DressCode: Subtitle to come

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

2nd 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

A common objective of computer-science education is to prepare young people for*future* careers in technology. We recognize the value of computer programming for young people, not as a skill for future employment, but rather as a powerful creative tool for personal expression. Furthermore, we believe that programing tools and activities that are motivated solely by concerns of professional development may actively hinder diverse participation in computer science. We draw upon Dewey's observation that educational activities which are oriented towards future responsibilities can hinder active participation by students in the present [9]. Furthermore, we recognize that limited participation in STEM fields by women and minorities is partly due to the perception that STEM applications themselves are constrained in cultural and intellectual diversity and breadth [8]. Our goal is therefore to engage young people in creative applications of programing by providing pathways towards forms of making that are relevant and appealing to their *present* lives and interests.

In pursuit of this goal, we have investigated the development of novice-oriented software tools within the field of computational design. Computational design affords a rich set of creative opportunities which are directly applicable to a wide range of forms of making, aesthetic tastes, and creative pursuits. In a further effort to situate computational design in a relevant context, we developed activities which apply our tools to the design and creation of physical and functional crafts which connect to forms of making and artifacts which are both familiar and desirable to groups of young people who are traditionally underrepresented in computer science.

The introduction is still pretty weak, but hopefully will be more evident how to flesh it out after finishing the rest of the paper This paper describes the development of the novice-oriented computational design software, DressCode. DressCode was developed to employ *linked editable representations* between programing code and graphic drawing and manipulation in an effort to support young people in the independent and flexible use of programing to achieve their personal design objectives. We were interested in understanding how a computational design tool could realistically be incorporated in the daily lives of young people. We therefore developed DressCode through an iterative process wherein successive prototypes were evaluated through series of workshops where middle and high-school students used DressCode to design and create their own fashion accessories and clothing through a combination of computational design and hand-craft. In the following section of this paper, we describe the specific creative accordances of computational design, and related work in the field of computational design software. We follow with a description of the DressCode software and an explanation of the values and principles that directed our design process. We describe the iterative design and evaluation of DressCode through a set of workshops with young people, and conclude with a discussion of these experiences and how they reflected our progress with our initial design principles and objectives. Throughout our research we seek to address key questions relating to youth participation in computation: What are important design principles to consider when developing computational design tools for young people? What methods are helpful in ensuring accessibility, creative flexibility, and support of computational affordances? How can both the software and the activity structure reduce technical and conceptual challenges in translating a computational design into a viable physical artifact? Lastly, what values are reflected in this form of production, what range of young people are engaged by this form of making, and how are the resulting artifacts applicable to their lives? How does situating computational in contexts that emphasize pleasure and personal relevance shift perspectives on computing in general?

CREATIVE AFFORDANCES OF COMPUTATIONAL DESIGN

Papert and collaborators recognized early on that computer programing can serve as a way for children to learn powerful ideas about mathematics, geometry and logic [16]. In computational design, programming allows people to practically apply math and logic as mechanisms for composition. In addition, computational design enables people to approach design through *procedural thinking*. In a general description of computational design, the designer author rules that define a system capable of producing many design outcomes, enabling the production of multiple design variations that share a set of constraints. Through this systematic abstraction, the following affordances are present[19]:

- **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.

These affordances are closely connected to the advantages computation offers for a wide range of disciplines. Our goal is to effectively demonstrate that although complexity, parameterization and precision are useful for science and engineering tasks, they are just as well suited for aesthetic, artistic and personal pursuits; for example producing a random distribution of hundreds of unique flowers in an illustration, or parametrically defining a dress pattern that can be customized to fit a variety of people.

Challenges in Access and Application

Although amateur use of computational design growing, it remains largely inaccessible to individuals who are new to programing. This is in part due to practical and technical barriers. 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 [21]. There also are perceptions of technology in general which may engagement by people who are interested in making. Digital technology is often portrayed as reducing the need or desire for manual human labor, rather than supporting it in creative forms [6].

RELATED WORK

This entire section needs to be shortened In the development of DressCode, we examined several fields of related software tools including learning-oriented programing environments, Computer Aided Design (CAD) tools with computational design functionality, and digital-design tools with linked forms of editing. Moreover, we situated our development of DressCode within the context of prior research in combining craft and computational design for young people.

Learning Oriented Programing Environments

There are numerous examples of programing environments for helping young people learn to program. Here we list examples which are most pertinent and inspirational for our research. Logo is the seminal novice programming language founded on principles of constructionism and embodiment [16]. The Scratch programing environment was developed as a successor to Logo. Scratch is one of the most successful modern novice-oriented programming environments, and allows users to create interactive projects by combining command blocks [20]. Processing is a language and development environment for creating complex forms and animations. Described as an entry level programming environment, Processing has also become a successful professional computational design tool[18]. 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. In addition, each of these examples are designed to helping people to apply programing to creative practices, rather than engage people in learning programing for its own sake.

CAD tools with Computational Design Functionality

Computational design can be considered as a subset of computer-aided design (CAD). Although computational design can be technically performed with any programming language that has the capability to output some form of graphic data, a number of CAD software tools have been developed that incorporate specific tools to expressly enable computational design. Most are designed for professional users. Adobe software like Photoshop and Illustrator and 3D modeling tools such as Maya and Blender feature the ability to script behaviors in languages that are syntactically similar to Javascript, Perl and Python, respectively. In all of these examples the programming interface is omitted from the primary interfaces of the application. Other CAD-computational design tools feature a direct emphasis on programing in their interface. Grasshopper is data-flow programming environment that enables users to combine a variety of visual modules and blocks to create and adjust 3D models in Rhino [**?**]. DesignScript is an add-on to the Autodesk AutoCad software with an ever-present text editor, allowing users to script 3D architectural forms through a combination of associative and imperative programing paradigms [3]. OpenSCAD is a script-based constructive solid-geometry modeling tool developed specifically for CAD applications. Design actions in OpenSCAD are conducted exclusively by programing textual descriptions of 3D models in a declarative format [1].Although these examples offer powerful tools for design, we argue that they are not suitable for novice use, both in terms of difficulty and cost.

Novice-oriented CAD tools with computational design features are more rare than professional tools. FlatCAD is a domain-specific tool allows users to design customized gear-based construction kits by programming in FlatLang, a novice-oriented programming language modeled on Logo [12]. TinkerCad is 3D-modeling tool designed for entry level users to create forms for 3D printing. TinkerCAD recently added ”shape scripts”, functionality enabling the creation of Javascript programs that produce 3D forms [24]. Autodesk’s 123D tools consist of a variety of novice oriented CAD applications that support the creation of designs for 3D printing and laser cutting [4], but only minimally enable computational design by allowing users to automate the repetition of elements in predefined patterns. All of these examples emphasize physical forms of making through digital fabrication. It is also encouraging however, that commercially developed tools like TinkerCad and 123D are beginning to explore computational design for novices, albeit in forms that are still secondary to graphic manipulation.

Linked Editors

Educational researchers have found that multiple linked representations are advantageous in supporting learning [26]. When applied to the appropriate context, multiple external representations can reduce the amount of cognitive effort required to solve a problem, and often better communicate complex content [2]. Linked representations have been applied to digital design tools in a number of applications, and with mixed results. Avrahami, Brooks and Brown demonstrated an early approach to a two-view system for designing user interfaces by combining a graphic editor with direct manipulation of interface elements with a special-purpose programming language. Interface designs can be edited in either view, and the changes are reflected in the other view [5]. Commercially, the two-view approach has been incorporated primarily into web editors and GUI design tools in software development kits with varied success [13]. Victor has convincingly advocated for the incorporation of a of forms of linked representations in other design fields including circuit design, game development and graphic design. His approach has been to advocate through the development of demonstrative examples rather than functioning tools [25].

Craft and Computing Research

Eisenberg and Buechley’s research on pervasive fabrication provides a survey of approaches and benefits in combining computational design and physical making for young people. Their research encompasses techniques and approaches for incorporating digital fabrication into educational settings, and describes how computational forms realized through digital fabrication enable youth to decorate their environments, allows them to create novel artifacts in the service of personal expression [10].

DressCode also builds on research by Jacobs and Buechley in combining computational design and fashion. Codeable Objects is a Processing-based programing library developed by Jacobs to facilitate a series of workshops where young people produced fashion accessories and clothing by creating designs computationally and realizing them through a combination of digital fabrication and manual sewing [11]. Our focus on computational design and craft is motivated by the rich engagement opportunities illustrated by Codeable Objects research. In creating DressCode however, we are seek to develop a stand-alone design tool for novice computational design that combines textual programing with graphic drawing and manipulation, thereby addressing the issues Jacobs and Buechley describe in their use of Processing with novice coders. These issues include difficulties learning programing syntax, a frustration with the lack of visual feedback, and the need for frequent instructor assistance in technical programing tasks.

DRESS CODE SOFTWARE DESCRIPTION

DressCode is a 2D CAD tool with computational design tool that supports *linked editable representations* of a design in two forms: programmatic and graphic. As designers create and manipulate shapes graphically, the software generates readable, editable programming expressions that correspond to their actions. Designers can also generate graphic forms by writing programming expressions and then manipulate these forms with the graphic editing tools. We theorized that in the hands of young people, a computational design tool featuring interconnected graphic and programmatic interactions would allow for intuitive engagement, open ended forms of design, and diverse appeal for a variety of design approaches and skill sets. prior sentence is very important, but also very weak. Here, we describe the interface and tool-set of DressCode and the interactions between the programing and graphic portions of the tool.

System Overview

The DressCode application resembles a mash-up of a digital graphic drawing tool and a text editor. The interface is divided into two primary panels: a graphic panel on the right and a text editor panel on the left (figure:1). A designer may write and execute programs using the text editor; or they can draw and transform shapes using the mouse in the graphic panel. Each panel contains specific features and tools to enable these respective interactions. The text editor contains a console for output and error messages and a panel with buttons for compiling and running the current program. The design panel contains a re-sizable drawing board and grid with rulers corresponding to physical units (inches and millimeters), and pan and zoom tools to facilitate navigation. A toolbar on the right-hand side of the graphic panel contains a menu of drawing and transformation tools as well as contains a print button which allowing one to export their current visual design in vector format for output through printing, or 2-axis forms of digital fabrication (figure:2).

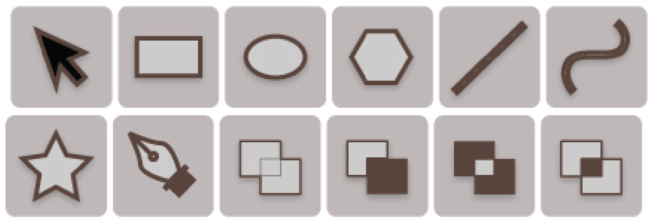
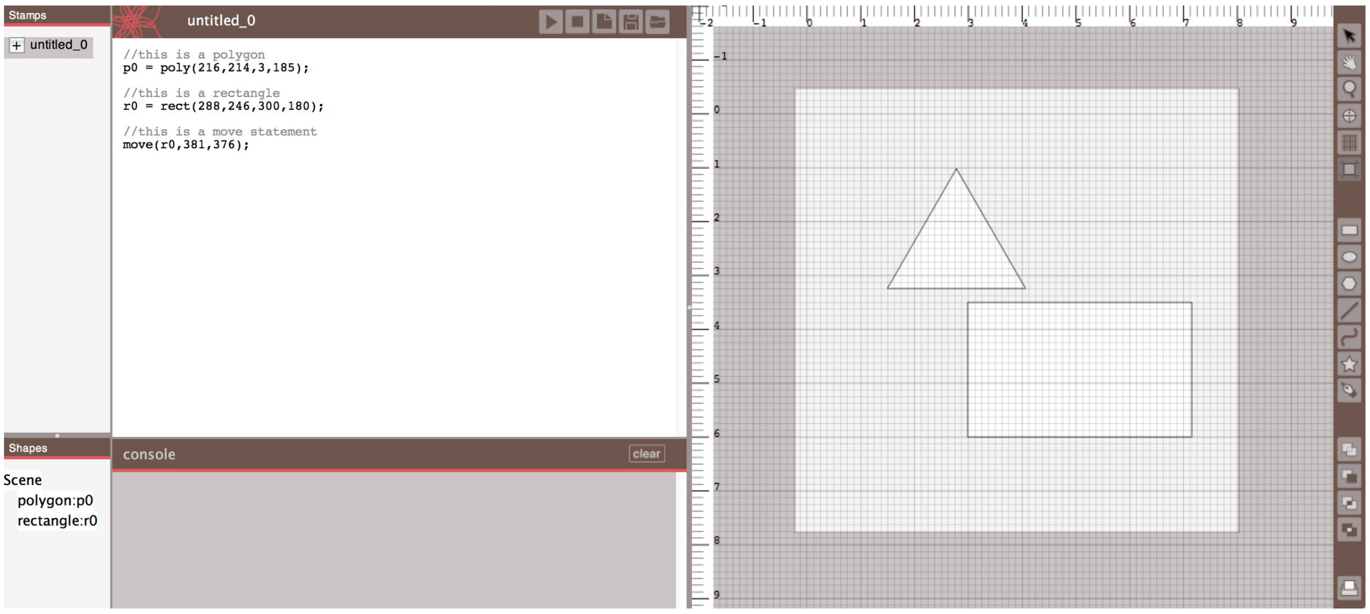


Figure 2. The graphic drawing and transformation tools in DressCode (from left to right: selection and move tool, rectangle tool, ellipse tool, regular polygon tool, line tool, curve tool, SVG import tool, pen tool)

The graphic drawing tools include regular shape creation tools and a pen tool for enabling the free-hand drawing of of



```
//this is a polygon
p0 = poly(216,214,3,185);

//this is a rectangle
r0 = rect(288,246,300,180);

//this is a move statement
move(r0,381,376);
```

Figure 1. The DressCode Software)

irregular shapes. In addition to the drawing tools, the selection tool allows for individual shapes and groups to be manually selected and moved, and the boolean operation tools allow for the combination of two or more shapes into a unified form through a variety of polygon-boolean operators (union, difference, intersection and either/or). The interface also contains two additional panels, the stamp panel and the declarative view which are described in the symmetry and readability subsections below.

DressCode contains a custom imperative programing language which supports conventional programming data-types, loops, conditional expressions and user-defined functions. Variables in DressCode are dynamically typed and identifiers can be assigned to data-types that differ from their original assignment.The language contains a subset of expressions which facilitate the drawing and transformation of 2D graphic geometric forms add figure here to show syntax. The language also supports math expressions and enables a variety of random noise generation methods. We describe the drawing API of the DressCode programing language and their relationship with the graphic manipulation tools at length in the following section. A note on terminology: for the remainder of this paper, we denote actions made in the graphic panel with the mouse as *graphic actions*, and actions made in the programing panel by typing expressions as *programmatic actions*. We also distinguish between two types of actions: *initialization actions* denote programmatic or graphic actions which result in the creation of a new shape and *transformation actions* denote programmatic or graphic actions which result in an existing shape being moved, rotated, scaled or otherwise altered.

Linked Representations In Practice

Graphic tools and programing tools have separate affordances. Graphic tools are based on metaphors of physical drawing and manipulation with traditional media, and often represented by icons and symbols that reflect this. As a result, they can be intuitive or inviting to new users. Textual programing tools are traditionally less intuitive, but have the advantage of readily supporting complexity and automation in design, and are a natural fit for parametric representations. Because of these differences, combining computational and graphic design into a unified interaction raises many questions around interaction design: What forms of interaction best support the distinct affordances of graphic manipulation and programing? How should graphical organizational structures be reconciled with computational forms of organization? What rules dictate the resolution of inevitable conflicts between graphical and programmatic actions? How does the intended application of the software dictate the relationships between the two paradigms? To approach these to these questions, we developed the linked representations in DressCode around two primary design principles: Symmetry and Readability.We describe the latest state of the tool with respect to these principles. We wish to emphasize however, that this state was achieved though a series of tests with successive versions in our workshops. The workshops not only directed the iterations of the tool itself, but assisted in the clarification of the design principles behind it.

Symmetry

The governing design principle in DressCode is symmetry between programmatic actions and graphic actions. For every shape that is initialized programmatically, a graphic element is generated in the graphic view. Conversely, for each graphic action, a corresponding programing expression appears in the text editor. The DressCode programing language syntax was developed to support the translation between graphic and programmatic representations. The drawing API is formulated on an Object Oriented Programming paradigm where basic shapes (points, lines, curves, polygons etc.) are initialized by calling the appropriate method and passing it a set of parameters designating its location and dimensions. If a shape is initialized graphically, its parameters are determined by the mouse gestures of the designer (where they click to determine the origin, and how far they drag from the origin to determine the dimensions). The method-type of the auto-generated expression that results from a graphic action is determined by the type of graphic tool that was used to create the shape, and the parameters determined by the graphically defined dimensions. Shapes are layered in a design in order of the designer's graphic and programmatic actions, with shapes that were drawn most recently appearing on top of those created earlier.

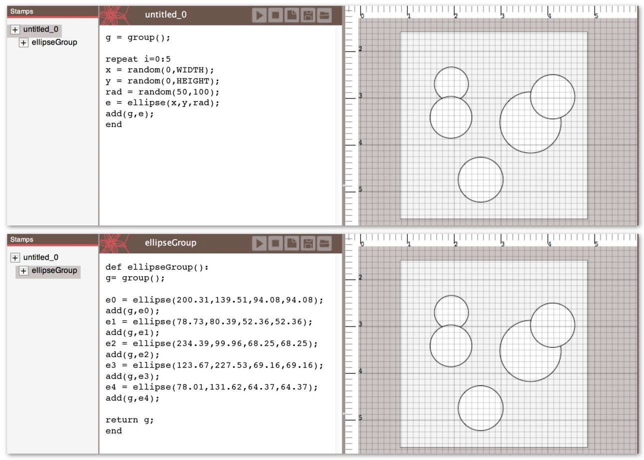
Transformation methods, including moving, scaling, rotation, color and stroke changes, and shape booleans follow a similar structure to shape initialization. In the programing language, transformations are performed by either wrapping a shape-initialization expression in an a transformation expression, or by assigning an identifier to the shape, and then calling the transformation method with the identifier. Each graphic transformation tool corresponds to a transformation method in the DressCode language, enabling the auto-generation of an expression in the programing panel which contains as its first argument a reference to the shape which was selected and manipulated graphically. Throughout the design process, a complete representation of the current state of the graphic design is continually maintained in the programming panel. This representation allows programs to be shared and remixed easily; if the textual program from one design is copied and inserted into another designs' program, it will re-generate the exact design in the context of the new design.

Static and Dynamic Generativity

Organizational structures are an essential component of CAD as a design grows in complexity. need to re-work this section so it fits in with symmetry, or move it somewhere else 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 users to save a specific instances of a generative design (see Figure 3). 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.

Readability

As we discovered through our implementation, Symmetry between graphic manipulation and textual programing must be tempered by concerns of usability. Merely producing a textual expression that accurately reflects a graphic action does



```
g = group();

repeat i=0:5
x = random(0,WIDTH);
y = random(0,HEIGHT);
rad = random(50,100);
e = ellipse(x,y,rad);
add(g,e);
end
```

```
def ellipseGroup():
g= group();

e0 = ellipse(200.31,139.51,94.08,94.08);
add(g,e0);
e1 = ellipse(78.73,80.39,52.36,52.36);
add(g,e1);
e2 = ellipse(234.39,99.96,68.25,68.25);
add(g,e2);
e3 = ellipse(123.67,227.53,69.16,69.16);
add(g,e3);
e4 = ellipse(78.01,131.62,64.37,64.37);
add(g,e4);

return g;
end
```

Figure 3. 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.)

not ensure that the interaction will be interpretable to the designer, let alone useful to their design process. We therefore considered how we could design linkages between programmatic and graphic actions which were not only functional, but transparent, readable, and reconfigurable by young people.

Readable Edits

First, we considered *where* auto-generated expressions would appear within a program. For an initialization graphic action the programming expression will always appear below the last line in the program. If the designer performs a transformation graphic action however, the expression will be inserted into the line below the initialization of the selected shape, or below the last transformation expression for the shape. This structure ensures that the modified program will reproduce the correct order of operations when run, but also provides a form of organization to auto-generated statements. Naturally, in the process of manually writing code, the designer may consciously or unconsciously write expressions in a manner that deviates from this organization. Fortunately, manual edits by the designer will not prevent the ordering mechanism from functioning for successive graphic actions. More importantly, the consistent, simple rule-set for auto-insertion enables the designer to anticipate where expressions will be inserted into their code, which is essential to allow them to make edits as a program grows in length and complexity.

Readable References

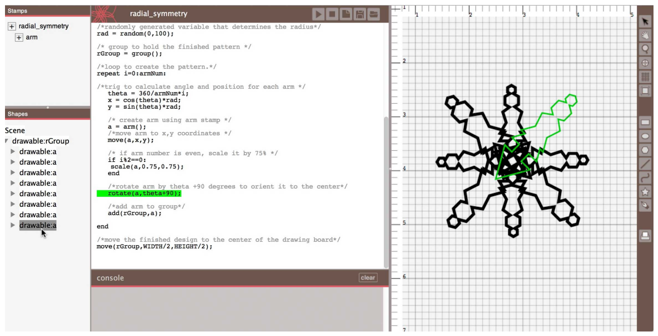
Like many existing programing languages, DressCode gives the developer flexibility in how they reference program are referenced. The DressCode language enables methods to be nested within one-another, so there is a degree of ambiguity for how a designer may use identifiers in their code. As a result, it was necessary to evaluate *how* shapes should be referenced when transitioning between programmatic and graphic representations. Here we also advocated for an approach that would make the design ”readable” for new programmers. All initialization graphic actions produce programmatic expressions that are automatically assigned an identifier. Subsequent transformation graphic actions will produce programmatic expressions that reference the auto generated identifier on the following line. This produces code that while requiring more lines, is arguably more readable than a nested chain of expressions. If a shape that was programatically initialized without an identifier is transformed graphically, DressCode will recognize the distinction and resort to wrapping the initialization expression in the appropriate transformation expression. If the designer re-assigns or modifies the identifier of a shape through a programmatic action, future graphic actions on the shape will recognize this modification and use the new identifier.

Readable Design Steps

It was important to us that the steps used to arrive at a design were also readable. Because the DressCode programing language employs an imperative paradigm, designs are represented as a series of the designer's actions, rather than a declarative state. We structured the auto-generated statements so that the programmatic expressions reflected the order of steps a designer made in the graphic interface. DressCode takes a verbose approach to generating programming expressions. For example, when a shape is moved with the move tool, a textual move expression is inserted into the designer's program. For all subsequent moves following the first move, rather than generate a successive string of additional move statements, the move expression is updated to reflect the new coordinates of the shape. However, if another tool is used to alter the shape, or a programmatic expression is manually inserted by the designer following the move statement, future actions with the move tool on the same shape will generate a new move statement, which will be subsequently updated until another tool is used (see Figure **??**). This same logic applies to all other transformation tools in DressCode. We chose this structure, because it allows the program to preserve the designer's actions in the graphic panel as a series of discrete steps, thereby providing an opportunity for the program to also serve as a means of reflection and evaluation on one's practice and design intentions I feel like this section is maybe repetitive of what I say in the symmetry section.

Declarative Representations

Although DressCode relies on an imperative paradigm, we recognized that a state-based representation could provide a useful means of illuminating the connections between the visual design and the textual program. Specifically, it was important to help a designer determine what code in their program was affecting specific shape in their design. Contemporary programing environments now frequently employ debug tools that allow the user to step through their code in an effort to decipher its effects. Similarly many code editors contain declarative listings of all of the methods and identifiers in a class, providing an alternative means of navigating one's code. We combined these tools in a way that is specific to a tool for programing and graphic drawing. DressCode features a declarative view which 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 4). The declarative view provides visual feedback on how elements of a design connect to the program, and provide a practical selection technique for complex designs.



```
/*randomly generated variable that determines the radius*/
rad = random(0,100);

/* group to hold the finished pattern */
rGroup = group();

/*loop to create the pattern.*/
repeat i=0:armNum:

/*trig to calculate angle and position for each arm */
   theta = 360/armNum*i;
   x = cos(theta)*rad;
   y = sin(theta)*rad;

   /* create arm using arm stamp */
   a = arm();
   /*move arm to x,y coordinates */
   move(a,x,y);

   /* if arm number is even, scale it by 75% */
   if i%2==0:
    scale(a,0.75,0.75);
   end

   /*rotate arm by theta +90 degrees to orient it to the center*/
   rotate(a,theta+90);

   /*add arm to group*/
   add(rGroup,a);

end

/*move the finished design to the center of the drawing board*/
move(rGroup,WIDTH/2,HEIGHT/2);
```

Figure 4. Declarative view with selected primitive

DESIGN PROCESS

Technology use in the real world can be understood from an ecological perspective, where people's activities and tools adjust and are adjusted to each other in a complex system of interactions between norms, technical functionality and cultural values [15]. We developed DressCode with the objective of better understanding how a computational design tool could be effectively situated within the everyday lives of young people. In addition, we wished to incorporate a specific set values into our tool design which are frequently absent from mainstream technological use. It was necessary therefore that our design process and evaluation methods allowed us to understand the context in which our tool might be applied in the real world, and what values were reflected by this use. Our design process was to develop DressCode in an iterative fashion by testing successive versions in workshops with young people where they used the tool in the context of hand-craft applications to produce a finished artifact for their personal use. Here we elaborate on the values and specific evaluation behind DressCode, demonstrate the rationale for applying DressCode to craft, and describe our workshop methodology.

Values and Evaluation Criteria

Considering shortening these to just a set of evaluation criteria in the form of questions for each subsection. Not sure values makes sense here.. By evaluating prior research in youth engagement in computational tools [7], [17], in conjunction to our own research motivations described in the introduction, we outlined set of value categories for DressCode, and corresponding evaluation critera.

Accessibility

As is perhaps self-evident when developing tools for novices, accessibility is a primary concern. To evaluate the accessibility of DressCode, we were concerned with the following questions: Could young programmers use the tool with a high degree of independence? What forms of experimentation and exploration were demonstrated? Were components of the tool intuitive to use? What level of confidence did people express in their use of the tool?

Diversity

Whether performed by experts or novices, individual approaches to design are incredibly subjective and diverse [22]. It is essential then that computational design tools afford a diversity of practices. Drawing from Brennan and Resnick, we wish to value diverse forms of computational practice. Are participants able to meaningfully apply computational constructs to their design objectives? Are they able to remix other people's designs? Are they able to critique other's designs and offer technical and aesthetic advice [7]? Does use of the tool produce a wide variety of end products? Beyond a diversity of design practices, what forms of diversity are exhibited by the people who are interested in using the tool?

Relevancy

Despite our belief that computation can serve as a valuable tool for personal expression, people who are new to computer science often view computer programing as irrelevant to their personal interests [17]. It is important that our approach with DressCode allows for a reconciliation between computational practices and the values and goals of young people. We wonder how artifacts created with DressCode may be used in daily lives of the people who made them. How long are they kept and cared for by their makers? Further, how might the use of DressCode alter perspectives towards the creative applications of programing?

Engagement

This section is weak In developing our tool, we were concerned with specific behaviors that we believe could be used as indicators of a deep engagement in the creative application of computational tools. Do users of the tool demonstrate motivation to take on difficult tasks? Do participants take pride in their artifacts and exhibit care and attention to detail in their creation? Do people express an interest in future participation in similar activities?

Social Connections

A common perspective of computer programing is that it is an asocial and solitary activity [17]. Because we recognize that social engagement is a core component of a young person's life, as well as a necessary element of creativity, we seek to promote forms of computational engagement that emphasize social connections, and evaluate DressCode by how it enables people to work together. In particular, we wish to foster forms of creation where the process of creating with others is a central component of an activity, leading to forms of social interaction with peers which are both pleasurable and productive.

Craft Application

this section is just an outline Craft cosely connects with the values we are interested in promoting in technological use. Craft is social- cite rosner, craft can produce artifacts that are relevant to young people-cite buechely, craft is aesthetically and creatively appealing-cite buechley again, craft is diverse in forms of engagement- craftsmanship, intellectual engagement, pleasure through physical labor-cite David Pye, Sennett and McCullogh. We applied DressCode to craft for these reasons.

WORKSHOPS

taken from prior paper, need to condense and make specific to this version. We evaluated DressCode with young people through two workshops. Both workshops focused on using DressCode to produce wearable artifacts. The preliminary study consisted of a one-day workshop in which participants designed and fabricated leather wrist-cuffs. The long-term evaluation was conducted through a four-day workshop, divided between two consecutive weekends, in which participants created computational designs and screen-printed them onto t-shirts.

Evaluation Methods

The participants' experiences were evaluated through written surveys, interviews and group discussions. The surveys were aimed at understanding participants prior attitudes towards programming, design and craft, their interest in and attitudes toward programming and design, and their engagement in and enjoyment of the workshops. 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 of 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. In addition, post surveys contained a range of written questions asking the participants to describe their design process, their opinion of the success of their projects and their experience using DressCode. Interim surveys were also administered during the primary workshop 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.

In-person interviews were conducted with the participants in the preliminary workshop. These interviews lasted an average of 10-20 minutes and were audio recorded and transcribed. During the interviews, the participants were asked to describe their artifact, and talk about the process of conceptualizing, designing, and producing it. In the primary study, participants were engaged in three group discussions 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 interview and discussion responses and project outcomes were analyzed to determine how they connected to the primary research questions we associated with algorithmic craft in the introduction. We also used this information to identify repeated and notable elements of participants experiences, and formulate a set of criteria for future development in algorithmic craft.

Preliminary Study: Wrist cuffs

The wrist cuff workshop was conducted among 10 young adults, aged 15-17, 20% male and 80% female. Of those surveyed, three participants had prior experience with Scratch, and two had worked briefly with the Arduino programming environment in a prior workshop. All participants said they had some prior experience in art, design, or craft. Prior to the workshop, 60% of the participants indicated that they did not feel comfortable programming on their own; however, the majority indicated they were interested in learning more about the process.

During the workshop, participants used DressCode to design a pattern for a wrist cuff, laser-cut their patterns into leather and assembled the finished wrist cuff by hand. The computational design in the preliminary study was structured around radial symmetry, and the majority of the resulting artifacts featured patterns that were derived from radially symmetric structures. Participants wrote their own radial symmetry algorithms, and were then provided with a template in DressCode that automatically clipped their designs to the dimensions of a cuff sized to their wrist. Participants were given two hours to design, and 3 hours to fabricate and assemble their piece.

Preliminary Results

Each participant in the preliminary workshop was successfully able to use DressCode to produce a unique leather cuff. The design approaches and aesthetics of participants varied greatly. Some cuffs were geometric, some were floral, and many styles were difficult to link back to the original radial algorithm. Each participant was able to write their own radial symmetry algorithm in the initial lesson and produce a variety of patterns with it. During the design portion, several participants extended the radial symmetry algorithm with additional design elements, however most participants designs relied exclusively on structures that were possible with radial symmetry. Eight of the participants said they planned to wear the bracelets they created, and two indicated that they planned to give them as gifts. A comparison of the pre-and post-workshop surveys demonstrated increased comfort with programming as a result of the activity. After the workshop, more participants indicated that they thought programming was a good tool for personal expression and design. All participants stated that they believed they could use programming to make things that were beautiful, whereas several had disagreed with this statement before the workshop. All participants unanimously identified the programing portion as the most difficult component of the workshop, however the majority said they were interested in continuing to learn more about computational design for craft applications.

Design Revisions

Here is how we changed DressCode between workshops

Primary Study: Screen Printing

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 for the workshop were deliberately selected to represent a range of programing and design experience. Two participants, (one being the oldest participant), were relatively experienced programmers, having worked extensively with java, python and lua. Two participants considered themselves intermediate to novice programmers with basic experience in Scratch, HTML, Python or App Inventor. Two other participants had prior exposure to Scratch, but had not worked with it extensively. One participant had no programing experience, but was interested in learning. 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. The majority of participants indicated they had a variety of prior crafting experience, including making jewelry, basic woodworking, sculpture and origami. One person had screen printed before using vinyl cut stencils. Interestingly, one participant also identified soldering a electronic synthesizer kit as a craft activity. Only one participant stated that they had no prior craft experience.

Because the screen printing workshop occurred over a four-day period, it provided the opportunity to explore the computational design process in DressCode in greater depth, and allowed for the participants to engage in a more complex craft process: photo-emulsion based screen printing. 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 screen printing and demonstrated several other forms of generative design first 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:5). 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:6). The final session, participants practiced printing with their designs on test materials, and then printed onto their t-shirts.

Figure 5. Sample Designs and corresponding examples in DressCode

Figure 6. Screen printing process

Screen Printing Results

Each participant in the primary workshop was successfully able to use DressCode to produce a design for their t-shirt. 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 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.

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.

DISCUSSION

textcolorredreturn to evaluation criteria from design section, and evaluate how workshops, tool correspond to this criteria

Accessibility

Diversity

Relevancy

Engagement

Social Connections

CONCLUSION

REFERENCES

1. About, O. `http://www.openscad.org/about.html`, 2013. (Accessed: 06/3/2013).
2. Ainsworth, S. Deft: A conceptual framework for learning with multiple representations. *Learning and Instruction* (2006).
3. Aish, R. *DesignScript: origins, explanation, illustration.* Springer, 2012.
4. AutoDesk. 123d. `http://www.123dapp.com/`, 2013. (Accessed: 06/21/2013).
5. Avrahami, G., Brooks, K. P., and Brown, M. H. A two-view approach to constructing user interfaces. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, ACM (New York, NY, USA, 1989).
6. Bean, J., and Rosner, D. Old hat: Craft versus design. In *Interactions Magazine* (2012).
7. 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).
8. Buechley, L., and Hill, B. Lilypad in the wild: How hardware's long tail is supporting new engineering and design communities. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, DIS '10 (New York, NY, USA, 2010), 199–207.
9. Dewey, J. *Experience and Education.* Touchstone, New York, 1938.
10. Eisenberg, M., and L., B. Pervasive fabrication: Making construction ubiquitous in education. In *Computing and Communications Workshops, IEEE* (White Plains, New York, 2007).
11. Jacobs, J., and Buechley, L. Codeable objects: Computational design and digital fabrication for novice programmers. In *CHI, ACM,* (Paris, France, 2013).

12. Johnson, G. Flatcad and flatlang: Kits by code. In *VL/HCC, IEEE,* (Herrsching am Ammersee, Germany, 2008).

13. Li, P., and Wohlstadter, E. View-based maintenance of graphical user interfaces. In *Proceedings of the 7th International Conference on Aspect-oriented Software Development*, AOSD '08, ACM (New York, NY, USA, 2008).

14. McCullough, M. *Abstracting Craft: The Practiced Digital Hand*. The MIT Press, Cambridge, Massachusetts, 1996.

15. Nardi, B., and O'Day, V. *Information ecologies: using technology with heart*. MIT Press, 1999.

16. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas.* BasicBooks, New York, 1980.

17. Porter, L., Guzdial, M., McDowell, C., and Simon, B. Success in introductory programming: what works?

18. Processing. `http://www.processing.cc.`, 2001. (Accessed: 06/3/2013).

19. Reas, N., McWilliams, C., and LUST. *Form and Code: In Design, Art and Architecture, A Guide to Computational Aesthetics*. Princeton Architectural Press, 2010.

20. Resnick, M., Maloney, J., Monroy-Hernndez, A., Rusk, N., et al. Scratch: programming for all. In *Communications of the ACM* (2009).

21. 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).

22. Rivard, K., and Faste, H. How learning works in design education: Educating for creative awareness through formative reflexivity. In *Proceedings of the Designing Interactive Systems Conference*, DIS '12, ACM (New York, NY, USA, 2012).

23. Sennett, R. *The Craftsman*. Yale University Press, New Haven and London, 2008.

24. Tinkercad. `http://tinkercad.com.`, 2012. (Accessed: 06/3/2013).

25. Victor, B. Inventing on principle, 2012.

26. White, B. Thinkertools: Casual models, conceptual change, and science education. *Cognition and Instruction* (1993).