DRESSCODE: SUPPORTING YOUTH IN COMPUTATIONAL DESIGN AND MAKING

Jennifer Jacobs¹, Mitchel Resnick², and Leah Buechley³

Abstract
Computational design is a powerful tool for conceiving and constructing physical artifacts. As researchers invested in making design tools for young people, we believe that the combination of computational design and craft can be highly relevant for young designers interested in aesthetics and making. Youth engagement in these endeavors requires the development of accessible computational tools for physical design. We present DressCode, a novice-oriented computational design tool for craft. DressCode features linked editable representations between textual programming and graphic drawing. The software produces vector designs for 2-axis digital fabrication and handcraft. We describe DressCode and our evaluation of the software through a workshop with young people. This evaluation revealed the diverse design affordances of linked representational tools for young designers, and afforded the creation of projects that blend hand-drawing, computational patterns, and craft.

Keywords computational design, visual design, craft, making

1. Introduction
In Design By Numbers, John Maeda states, “the true skill of a digital designer today is the practiced art of computer programming” [1]. Maeda’s observation points to the role of computational design in contemporary design practice. Computational design is the process of using computer programming for visual design. Using programming, the designer creates a system that is capable of producing many design outcomes depending on input, supporting the production of multiple designs within a set of constraints. Designs produced computationally also are notable for their geometric complexity, or contain generative elements [2]. Today, computational design is an established professional pursuit. We believe that with the right tools and applications, computational design can also be a compelling creative space for young people.

Youth engagement in computational design is limited by practical and perceptual barriers. The programming languages used for computational design require many years to learn and apply [3]. Novice-oriented computer-aided-design (CAD) software exists, but emphasizes graphic manipulation rather than programming. Many young people consider programming to be irrelevant to their interests [4], [5]. Furthermore, existing tools for novice programming frequently emphasize applying programming to create interactive behaviors rather than visual design.

¹ jacobsj@media.mit.edu, Massachusetts Institute of Technology; Cambridge, Massachusetts
² mres@media.mit.edu, Massachusetts Institute of Technology; Cambridge, Massachusetts
³ buechley@gmail.com
We believe that computational design is especially relevant for young people with an interest in aesthetics, visual design, and making. By blending computational design with craft, programming can become a tool to express visual identity. Sewing, jewelry-making, and t-shirt design offer the opportunity to craft a personal style, something that is particularly important in the lives of young people [6]. Professional designers already have begun to apply computational design to the production of fashion accessories and clothing [7] [8]. Our intention is to provide similar, creative opportunities for young people.

In this paper, we describe our efforts to develop a youth-oriented computational design tool called DressCode, developed to produce designs that are compatible with craft. DressCode features linked editable representations that enable people to create designs through simultaneous graphic manipulation and textual programming. We evaluated DressCode in a workshop where young people designed patterns for screen-printed t-shirts. In the process, we examined two primary research topics: How does the design of a computational tool shape the ways in which people design with it? and How is the experience of combining computational design and craft relevant to the lives and interests of young people? In the following section, we outline related work in computational design software. We then describe the DressCode software in detail and our evaluation of it. We conclude with a discussion of the results of the evaluation; we found that dual support of graphic drawing and programmatic manipulation supports different design processes for different people and supports diverse interests in making and design.

2. Related Work

There are three fields of related software tools: learning-oriented programming environments, CAD tools for computational design, and digital-design tools with linked forms of editing. We also developed DressCode within the context of research in combining craft and computational design.

2.1 Learning-oriented programming environments

DressCode was developed to help young people use linked representations in the design of physical artifacts, which distinguishes it from other novice programming tools. LOGO [9], Scratch [10] and Processing [11] all emphasize the production of screen-based media. Although we are avid enthusiasts of all of these environments, we chose to develop our own tool because we were interested in exploring linked representations for visual design, which these existing tools do not support. It was simpler to build our own software than to alter these existing platforms.

2.2 CAD tools with computational design functionality

Although computational design can be performed with general-purpose programming languages [3], there also are professional CAD tools that support computational design. Adobe Illustrator software and 3D modeling tools like Maya and Rhino feature the ability to script behaviors in languages that are syntactically similar to JavaScript, Perl and C#, respectively; however, the programming interface is omitted from the primary software interface. CAD tools that emphasize computational design include Grasshopper (a plugin for Rhino) [12], DesignScript (an add-on to AutoCad) [13] and OpenSCAD [14]. Although these examples are powerful design tools, they are not suitable for novices. Novice-oriented CAD tools with computational design features are rarer than professional tools. FlatCAD allows users to design customized gear-based construction kits by programming in a language modeled on LOGO, and supports design through text-based
programming [15]. TinkerCad is a 3D modeling tool that includes “shape scripts,” which are Javascript programs that produce 3D forms [16]. Autodesk’s 123D tools consist of a variety of novice-oriented CAD tools [17], and enable automated repetition of elements in predefined patterns. Tinkercad and 123D support design for digital fabrication; however, their computational features are limited in comparison to their methods for graphic design.

2.3 Linked editors

Linked representations have primarily been applied to digital design tools in interface-design domains. Avrahami, Brooks, and Brown demonstrated an early approach to a two-view system for designing user interfaces by combining a graphic editor linked with a special-purpose programming language [18]. Commercially, the two-view approach has been incorporated into application interface design tools in software development kits [19]. Victor has advocated for the incorporation of linked representations in other fields, including circuit design, game development, and graphic design [20]. We seek to apply linked representations to a new context of entry-level computational design.

2.4 Craft and computing research

Eisenberg and Buechley’s research on pervasive fabrication provides a survey of approaches and benefits of combining computational design and physical making for young people. Their research describes how computational forms realized through digital fabrication enable youth to decorate their environments and express themselves [21]. In our earlier work, we engaged children in computational design with the Codeable Objects Processing Library [2]. With Codeable Objects, we re-purposed existing software. DressCode is a novel tool we developed to address difficulties experienced by novices using repurposed tools, including learning programming syntax, limited visual feedback, and reliance on instructor assistance to program.

3. DressCode Software Description

DressCode is a 2D vector-graphic computational design tool for craft applications. DressCode supports linked editable representations of a design in two forms: programmatic and graphic. When designers create or manipulate shapes graphically, the software generates editable programming expressions that correspond to their actions. Designers also can generate graphic forms by writing programming expressions.

3.1 System overview

The DressCode interface is divided into two panels: a graphic panel on the right and a text-editor panel on the left (figure: 1). Designers can type programs using the text editor, or they can draw and transform shapes using the mouse in the graphic panel. The text editor contains a console for output and error messages, and a button for running the current program. The design panel contains a resizable drawing board and rulers and tools for panning and zooming. A toolbar in the graphic panel presents a menu of drawing and transformation tools and a print button, which allows designers to export in vector format. The vector graphics consist of points, lines, Bezier curves, and polygons, which are rendered through the JOGL OpenGL wrapper. The drawing tools include regular shape-creation tools and pen and curve tools for free-hand drawing. The selection tool allows for individual shapes to be manually selected and moved. The boolean operation tools allow
for the combination of two or more shapes into a unified form through polygon-boolean operators (union, difference, intersection, and either/or). The interface also contains the stamp panel, described in the following subsection.

We designed a custom imperative programming language for DressCode, which supports conventional programming data types, loops, conditional expressions, and user-defined functions. Variables in DressCode are dynamically typed. The language contains a subset of expressions that facilitate drawing and transformation of 2D graphic geometric forms. The language also supports math expressions and a variety of methods for random-noise generation. The DressCode language functions via an interpreter with semantic functionality that we wrote using a Java-based parser-generator. A note on terminology for the remainder of the paper: we denote actions made in the graphic panel with the mouse as graphic actions, and actions made in the programming 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. Transformation actions denote programmatic or graphic actions which result in an existing shape being moved, rotated, scaled or otherwise altered. We developed the linked representations between graphic and programmatic actions in DressCode around two design principles: correspondence and readability.

Figure 1: DressCode’s software interface with sample design, and a close-up of the graphic tools.

3.2 Correspondence

DressCode maintains correspondence between programmatic actions and graphic actions. For every shape initialized programmatically, a shape is generated in the graphic view. For each graphic action, a corresponding programming expression appears in the text editor. The DressCode programming language was explicitly designed 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, and polygons) 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, with a click to determine the origin, and the distance of the drag from the origin to determine the dimensions. The method-type of the auto-generated expression is determined by the type of graphic tool used to create the shape.

Transformations, including moving, scaling, rotation, color and stroke changes, and shape booleans follow a similar structure to shape initialization. In the programming language, transformations are
performed by either wrapping a shape-initialization expression in a transformation expression, or by assigning an identifier to the shape and then calling the transformation method with the identifier. On the graphic side, each transformation tool corresponds to a transformation method in the DressCode language, enabling the generation of a corresponding expression in the programming panel, which contains as its first argument a reference to the shape selected and manipulated. Throughout the design process, a complete representation of the current state of the design is maintained in the programming panel.

3.2.1 Generativity in DressCode
Generativity is one of the most powerful aspects of computational design, but its systematic nature is not easily represented in graphic form. DressCode deals with this by giving users the choice of capturing the abstraction of a program or the current state of the design using stamps. Stamps are graphically created functions that return shapes. Stamps translate a design generated by random programmatic methods to a set of expressions that describe discrete shapes with hard-coded parameters. This allows users to programmatically represent static instances of a generative design. Stamps are created by graphically selecting a single shape or group of shapes and selecting the stamp option from the application menu. Stamps are listed in the stamp panel and can be added to a user’s primary program by selecting the + icon next to each stamp. The code of a stamp can be modified by the user after it is generated by double clicking on the stamp. This representation allows multiple versions of a generative pattern to be preserved in a single design, and enables specific patterns from a generative program to be shared.

3.3 Readability
Correspondence between graphic manipulation and textual programming must be readable. Merely producing a textual expression that accurately reflects a graphic action does not ensure that the interaction will be understood the designer. We considered how we could design linkages between programmatic and graphic actions that were readable and reconfigurable by young people.

3.3.1 Readable references
Like many existing programming languages, DressCode gives the developer flexibility in how elements are referenced. Because the DressCode language enables methods to be nested within each other, there is a degree of ambiguity in identifier use. In creating the linkages, we considered how shapes should be referenced when transitioning between programmatic and graphic representations in a way that was readable. All graphic actions that create a new shape produce programmatic expressions that are automatically assigned an identifier. Subsequent actions that transform graphics produce programmatic expressions that reference the auto-generated identifier on the following line. This produces lengthier code, but it is more readable than a nested chain of expressions. If a shape that was programmatically initialized without an identifier is transformed graphically, DressCode recognizes the distinction and resorts 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 recognize this and use the new identifier.

3.3.2 Readable edits
It is important to consider where auto-generated expressions appear in a program, and DressCode does this. For an initialization-graphic action the programming expression always appears below the last line in the program. If the designer performs an action to transform a graphic, however, the expression is 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 reproduces the correct order of operations when run. It also provides a form of organization for auto-generated expressions.

The DressCode programming language employs an imperative paradigm where 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 reflect the order of steps a designer made in the graphic interface. 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. Because order of operations matters, if another tool is used to alter the shape, or if a programmatic expression is manually inserted by the designer following the move statement, future actions with the move tool on the same shape generate a new move statement, which will be subsequently updated until another tool is used (see figure: 5). This same logic applies to all other transformation tools. In writing code, the designer may deviate from this organization, however; manual edits will not prevent the ordering mechanism from functioning for successive graphic actions. Furthermore, the consistent rule-set for auto insertion enables designers to anticipate where expressions will be inserted into their code and then textually manipulate the graphical steps they took to arrive at a design.

3.3.3 Tree representation
As programs grow in length and complexity, it is helpful to provide other representations that make structural relationships readable. DressCode features a tree view that contains a listing of all groups of shapes in the current design. Child shapes are nested within their parent groups. When selected in the tree view, a shape is selected and highlighted in the design view, and the line where the shape was last modified in the text editor is highlighted. The tree view provides visual feedback of the connections between the elements of a design and the program, and provides a practical selection technique for complex designs.

4. Workshop

We evaluated DressCode through a 4-day workshop with 7 teenagers, aged 13-17, and one young adult, aged 21. Three participants were male, and five were female. Participants were selected to represent a range of programming, craft, and design experience. In the workshop, participants used DressCode to design images and screen printed them on t-shirts. The process enabled us to evaluate how DressCode could be applied to craft in a realistic setting.

4.1 Workshop Structure and Evaluation

Participants were tasked with creating a computational design to screen print on a t-shirt they would want to wear. We introduced participants to DressCode, focusing on principles of generative design and the use of random noise. In the process, we provided them with a set of example designs that showcased different computational techniques like random walks, clipping masks, and Gaussian distributions. Following a series of iterative critiques and design sessions where people made revisions to their designs, the participants printed their final designs, transferred them to screens through an exposure process, and printed the designs on their shirts. The participants’ experiences were evaluated with written surveys and group discussions at the start, middle, and end of the workshop. 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 workshop.

4.2 Results

Each participant successfully used DressCode to produce a design for a t-shirt that they planned to wear (figure: 9). Two participants also brought in additional garments to print for friends and family. All asked to keep their screens and described plans to continue printing for themselves and others. In the screen-printing workshop, participants took different paths to produce their finished designs. We observed three general design approaches: **Emphasis on programmatic methods:** Three participants used the graphic drawing minimally, almost exclusively relying on generating and transforming methods computationally. **Programmatic manipulation of example graphic elements:** two people used programming expressions to manipulate pre-existing graphics that were provided in examples. **Equal use of graphic drawing and programmatic manipulation:** Three people used the graphic drawing and transformation tools in equal proportion with programmatic methods.

![Figure 9: Several completed shirts. From Left: generative landscape, cloud pattern, geometric spiral, linear-radial pattern, dandelion-random walk, random heart pattern.](image)

Participants’ attitudes towards programming changed after using DressCode. Primarily, this change was positive; participants indicated greater interest in learning programming in the future, a stronger belief that programming was a tool that they could use to create things they would use in their daily life, and greater comfort in programming on their own following the craft activity. The one exception was the most experienced programmer of the group, who said he would prefer to apply programing to screen-based applications. Participants were positive about the graphic tools and requested that their functionality be extended to incorporate a greater range of drawing and transformation methods. All participants said they would be interested in using DressCode for another activity, and all but one indicated they would like to continue using the DressCode programming language.

5. Discussion

Our analysis of the workshop revealed that the dual support of graphic drawing and programmatic manipulation in software resulted in a range of design practices by young people. Furthermore, we found evidence that the way computational features are presented in a software tool directly affects how the tool is perceived by the user. Lastly we gained an understanding of how the activity design surrounding a tool can connect to a broad range of youth interests and values.

5.1 Diversity in design practice

The linked representations in DressCode led to diversity of design practices for novices in two important ways. First, the intuitive graphic tools offered a way to scaffold the process of learning
programming. Second, the correspondence between graphic and programmatic manipulation fostered design aesthetics that contained hand-drawn elements in geometric or generative compositions. Responses from the screen-printing participants demonstrated that the graphic portions of linked representations were easy to use. Before the graphic tools were demonstrated, several participants independently experimented with them and described them as intuitive. Furthermore, linkages between the graphic drawing and programming were considered helpful in understanding the programming language:

Having a graphical side and having it auto-update the code, it can show you that [if] you want to work with the code you’re learning as you’re using the GUI. So that people could try and say ok, if I can’t do something with the graphical tools, let me try and manipulate the code. And then you already have some understanding because you’ve been using the graphical tools and it’s been appearing over there the entire time.

Participant B

The combination of graphic drawing and programming also resulted in aesthetics that we, as computational designers, found exciting. The participants who relied on a balance of graphic and programmatic design tools produced designs that contained imperfect, irregular forms in direct conjunction with generative structures, geometric repetition, and visual complexity. These designs were distinct to the individual creator, and their imperfections were deliberate. During one critique session, the creator of a heart-shaped design explained that he had deliberately chosen to draw an irregular heart form because that was his style (figure: 9). We characterized this as a hand-drawn/computational hybrid, and see this as an instance when computational tools provided evidence of the human hand rather than eliminating it. We consider the development of a more nuanced palette of hand-drawing graphic tools and mechanisms as a worthy step for exploration in this regard.

5.2 Tensions in linked representations

When developing computational tools with linked representations, it is important to create relationships between graphic and programmatic actions that do not make assumptions about the intentions of the designer. Furthermore, any correspondence between graphic and programmatic functionality should be judged not only for ease of use, but also for how effectively it communicates the computational concept. People in the workshop had a range of opinions about new graphic tools to add to DressCode, and how graphic tools should correspond with programming. Because most people used some form of randomness in their design, there was a debate about the creation of random distributions using graphics or programming. Two young women stated that they would have preferred graphic tools that enabled them to create random groups of objects by pointing and clicking with the mouse. Three others stated that they preferred to create random distributions using loops and variables in the text editor, and recognized that by creating random distributions graphically, DressCode would become too similar to existing tools and lose some of the control gained with programming:

The important thing I really feel about DressCode is you can turn things like “random.” These are drawn [gesturing to a graphically drawn element of his design] and these are from the programming part [gesturing to the repetition of them] ... but in [Adobe] Illustrator it’s just drawing. You can’t have everything.

Participant Z

Similarly, participants had different ideas about how new graphic tools should generate programming expressions. Here two people discuss their ideas for the incorporation of an eraser tool:
Participant Z (Talking about erasing graphically): 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 it, you don’t have to go to the code, erase it, 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?

In linked representations, we argue exercising caution in linking every programmatic method to a direct-manipulation equivalent because this may severely limit the creative potential of the software. For example, relegating random distributions to a predefined set of graphically selectable icons may make them more accessible initially, but this could also prevent a designer from applying a random pattern in a unique way, or modifying the underlying algorithm of a pattern. Furthermore, the automatic inclusion of features that seem simple (such as an eraser tool) may in fact lead to inaccurate assumptions about the design objectives of the people using the tool. In developing linked representations between programming and graphic interactions, we advocate incorporating graphic drawing tools that are intuitive and familiar and can be directly linked to a single programming method. We suggest developing alternative solutions for programmatic structures that are difficult to represent graphically (for example our use of stamps to deal with the correspondence issues raised by randomness). Lastly, we do not rely exclusively on graphic tool correspondence to scaffold understanding of programming, but rather incorporate other forms of visualization and representation, such as the tree view, to help people understand the relationships between the program and the design.

5.3 Designing experiences for diverse interests

People approach tools and activities with different motivations, and their response to a tool will be affected by how relevant the activity is to their subjective interests. In the workshop, we observed great variation in how people valued their experience. Several young people were concerned about producing artifacts that were unique or that expressed their personal style, whereas others were excited about the opportunity to learn to program. We also found that people valued the opportunity to learn a craft process and the physical labor required to finish a project. Other people talked about what they wanted to do next with DressCode, including creating gifts for friends, building things they could sell, or combining DressCode outputs with 3D printing. In many youth computer-science activities, we believe there is an inordinate emphasis on learning computational concepts rather than supporting diverse and subjective experiences. Brennan and Resnick point out that in observing young people use Scratch, framing experience around computational concepts insufficiently represented other key elements of learning and participation [22]. Building on this idea, we advocate designing computational activities in a way that actively supports different experiences for different people. The deep and thoughtful integration of craft with computation offers an excellent way to achieve this diversity because of the variability of craft materials and the interconnected pathways of making that emerge when interweaving computation with handiwork. In addition, just as the design of a computational activity can determine the range of experiences, the design of a tool can determine the types of people willing to use it. In the post surveys and discussions with participants, we noticed that people associated DressCode with digital graphic design tools, rather than programming tools. Participants stated that it reminded them of Photoshop, Illustrator, and MS Paint. We find this notable, given that half of DressCode’s interface is dedicated to textual programming. For us these responses reinforce the importance of developing computational tools that provide programming as an accessible option, but equally present features that will resonate with those new to programming.
8. Future work

We view DressCode as a starting point for continued work in developing software tools to support computational design and craft. There are many directions for future study. We are interested in exploring how computational design software can incorporate hand drawing. This pathway raises several questions. How can the continuous information in a drawing be represented in a way that allows novices to manipulate it programmatically? Further, what other computational approaches might be applicable to hand drawing? There is also significant work to be done in understanding how computational design tools can be incorporated into existing efforts to engage young people in making. What support structures are needed to integrate DressCode into maker spaces?

As computers grow in complexity, it is possible to view computation as incompatible with familiar forms of making. In our experience with DressCode, computation can add to established forms of creation. We look forward to the growth of future tools that provide people the opportunity to experience the pleasure and excitement of using programming and their hands to build something of their own.

References