu, which will display the stamp code in the primary code window in an editable format. The code generated by static stamps is human readable, however if very complex designs are selected, the stamp function will consist of numerous lines of code.

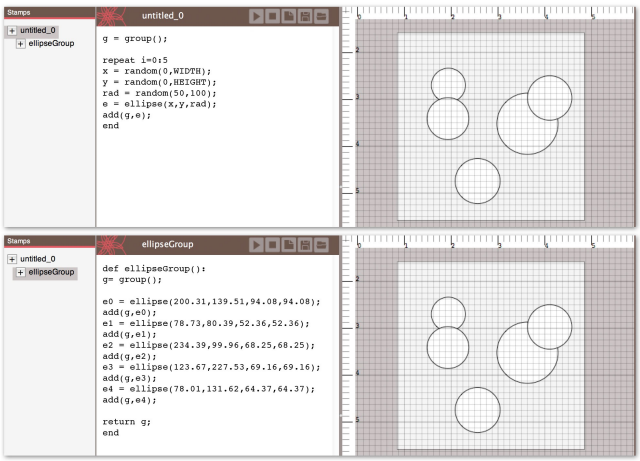

Figure 5. Static stamp functionality. (Top: User defined code which generates five random ellipses. The ellipses' positioning and size will change each time the program is run. Bottom: static stamp created from ellipses which will always return the same design.)

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 to the production of craft artifacts. 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: *concepts, practices and perspectives* [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 programing concepts, and recognizing how these components relate to design objectives. Craft learning involves learning technical aspects of craft practices and evaluating designs in respect to these practices.

Creation

Creation in algorithmic craft is comprised of *iteration and repetition*, *reusing and remixing*, and *critique*. In computational design, iteration occurs as designs evolve through incremental steps. Craft also requires iteration, but also repetition, where successful execution requires practice of manual

skills. Reusing and remixing play key roles both in computational design and craft; example programs serve as starting points for novices and existing designs are merged to produce something new. Parallels occur in craft, where it is feasible to 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. Reflection is often facilitated by the process of sharing, where practitioners engage friends and peers by sharing the process behind their artifacts and giving them as gifts. Participants also make connections 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 programing, craft and design, how personally useful they felt programing to be, and how good a tool they found it for creativity, personal expression. They also asked students to describe their views on design, programing and craft practices as they related to referencing examples, learning from other's work. 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 design process, their opinion on the success of their projects and their experience using DressCode. Interim surveys were also administered following the first two days, and questioned the participants on experiences with DressCode programing language and graphic tools and how the combination of features hindered or supported the design process. We held three group discussion with the participants at the start, middle and end of the workshop which were similarly 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. In both the interviews and discussions, participants were also asked to describe what they enjoyed, what was difficult for them, and what they felt they had learned through this process. 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.

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

Table 1. Prior participant experience

Demographics of Participants

The screen printing workshop was conducted among 7 young adults, aged 13-17, and one older participant, aged 21, three male, five female. Participants for the workshop were deliberately selected to represent a range of programing, craft and design experience (table 1). Participants also varied on their prior experience with digital design software. Two indicated that they had no prior experience with design software, two had previously used the graphics editor in Scratch, three had used Adobe Illustrator or Photoshop, and one person was shown Solidworks in a summer course, but had limited experience with it.

Workshop Progression

Prior to the workshop, we had participants select t-shirts in a size and color of their preference. The first session of the workshop, we introduced participants to the concept of generative design, and demonstrated techniques for incorporating random elements into a pattern or graphic. We introduced participants to textual programing in DressCode by guiding them through writing their own random walk algorithms that produced patterns composed of lines. Following this, we explained the functionality of the graphic drawing tools, and had participants modify their random walk patterns by incorporating shapes they created graphically. The following session we discussed the general process of photo-emulsion screen printing and demonstrated several other forms of generative design. We provided them with images of sample designs and had them work in groups to write a set of instructions that they thought would re-produce that design. As a group we discussed their results, and then demonstrated examples in DressCode that would produce similar results to the sample designs (figure:6).

Participants were then provided with the design task of creating a computational design to screen print onto a t-shirt they would want to wear, and given 3 hours of open design time with the option of modifying the examples we provided, working off of the original random walk pattern they had created the first day, generating a design from scratch, or working with an instructor to come up with additional design ideas. The following session participants prepared their screens for screen printing, by stretching the screen material over the frames in groups and applying the photo emulsion to their screens. Afterwards, participants were given 2.5 hours of additional open time to complete their designs. Participants laser-printed their finished designs on transparencies, which the instructors exposed on the screens overnight (figure:7). The final session, participants practiced printing with their designs on test materials, and then printed onto their t-shirts.

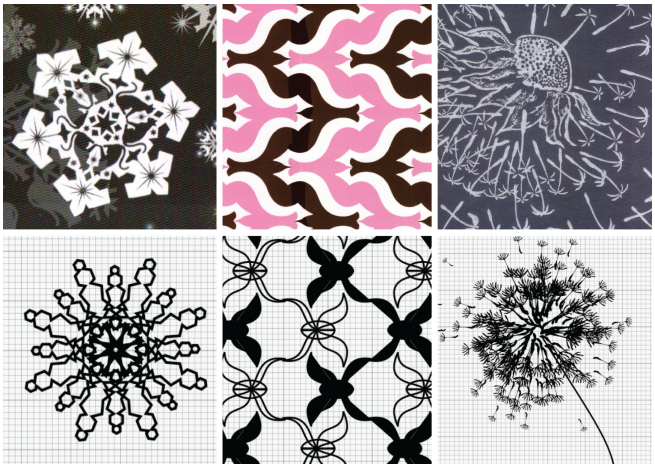

Figure 6. Sample designs and corresponding examples in DressCode

Figure 7. 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 primary workshop was successfully able to use DressCode to produce a design for their t-shirt (figure 8). 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. Discuss importance of booleans. The screen printing process was extremely popular among the participants. Each person was successful in creating their screen, transferring their design to the screen and printing to the shirt, although one participant had to re-expose his screen because of errors in applying the emulsion. 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 8. Completed shirts from screen printing workshop

From survey data and group discussions, 3 participants found the screen printing portion to be most difficult, 3 found the programing to be the most difficult, and one was equally challenged by both portions. An evaluation of the pre, mid and post-workshop surveys demonstrated that following the screen printing, participants attitudes towards programing, 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 programing in the future, a stronger belief that programing was a tool that they could create things they would use in their daily life, and greater comfort in programing on their own following the craft activity. Six of the eight participants stated that they liked their design better after printing it, and all participants stated that the process of screen printing had given them ideas for additional designs they would like to create with DressCode. For a minority of participants however, the difficulty of the craft portion seemed to result in a slightly diminished interest in combining craft and programing, and a diminished interest in craft in general. Participants were positive about the graphic tools, although they requested that their functionality be extended to incorporate a greater range of transformation methods (rotation, scaling, and boolean operations). All 8 participants said they would be interested in using DressCode for another activity, and the majority indicated that they would like to combine computational design with 3D printing, and 7 of 8 participants indicated they would like to continue using the programing language in particular.

DISCUSSION

In our discussion of the workshops we focus on three primary elements of the participants' experiences. First, we examine the successes and limits of the DressCode software, with an emphasis how people used the graphic drawing and manipulation functionality. Second, we evaluate participant's design practices and discuss how they connect to the affordances algorithmic craft. Finally we consider what participants got out of the workshop, by describing participant reactions to their

	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

finished artifacts and evaluating how these artifacts resonate with participants' personal values and daily lives.

Design Principles for Combining Graphic Manipulation with 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. The graphic drawing tools were used in the majority of participants' designs (table 2.

Participant responses on the survey and during discussion suggest the tools were accessible to use, and assisted many in understanding the functionality of their programs. 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:

> *I think that the way it already is, is pretty good. 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, like if you make a line from this point to this point to this point this is actually what the code is doing, so having a graphical side and having it auto update the code, and so it can show you that you want to work with the code you're learning as you're using the GUI too. 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 themselves. 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.

Beyond facilitating the design process, the introduction of the graphic tools also resulted in a discussion among participants in the primary workshop about future paradigms for graphic design and programing tools. This discussion pointed to specific elements in the graphic tools that participants found confusing or helpful. Participants were 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. Incorporation of more sophisticated drawing tools would provide the opportunity for further personalization of computational forms.

While the primitive drawing tools proved to be popular among the participants, several had difficulty with the move tool. From a development perspective, the move tool was challenging to implement because there were numerous possibilities on how it could be represented in the user's 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 tool behavior confusing because it modified an existing statement in their program, rather than generating a new line of code. This frequently resulted in syntax errors when the participant then modified shape statement and omitted a parentheses or placed arguments in the incorrect portion of the statement. These issues make it clear that we should re-consider the functionality of the tool, however they also convey a general 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. Furthermore, the fewer the assumptions, the easier the tool will be to implement. 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 M: *[I want] an erasing tool, so you can erase the line kind of like in Sketchup.* 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?* Participant M: *Or you could just drag the shapes and it just goes away.*

This conversation reinforces the difficulty in 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 how they would wanted to preserve static instances of randomly generated designs. Several participants used the static stamp tool for its intended purpose during the workshop, but others requested different forms of control over the randomly generated values. One participant wondered if they 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 and noise generation through a form of graphic input similar to the graphic dialog tools, or if they felt creating random methods in the programing 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 programing 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 turn things like random, these are drawn (indicating a hand drawn graphic) and these are from the programing part (indicating the random repetition of the graphic), but in Illustrator it's just drawing, you can't have everything.*
> 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 programing demonstrated several conclusions. Drawing tools proved to be largely intuitive and easy to use and they distinguished the participants' designs from one another, and from the work of participants in the prior workshop without the tools. The incorporation of personal drawing styles techniques is highly relevant to our objectives in algorithmic craft, because it allows people create artifacts that are personally unique, and conform to the stylistic preferences of the individual who produced them. Furthermore, the evaluation of the tools by the participants demonstrated that the tools were assisted in communicating basic computational principles as demonstrated 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 the process of programing.

Algorithmic Crafting in Practice

what did people actually end up doing? If the graphic tools assisted with some of the technical challenges of algorithmic craft, we are still left with the 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 our 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 programing.

> *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
>
> *I now know that there are so many more possibilities of programming because I've seen a connection.*
> Participant J
>
> *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 designing itself, 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 programing to design is a fundamental step to participating in algorithmic craft. It is more challenging however for people to understand the specific affordances of computational design and incorporate them in their practice. The participants in the screen printing workshop performed even better than we hoped in this respect. A combined evaluation of their end products, and statements from the discussions and surveys points 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 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 itself **??** revolves around the idea of the *visual complexity* that is enabled through computational iteration and repetition. Her design is also an example of *remixing*, as she combined portions of the

code from an example with a program she had written herself, making adjustments to both to produce a novel composition. Another participant incorporated aspects of a fellow participant's program into his own design by adding the code for the stars in the landscape design to his heart design. The landscape design serves as an example of the benefits of parameterization. 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. Because 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 the majority of participants designs. All but two of them incorporated some form of noise in their code. Below, a participant with no prior programing experience describes her experience in using uniform 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 was demonstrated the most elaborate use of generative design in the workshop. During the second critique, he presented 17 unique and complex patterns, solely comprised of overlapping lines. This astonished the other participants who were surprised to see that such variety could be produced by one algorithm. He described his process with DressCode in this way:

> [DressCode] allowed me to see random patterns such as the one I made. I may have been able to 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 to produce an emphasis in her composition. In attempting to do so however, she struggled in determining how to have her algorithm deviate from the gaussian distribution and 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 at arbitrary points are often unclear or tedious to implement. Another challenge in computational design involves the process of formalizing complex problems. As design problems grow in complexity, formalizing the problem in a manner that can be expressed programmatically becomes increasingly difficult. The participant who created the generative landscape pattern wished to modify his design so that the distribution of the mountains, clouds and stars appeared more "realistic". He had difficulty reconciling his intuitive understanding of the qualities realistic landscapes with a set of formal properties that could be communicated through programing. Lastly, the 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, individual 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 programing context and consisted of helping to correct syntax errors and pointing participants towards relevant programing 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 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 criticize the work of their peers or offer suggestions. Through successive critique sessions participants gradually became more comfortable in voicing their opinions. By the end of the workshop, several participants stated that the critiques had helped to improve their design. For example:

> 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 often simultaneously provided in the form of stylistic suggestions and technical tips on how modify the code of a design 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 used to create the pattern, and 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 on 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 20 design instances with the opportunity to receive group suggestions on what his final design should be, while simultaneously stimulating a discussion on the challenges of gen-

eratively 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's perceptions of the creative affordances of programing 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 on the advantages and challenges of computational design in general.

Physical and Digital Notions of Value

what mattered to people in the end? This section is not done. Participants varied in both workshops on the elements of algorithmic craft they found to be the most difficult, as well as the parts they found most enjoyable. People in the bracelet workshop were unanimous in their identification of the programing component as the component that they found most frustrating.

In comparison, the screen printing participants also found aspects of the programing challenging, but expressed a greater variety of between what they found to be difficult. Two people said the actual process of coding was most difficult, and another said that the process of understanding the concept behind her design was challenging. The remainder of the participants identified aspects of screen printing as the most challenging portion, such as preparing the screen, avoiding alignment issues, or printing without error.

Screen printing, the first print had too little ink, the next one too much, and the third one came out and that was on my t-shirt so that was good. Miriam: 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. But yeah, I had to make a lot of copies. Beckett: I enjoyed the design/ creation process, but the printing itself? no Roger: 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

Added value as the result of physical labor. Does the difficulty of screen printing change how you feel about the piece Roger: It lets you take pride in it. Luisa: After all this hard work, you have this

Is that similar to the difficulty of programing I feel as though maybe like when you're programing 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 like hard work, well you do put hard work into programing, but it's more a thinking thing, besides actual physical labor.

subsubsectionCraft and Computational Compatibility All participants felt computational and screen printing worked well together All participants said that after screen printing they were more interested in computational design All participants stated that they would like to use computational design for screen printing in the future. All participants indicated that screen printing gave them additional ideas for things they would like to create with DressCode All participants felt DressCode worked well with screen printing Consideration of materials in design

Personal Accomplishment Uniqueness

Personal and Cultural Relevance in Algorithmic Craft
cultural appropriateness of aesthetics (gender, youth)

entrepreneurship Open question of what the resultant product is. (screen? artifact? program?) Community Setting

> *I enjoyed the people who participated in the program because they make the whole process really fun. I love the environment we worked in, everything around us is so fascinating.*
> Participant J

Isolation

> *Something I found really challenging was that many of the other participants already have lots more experience in this field than me so I kind of feel isolated and behind everyone else.*
> Participant J

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 accessible to new programmers by providing programming examples that suggest design approaches or by creating built in functionality with compelling aesthetic potential.

- **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 is a creative practice, not a technical exercise, and as such, it benefits from discussion among peers, critique, and reflection. With novice programmers, engagement in these reflective practices is best conducted in a group setting with some form

of facilitation. Group algorithmic crafting activities can also provide a casual social atmosphere that is conducive to hand-craft.

- **Materials matter.** The use of rich, interesting materials greatly enriches the experience of algorithmic craft and contributes to the success of the finished artifacts. Raw materials like leather, wood, textiles, and art-quality paper complement the precision and complexity of computational design and digital fabrication. Combined in the hands of an engaged user, they can result in objects that are attractive and durable. Care should also be taken to demonstrate 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, activities should aim to be mindful of the gender, age, cultural background, and social norms for participants. It is useful to engage with participants in a dialog about these topics and how they could be further supported in future algorithmic crafting experiences.

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. 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).
4. Johnson, G. Flatcad and flatlang: Kits by code. In *VL/HCC, IEEE,* (Herrsching am Ammersee, Germany, 2008).
5. Johnson, G. Sketch it, make it. `http://sketchitmakeit.com.`, 2012. (Accessed: 06/3/2013).
6. McCullough, M. *Abstracting Craft: The Practiced Digital Hand*. The MIT Press, Cambridge, Massachusetts, 1996.
7. Processing. `http://www.processing.cc.`, 2001. (Accessed: 06/3/2013).
8. Reas, N., McWilliams, C., and LUST. *Form and Code: In Design, Art and Architecture, A Guide to Computational Aesthetics*. Princeton Architectural Press, 2010.
9. 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).
10. 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).
11. Sennett, R. *The Craftsman*. Yale University Press, New Haven and London, 2008.
12. Zoran, A., and Buechley, L. Hybrid reassemblage: An exploration of craft, digital fabrication and artifact uniqueness. In *LEONARDO*, vol. 46 (2013).