Greater than the Sum of its PARTs: Expressing and Reusing Design Intent in 3D Models

Megan Hofmann,[*] Gabriella Hann,[*] Scott E. Hudson,[*] Jennifer Mankoff[†]

[*] Human Computer Interaction Institute
Carnegie Mellon University, Pittsburgh, Pennsylvania
{meganh, ghhann, scott.hudson}@cs.cmu.edu

[†] Allen School of Computer Science and Engineering
University of Washington, Seattle, Washington
jmankoff@uw.edu

ABSTRACT

With the increasing popularity of consumer-grade 3D printing, many people are creating, and even more using, objects shared on sites such as Thingiverse. However, our formative study of 962 Thingiverse models shows a lack of re-use of models, perhaps due to the advanced skills needed for 3D modeling. An end user program perspective on 3D modeling is needed. Our framework (PARTs) empowers amateur modelers to graphically specify design intent through geometry. PARTs includes a GUI, scripting API and exemplar library of *assertions* which test design expectations and *integrators* which act on intent to create geometry. PARTs lets modelers integrate advanced, model specific functionality into designs, so that they can be re-used and extended, without programming. In two workshops, we show that PARTs helps to create 3D printable models, and modify existing models more easily than with a standard tool.

Author Keywords: Fabrication; prototyping; 3D Printing
ACM Classification Keywords:
H.5.2 [User Interfaces]: Input devices and strategies, Interaction styles

INTRODUCTION

Consumer-grade 3D printing allows the creation of customized objects by nearly anyone. However, people are limited by their modeling ability. Most modelers quickly move beyond novice oriented tools for creating geometric designs (*e.g.*, TinkerCAD [47]) because expression of geometric form is only one part of the modeling task. Bridging the gap between geometry and function is a more substantial challenge [34], even for experienced users.

Modelers would benefit from the equivalent of an end-user-programming tool. This is what our *Parameterized Abstractions of Reusable Things (PARTs)* framework provides. It puts advanced methods for capturing 3D modeling design intent in the hands of non-expert modelers. This supports reuse, experimentation, and sharing.

PARTs' basic abstraction, *functional geometry,* is analogous to the programming concept of classes [9,22]. Like classes, functional geometry encapsulates data and functionality, making it easier to *validate* and *mutate* data and support *modularity*. *Functional geometry* includes *assertions* that test whether a model is used correctly, and *integrators* that mutate the larger design context. These abstractions increase model usability and re-usability.

The PARTs framework is an extension of the Autodesk Fusion360 Computer Aided Design (CAD) tool. CAD tools provide many helpful capabilities, including validation methods [48], simple geometry operations [49], parameters, constraints [50], and data structures for hierarchical composition [51]. PARTs builds on this past work by uniting this type of functionality within a single framework, making this functionality easier to use and providing users with a single, consistent mental model. It does this through small, but important, additions to the Fusion360 GUI that provide three important benefits. First, PARTs gives designers a way to represent abstract design intent geometrically, so that less skilled modelers can easily see, react to, iterate on, and experiment with it. Second, PARTs facilitates description of the modeling context with assertions of design expectations. This makes model reuse simpler and more intuitive and is particularly useful when a design is reused in a new context. Third, PARTs enables encapsulation of design ideas. This allows separation of concerns between aspects of a design. It also makes it easy to combine and extend designs all in one unified interface. By leveraging these unique features, a designer can easily encapsulate information about how a design should be used. This makes direct manipulation of design intent possible, which, in turn, makes model reuse, customization and recombination simpler and more intuitive.

When experts can share designs in a re-usable fashion, amateurs can accomplish more [23] –a successful pattern among programmers [15] and makers [17,29,35]. A study of collaboration among professional CAD users shows that programmers extend their tools to share their skills with less skilled users [14]. Similarly, PARTs enables sharing of new capabilities by *designers* (non-experts who create and share complex designs) with *modelers* (amateurs who can use basic CAD tools).

CHI 2018, April 21–26, 2018, Montreal, QC, Canada

ACM ISBN 978-1-4503-5620-6/18/04…$15.00
https://doi.org/10.1145/3173574.3173875

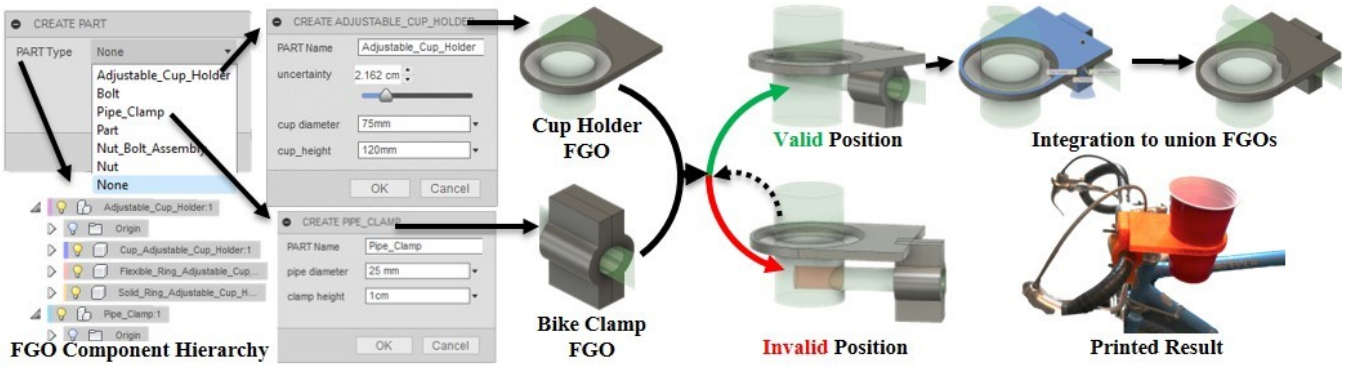


Figure 1. Modelers combine geometry and logic to define FGOs as a set of assertions and integrators. For example, to create a cup holder that attaches to a bike handlebar, the modeler instantiates cup-holder and bike-clamp FGOs from the PARTs library (A), and iteratively modifies them in Fusion360 while the FGO assertions help to visualize potential errors (B). When the modeler is ready, he can integrate the cup-holder and bike-clamp to create one design (C). The printed result is from our second workshop.

Scenario: Creating a Cup Holder Mount for a Bicycle

Consider a hypothetical *modeler*, Kavi, who is creating a bike-cup holder. Kavi views a dialogue of available *functional geometry objects* (FGOs) to find a cup holder and bike mount (Figure 1A). FGOs capture design intent to make it visible to the *modeler* (see the semi-transparent cup in Figure 1B). Kavi selects the cup holder to add its geometry to the Fusion360 model space and component hierarchy. He adjusts the cup size to match his water bottle. Then he adds the bike clamp and positions them so they overlap (Figure1B). PARTs checks for violations of design intent (failed *assertions*) and highlights them in red during modeling. For example, when the bike handlebar would block the cup, it is highlighted in red. Kavi changes the model until he finds a valid position and then integrates the cup holder to join them together (Figure 1C).

The cup holder and bike mount are fairly specific FGOs that are not in the default PARTs library. PARTs can easily import an FGO embedded in any Fusion360 file, to facilitate sharing of FGOs. We travel back in time to when a more experienced *designer*, Nisha, creates the cup holder FGO that Kavi downloads. She specifies her design intent by adding *assertions* and *integrators*. First, she models the geometry of the cup holder. Nisha then uses PARTs to specify that this should attach to any surrounding geometry by right clicking on the geometry and associating it with a union *integrator*. Next, she creates a cylinder representing the cup. She attaches the cylinder to an *assertion* that tests for interference using PARTs, specifying that it should always leave room for the cup. This assertion automatically highlights violations when Kavi positions the FGO.

Suppose a different *designer*, Lori, created Kavi's handlebar mount. Lori models the clamp and adds an *assertion* that ensures space for the handle bar. She also adds a fastener to close the clamp. The PARTs library includes a fastener FGO with length and diameter parameters; assertions ensuring there is enough material around the bolt and that the bolt has a clear path; and an integrator to create a bolt hole. All of this complexity is now encapsulated in the handlebar FGO.

This scenario highlights how the PARTs framework engages multiple classes of users in a synergistic way. Similar to the user types proposed in [30], PARTs supports the transfer of designs between *programmers,* who extend the framework*, designers* (Nisha, Lori)*,* and *modelers* (Kavi)*.* PARTs supports reuse at multiple levels, and enables the creation of complex, hierarchical objects.

Overview & Contributions

We first present an analysis of design patterns for reuse in the wild derived from our survey of 962 Thingiverse models. Next we describe our primary contribution, which supports such re-use, the PARTs framework. PARTs provides a GUI for creating new FGOs and showing real-time feedback, without programming. PARTs includes a runtime architecture and a small, but extensible, library of powerful *assertions* and *integrators*. PARTs can also be accessed through the Fusion360 scripting interface for adding even more advanced capabilities.

Our technical validation demonstrates that PARTs has the flexbility to address a wide variety of 3D modeling challenges for non-experts. The PARTs framework supports many tasks in-*situ* that are normally handled in special dialogues where non-experts may not find them or understand how to use them. While PARTs intentionally uses simple concepts to accomplish this, many model-specific design goals can be encapsulated using its *assertions* and *integrators*. We show the value of PARTs in creating complex objects, making a common CAD fastener tool easier to use, and reproducing existing research [19].

We studied PARTs' impact on *modelers* and *designers* in two workshops. Ten participants with an average of five years of experience with CAD tools were asked to create cup holders, similar to the scenario above. Participants could successfully create and reuse objects using PARTs more effectively than with the standard Fusion360 tool set.

RELATED WORK

Personal-scale fabrication techniques have given rise to new communities of makers and prototyping methods [4,33]. However previous research suggests that experts and

amateurs struggle to use CAD tools to express their designs. Amateur modelers have difficulty with issues such as uncertain measurements [19], spatial reasoning [7], understanding how parameters affect overlap and fit [21] and relating 3D models to real world geometry and objects [2,8]. While non-experts avoid some of these difficulties by using existing models found on sites such as Thingiverse, often such models need to be modified to be useful.

The extensive component libraries available in tools such as SolidWorks are limited in applicability to mechanical settings. Broader libraries such as SketchUp's 3D Warehouse do not necessarily support re-use. Studies find that modelers struggle to make design modifications [17,39]. Even where customization is possible, modelers do not engage in much reuse or innovation [35]. Although experienced designers may create and share parameterized models, designers struggle to anticipate the future use of those parameters, making correct reuse more difficult [21].

Even for expert modelers there is a gulf of execution [34] between a modeler's real world design intent and the geometry of most 3D models. Manufacturing experts have developed methods for documenting design intent in a separate process from 3D modeling [13,20,37]. Because of these common professional practices, there is a connection between geometric features and professional modular design intent [27], but this connection is lost in standard tools [13]. Further, although these connections are based on the common norms, there remains disagreement on terminology and best practice [26]. It is likely that expert amateurs have difficulty uncovering these norms. Indeed, a key challenge in the domain of non-expert rapid prototyping is to develop systematic engineering practices which require low cognitive effort [1].

Kim *et al.* have proposed a standardized format for CAD models that relates design intent more closely to geometric features and preserves this information for reuse [20]. However designs using this format are likely unusable to modelers, and even designers, without appropriate training. These practices and issues have influenced the features available in CAD tools (*e.g.,* composability [24], and support for assessing manufacturability [43], assembly [25], and strength [28]). However, non-experts may not be aware of, or find it easy to use such features because they do not share the norms and knowledge of professionals.

There is a long history of interactive systems research aimed at increasing the expressiveness of modeling tools. ThingLab used objects and decomposition to support specification of graphical simulations [6]. SketchPad [40] introduced constraints for expressing design intent and are used for manufacturing [10], graphics [5,40,41] and user interface design [16,31,36,46]. Xia *et al.* added object oriented structure and modularity to a 2D sketching tool [44]. This work shows the positive value of objects and modularity in design, but provides limited extensibility for end users through the interface, which limits the ability of modelers to define their own modular features. Many tools already aim to express specific forms of design intent, such as: connecting to physical objects [12,38,45], or assembling modular components [18,23,25]. These tools illustrate the power of attaching design intent to models, however none focus on recombination and reuse of such designs.

A SURVEY OF REPEATED PATTERNS IN THE WILD

We begin with a survey of 3D printable designs found on the Thingiverse 3D model sharing website, which is used by our target non-expert population. We explore design patterns for reuse among successful examples of complex 3D models.

Method: We surveyed 10,560 designs on Thingiverse, gathered in two phases in 2016 and 2017. Each year, we gathered 40 pages of 12 Things from Thingiverse's 11 categories (5,280). We collected 20 pages of the most popular designs at search time, and 20 of the newest designs. Of these, we kept the 962 designs that met all of the following requirements for further review:

Success: Fabricated, or marked as 'made' by a different Thingiverse user (8,839, 83.70%).

3D Printable: Intended for 3D printing (9,257, 87.66%).

Non-Trinket: Images and descriptions show interaction with existing objects. (1,246, 11.81%)

For each design, we collected photographs, descriptions, popularity metrics, and file types (coded as static or parameterizable). We used affinity diagramming [3,45] to group designs. Although modelers express design intent using many different approaches and software tools, we found a few very common design patterns for incorporating existing objects into models. As suggested by Flath *et al.* [11], these design patterns are important building blocks for reuse. Patterns we found, shown in Figure 2, included: approximating (682), molding (219), holding flexible objects bent to fit the model (51), connecting multiple similar objects (225), connecting multiple distinct objects (159), and hanging (10).

Findings: Reuse of parametric designs is common (74% of the reuse graph in [35] comes from parametric designs). Our data show a similar pattern. Of the 962 designs, 536 (56%) have been reused at least once. Parametric models are more likely to be reused than non-parametric models. Most reused designs (89%) were made available through the Thingiverse Customizer tool, a simple GUI for modifying parameters.

Surrounding Objects: Approximation, Molding and Holding. The majority of designs (578) use 3D printed materials to augment a single object. Most of these use two similar patterns: *Approximations* of the object's shape made with cylinders or boxes, or *molds* that exactly match the object geometry. Approximations are loose-fitting holes for simple objects like pens, usually made with cylinders or boxes (*e.g.,* T:73489). Molds more exactly match the geometry of the object, meaning the design is molded to fit the object like an iPhone case (T:40703). A similar and rarer design pattern is to *hold* objects that are flexible, where the *modeler* intends

Figure 2. Example Thingiverse Designs for each design pattern. Labels show 'Thing' number.

to bend the real-world object to fit into the space. This is usually seen with objects like cables and wires (T:13678).

Connecting and Organizing Objects: Many designs (N=384) use 3D printed models to connect and organize multiple objects, such as a pegboard tool organizer (T:1322545). Less common were designs that connect different types of objects such as a smartphone and game controller (T:2457666)

Discussion: Similar to findings in [11,35], our analysis suggests that people are interested in reuse, but only if they can easily modify a design (as with OpenSCAD or Customizer). If 3D modeling tools made customization as easy as parameter modification, we should see examples of reuse that leveraged those capabilities in our data. Yet, similar to [35], we find most reuse to be relatively minor parameter variation and reproductions. We disagree with Flath et al's [11] assessment that the models on Thingiverse are all combinable; rather modelers in our study struggled to create compositions of multiple distinct triangular mesh 3D models (the most common type on Thingiverse).

Our data set is limited by our focus on Thingiverse, which is known to include few parameterized designs [32]. However, this is a representative sample of the design efforts of a non-expert population. Another limitation is lack of information about what happens to the same designs outside Thingiverse. Because our analysis was done in a bottom up fashion, it does not call out what is missing in the data. It is possible that reuse is happening, but is not visible in our data set. We did not identify what design tools *modelers* who post STLs use, what design features *modelers* used, and how these features may have connected to design intent. However, if the original model is not shared, this intent is lost during reuse.

THE PARTS FRAMEWORK

The PARTs framework allows models to include design intent, making them easier to reuse. Through small changes to the Fusion360 GUI, PARTs supports creation and use of *functional geometry*. In this section we describe the PARTs architecture, including the *modeler*'s experience in using Fusion360 to instantiate and customize *functional geometry* and an overview of the PARTs library.

Overview of Functional Geometry

The purpose of *functional geometry* is to capture design intent in ways that can be easily visualized so that the *modeler* can work to avoid problems as models are combined to build up more complex designs. From a *modeler*'s perspective, *functional geometry* is similar to other Fusion360 components. Just as some Fusion360 commands take a surface or other geometry as input, *functional geometry* makes use of geometric parameters. Fusion360 allows the specification of model constraints, but *functional geometry* goes further than standard parameters and constraints to visualize violations of *assertions* that represent semantic expectations about models (Figure 1B).

A critical feature of PARTs is that a *designer* uses FGOs to express semantic concepts geometrically and without programming. An example is adding a cup model to a cup holder. When shared with a *modeler*, this reveals important considerations: 1) do not interfere with the cup, 2) size the hole to match the cup. The ability to use standard constructive geometry operations to document design intent is a powerful, unique feature of PARTs that positively impacted ease of use in our workshops.

From a programmer's perspective, an FGO is a data structure that inherits from the PARTs base FGO class hierarchy. It includes code for generating geometry and for operating on the geometry based on design intent. Each FGO includes two special types of components. *Assertions* check that the FGO is valid, *i.e.,* not violating some part of the design intent. Unlike constraints or parameters, *assertions* do not enforce a set of rules. Instead, they visually highlight violations. For example, an *assertion* might check that the model is not intersected by other geometry. *Integrators* mutate the model, such as cutting a hole for the FGO or generating a connection between the FGO and the surrounding model. *Assertions* and *integrators* can act on standard model geometry as well as other FGOs. *Assertions* and *integrators* are complementary in that *assertions* check and report on design intent violations, while *integrators* enact aspects of that intent.

Each FGO is composed of *assertions*, *integrators*, and related child FGOs. The FGO base class also includes a constructor that can be programmatically overwritten to generate geometry based on user specified parameters.

The PARTs Architecture and Implementation

The PARTs framework is implemented using Fusion360's scripting API. Because PARTs is deeply integrated with Fusion360, using it involves using standard Fusien360 capabilities for creating and manipulating geometry, such as the component hierarchy and interference and using PARTs commands embedded in the GUI.

Instantiation (Figure 1A)

Modelers instantiate a new FGO by opening a dialog showing the PARTs library from the Fusion360 *create* menu. At instantiation the default elements' geometry is generated and added to the Fusion360 component hierarchy. A dialogue is shown if parameters are needed to generate

geometry used by the FGO's *assertions* and *integrators*. The FGO's geometry can also be customized using standard Fusion360 features, and the *modeler* can add new *assertions* and *integrators* from the PARTs library.

A *designer* can create a novel FGO without programming. The designer starts with an FGO containing no *assertions* or *integrators*, and progressively adds them.

Internally, PARTs keeps a look up table that associates the component and FGO. When the user right clicks on a component, PARTs uses this table to find the associated FGO. If an FGO is found the menu displays commands to copy, delete, add functionality to, and integrate the FGO.

Assertion Checking (Figure 1B)

The PARTs framework automatically tracks changes to geometry and checks any *assertions* that intersect the changed geometry. Failed *assertions* are highlighted in red (Figure 1B). PARTs listens for Fusion360 timeline events which signal model changes, triggering a test for each *Assertions* that intersects modified geometry. This reduces the computation needed after changes in a complex model.

Importantly, PARTs does not impose solutions, interrupt *modelers*, prevent errors, or optimize designs to solve problems. Instead, PARTs provides visual information about design intent. Stronger design requirements can be captured using Fusion360 tools such as parameters or constraints. PARTs' complementary approach allows non-experts to understand and manipulate design intent, as we show in our workshops. By not strictly enforcing *assertions*, PARTS allows experimentation with design intent, something workshop participants praised.

Integration (Figure 1C)

Integrators are defined by the *designer* of an FGO. Prior to integration, the integration geometry helps to indicate design intent. Modelers activate *integrators* by right clicking in an FGO and selecting the integrate command. This converts the integration geometry into a feature of the surrounding model (*e.g.*, cutting a hole for a bolt).

Although integration mutates the model, Fusion360 has sophisticated support for undo, which PARTs leverages. A *modeler* can delete/edit the integration actions, which are labeled in the Fusion360 timeline, or re-invoke integration.

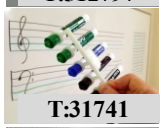
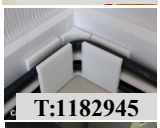
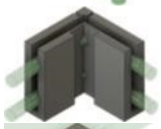
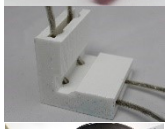
The PARTs Library

The PARTs Library contains base classes for *assertions*, *integrators* and *functional geometry*. After describing the base classes, we present *assertions* and *integrators* derived from our Thingiverse study. Changing the geometry connected to these library elements helps generalize it to a wide variety of new use cases without programming.

Assertions

The purpose of *assertions* is to test whether a design meets the original *designer*'s expectations, even when a new *modeler* alters the model in the future. For example, an *assertion* might test for space for an object, path for tools needed in assembly, or the presence of a related part.

Table 1. Design patterns used to create sample objects.

FGO	Survey Example	PARTs Model	Printed Result
Bounding Box			
Scaled Mold			
Swept-Path			
Connector			

Base Assertions: The *assertion* base class takes a geometric parameter as input and runs a Boolean function against that parameter. PARTs includes two subclasses, which test *interference*, and *overlap* of the geometric parameter against the surrounding geometry. Interference fails if there is an intersection, such as when a path for a tool is blocked. Overlap fails if any part of a geometric parameter is not intersected. For example, a privacy cover for a camera might associate an overlap assertion to check that there is no geometry in front of a camera lens.

Integrators

While *assertions* test design expectations, *integrators* use geometry and other parameters to enact design intent. *Integrators* can be combined, so that an FGO might include one *integrator* that puts a mold around a phone and another that attaches the phone holder to the surrounding geometry.

Base Integrators: The base class for *integrators* takes a geometric parameter as input and runs a function that modifies the surrounding model against that parameter. We provide two subclasses: union and cut. By combining multiple *integrators*, FGOs can execute a wide range of mutations to integrate with the surrounding model.

Implementing Support for Higher Level Design Patterns

Based on common constructs found in our Thingiverse survey, we identified four higher-level design patterns that we support with the PARTs library (Table 1). In three cases, the design pattern is supported by an *integrator* that cuts a hole for an existing object and two related *assertions* that test that the hole remains empty, and is surrounded by a minimum amount of material. These inherit from the cut integrator, interference assertion, and overlap assertion, respectively. The fourth case, connect, uses a single *integrator*, which generates material connecting two faces.

Bounding Box: Many designs in our survey approximate real-world objects using rectangular holes. To support this, we developed a *box-cut integrator*, *interfered-box assertion*, and *minimum-boundary box assertion*. Each generates an

object's bounding box, with either manually entered dimensions, or Fusion360's bounding box functionality, if the user provides the real-world object geometry. Table 1, Row 1 shows a thumb drive/ smartphone holder from our survey, (left), the PARTs model (middle), and the printed result (right).

Scaled Model: Following the mold design pattern in our survey we developed a *scaled-hole integrator*, *interfered-model assertion*, and *minimum-surrounding material assertion*. The *scaled-hole integrator* cuts space (a hole) for a real-world object and the interfered-model assertion ensures it stays clear. Each takes a model of the real-world object as a parameter (which a *modeler* could import or create). The minimum-surrounding material assertion checks that material surrounds the hole. This takes the same geometry as input, but scales it to a thickness defined by the user, using Fusion360's scale feature. Row 2 of Table 1, shows a drawing tool modeled with these elements.

Swept-path: Some designs in our survey define paths to hold flexible objects. We support this with a *path-cut integrator*, *interfered-path assertion*, and *minimum-boundary around path assertion* that use Fusion360's swept path feature to create paths for an object based on a user defined curve and orthogonal profile. These elements are more difficult to use because they require the definition of a curve and profile that match the object. Row 3 of Table 1 shows a cable organizer modeled using these elements.

Connection Integrator: The final design pattern found in our survey is connecting multiple objects. The *Connection Integrator* is parameterized by a starting and ending face, and grows a structure between them using the an adapted Steiner Tree algorithm [42]. In our adaptation, the connector grows from a set of points uniformly distributed across the starting face towards the target face. Starting with the furthest point from the target, each Steiner point searches its neighbors and creates a new point at the intersection of two cones directed at the target. If the new point is closer than the target, a connecting structure is generated between the first point and the new point. The new point is added to the set of Steiner points. Otherwise, a support is generated that connects the Steiner point to the target. This is repeated while there are unconnected points. As with [42], our algorithm does not account for weight. However, Table 1, row 4 shows that it holds a hefty mug full of coffee. Future work should examine how to model physical properties in PARTs.

While the design concepts behind these library elements could, in model-specific ways, be implemented without PARTs, it is PARTs that imbues them with both usability and re-usability through encapsulation. These powerful library elements can express a wide range of design intentions.

TECHNICAL VALIDATION

We demonstrate the power of PARTs through examples of complex FGOs created without programming. We then show how PARTs' ability to extend to and improve on common tools by creating a fastener wizard that tests for proper placement of the bolt. We next demonstrate PARTs' ability to support new types of tools by adding support for uncertain measurements [19]. Each of these new library elements can be accessed through the PARTs interface.

Composing Functional Geometry to Create an Organizer

In this demonstration, our hypothetical *designer*, Nisha, creates a tool organizer, like the 54 models found in our survey, using functional geometry to define her design intent. This example starts with a high-level goal: organizing tools by hanging them from a pegboard. Nisha breaks this problem into a series of simpler tasks: build a pegboard hook; incorporate a bolt and nut into the hook; build an organizer; add hooks to the organizer; arrange tools in the organizer; add multiple tools to the organizer. The entire design is done in the Fusion360 GUI and only requires basic modeling skills, creating boxes and cylinders.

Nisha first defines a pegboard-hook FGO with a union integrator parameterized by a small box to cover a peg hole. She also defines one assertion that ensures nothing in the model interferes with a thin box, representing the pegboard, behind the integrator. Next, Nisha adds a fastener FGO as a child of the pegboard-hook. She positions the nut inside the box while the head of the bolt protrudes. The bolt head will function like a peg (Figure 3, left). At integration time, integrators associated with the fastener FGO will cut holes for the bolt and nut. Having created a hook, Nisha solves the larger problem of using many hooks to hang an object from the pegboard. She creates a box to hold tools and attach hooks to. Using Fusion360 features, she makes multiple copies of the pegboard hook and spaces them on the back of the box so the box can hang from the pegboard.

Next, Nisha defines a tool FGO using a Scaled-Hole integrator and interference assertion parameterized by an approximation of the tool's shape. She makes slightly altered copies of this FGO to represent different tools. Then she positions the tool FGOs in the organizer box. When they are instantiated, tool-shaped holes will be created.

This example demonstrates that PARTs supports decomposition of a problem into subtasks: create a pegboard

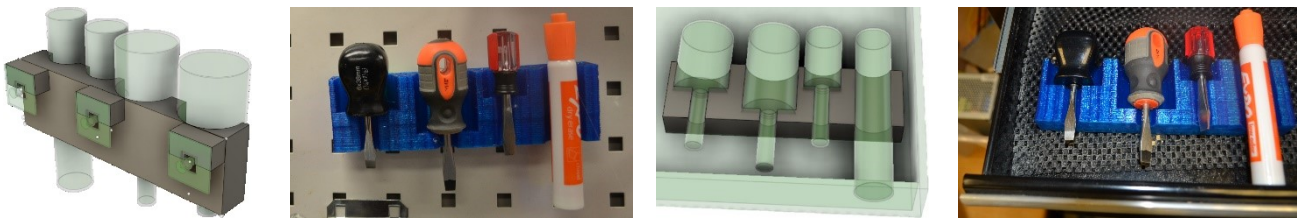


Figure 3. Models and printed results of a PARTs tool organizers to hang from a Pegboard (Left) and fit in a Drawer (Right).

hook; use a fastener, *etc.* As we will show next, one of the advantages of composing a design from subcomponents is the ease with which design intent can be changed or reused.

From Pegboard to Tool Drawer Organizer

Suppose that our modeler, Kavi, wishes to reuse the tool organizer to fit into a drawer rather than hang from a pegboard (Figure 3C&D). He first creates a hollow box, representing the drawer, with an interference assertion. This ensures that the organizer fits completely within the drawer. He places the organizer inside the box, which triggers an error highlighting the pegboard hooks that intersect the drawer. This prompts him to remove the unnecessary hooks. With these two small changes, Kavi has repurposed the existing design.

Expressing Design Intent in Complex Assemblies

The tool organizer demonstrates the benefits of composing relatively simple geometry to express design intent. Going further, PARTs is powerful enough to describe complex objects. To illustrate this, we recreate an example from our Thingiverse survey, a lamp composed of CDs, bike spokes, a bulb, and 3D printed components. Although this design involves many real-world objects, the Thingiverse model includes just the 3D printed geometry. In contrast, PARTs describes the whole design, including the use of the non-printed objects. This makes the final model easier to understand and reuse (Figure 4).

The designer, Lori, first creates FGOs for a CD and bike spoke. She models the CD as a thin disk and finds bike spoke geometry to import online. She uses the *scaled-hole integrator* and a matching *interfered model assertion* to reserve space for the spoke. Next Lori creates a new FGO, a *connector*, which attaches the CD to the bike spoke. To this she adds, as children, the spoke FGO and a fastener FGO. She customizes the fastener FGO by adding a washer to connect the cylinder to the CD. These FGOs will cut holes in the connector, creating an assembly of the spoke and CD.

Next, Lori creates a hub FGO with an arm for each spoke. To this hub, she adds an interference assertion to reserve space for a bulb socket also found online, and adds a *swept-path interference assertion* and the corollary integrator to represent the cable. Lori creates and adds a lightbulb FGO. The bulb FGO uses an interference assertion and a sphere around the bulb to ensure that no meltable plastic is too close to the hot bulb.

This example illustrates how functional geometry not only improves modularity and specifies components for 3D printing (the connectors and central hub), but also shows how the components interoperate with real-world objects (CDs, bike spokes, nuts, bolts, washers, lightbulb, socket, and cable). The FGO hierarchy created by the *designer*, Lori, lays out the relationships between the various objects, making it easy to alter the design while maintaining her design intent and expectations. In addition, this demonstrates PARTs' flexibility, including an ability to incorporate models found online, in the library, and to create custom geometry.

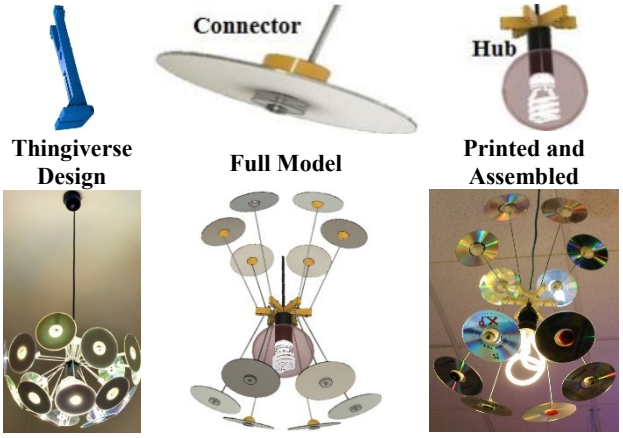


Figure 4. Functional geometry contextualizes the modeled components among existing objects.

Extensibility of PARTs through Scripting

Many CAD tools can already be extended with scripted plugins, indeed PARTs is one such extension. PARTs supports the encapsulation of new features by programmers because it provides a simple framework that enables programmers to create new library elements that express general concepts, are accessible through a unified interface, and can be shared widely. The *connector integrator* described earlier is an example of this. Here we illustrate the concept with a fastener FGO, based on standard existing tools, and an uncertainty buffer FGO based on prior research.

Fastener Wizard FGO: Fastener Wizards are common in CAD programs. For example, Solidworks has two tools similar to the fastener implementation in PARTs. The Solidworks "Hole Wizard" [52] helps modelers create holes that fit to a library of standard hardware. A separate Solidworks tool is the "Smart Fasteners" [53] which insert representations of the needed fasteners into an assembly (a separate stage from modeling) and validates the configuration. An advantage of PARTs is its ability to validate and integrate the fastener in one modeling stage.

The fastener FGO is specific to nuts and bolts. It consists of child FGOs for the nut and for the bolt. Each child FGO has an interference assertion that reserves space for the hardware, a cut integrator that cuts holes with surrounding clearance for the hardware to slip into, and an additional overlap assertion that checks for material surrounding the resulting holes. By sharing a parent FGO, the bolt and nut FGO are able to share parameters (bolt diameter and length) and be moved through the design as one component, maintaining their respective alignment.

The fastener FGO overrides the standard FGO class to add the bolt and nut FGOs. The bolt and nut similarly override the standard FGO class to add interference and overlap assertions and a cut integrator. The implementation generates geometry for the bolt and nut parameterized by shared information about the fastener size.

Like a conventional fastener wizard, the fastener FGO can integrate itself into the surrounding model by cutting an

appropriately size hole. Unlike traditional tools, the fastener FGO can be extended to express new design intent, such as a test that reserves space above the bolt and ensures a tool could reach it during assembly.

Uncertainty Buffer FGO: Our final example is the replication of a research result exploring a novel modeling construct that does not exist in current CAD packages: support for measurement uncertainty caused by modeler error [19]. We implement support for a flexible buffer that can accommodate small errors in diameter measurements of real-world objects.

The *ring buffer* FGO approximates a real-world object given a measure of uncertainty about that object. For example, a modeler could measure a cup three times and enter the diameter range they found.

The ring buffer uses two integrators to create this mechanism. The first integrator generates a ring around the body model that has a radius of the uncertainty parameter. The ring remains separate from the model and would be printed with flexible materials. The second FGO uses a union integrator to generate a hard ring that surrounds the buffer. The FGO creates a cup holder that accommodates mugs of different sizes and shapes.

This shows the power of PARTs to describe a designer's intent with respect to objects that do not have fixed or known dimensions, and thus break the assumptions of standard parametric modeling tools.

EVALUATION: MODELING WITH DESIGN INTENT

To validate that the PARTs framework would allow non-expert modelers to create reusable and validatable designs, we held two workshops with a total of ten participants who each had previous CAD experience.

Participants

Participants were sampled from non-professional 3D modeling and rapid prototyping communities. The first workshop was conducted with six members of a makerspace (ID's starting with A). The second workshop was conducted in a research lab with 4 graduate students (ID's starting with B). Demographic and experiential data for participants is shown in Table 2. Participants averaged 5.1 years of informal CAD experience with a wide range of tools including: SolidWorks (A2,A3,A6,B1,B2,B3), Blender (A3,A4,A5,B2), OpenSCAD (A3,A6,B4), and Fusion360 (A2,B1). No participants had formal training in 3D modeling, qualifying them as expert-amateurs.

Table 2. Participant Demographics

ID	Gender	Age	Experience	Profession
A1	Male	36	3	Software Engineer
A2	Male	31	10	Graphic Artist
A3	Other	40	20	Startup Owner
A4	Female	30	1/12	Cosplay Designer
A5	Male	33	1	Defense Contractor
A6	Male	28	4	Software Engineer
B1	Male	28	1	Design Student
B2	Other	23	2	CS Student
B3	Male	26	8	CS Student
B4	Male	33	2	CS Student

Method

The workshops consisted of four phases: discussion (~20 minutes), training (~20 minutes), think-aloud modeling (~60 minutes), and review (~20 minutes). Audio recording devices were distributed throughout the room to capture the discussion and participants' thoughts during the various stages. Researchers also took photographs and videos of the participants during the modeling stage. We asked participants about their expertise with 3D modeling, their goals, previous struggles, and prototyping/ modeling habits.

Prior to attending the workshop all participants were given links to Fusion360's tutorials with notes on which tutorials would be most valuable for the workshop. Six of ten participants used these tutorials, and two participants had pre-existing experience with Fusion360. A3 and B3 were the remaining participants without any Fusion360 experience prior to attending the workshops, and they had more CAD experience than most other participants.

Workshops had three phases – training, modeling and review. During the *training* phase, a researcher demonstrated the interfaces of PARTs and Fusion360 by running through the simple example of joining two blocks with a bolt and nut. The PARTs demonstration included: using an FGO from the library (fastener), checking *assertions* (from the bolt), and invoking integration (to cut the bolt hole). The researcher also gave quick descriptions of the other FGOs available in the library. The Fusion360 demonstration showed the same task with only the standard features of Fusion360.

During the *modeling* phase, participants were given a modeling goal, example objects, and measurement tools. While completing their tasks, researchers walked around the space asking participants to describe their actions. Participants were allowed to speak to one another and ask researchers for assistance with the Fusion360 and PARTs UI or for clarification on the design task. Researchers did not give advice on how participants could complete the task, but merely supported using the tools.

During the *review* phase, participants gathered into a group to discuss their experience. They were prompted to discuss how the PARTs framework differed from their previous experience with CAD tools, what criticisms of the tool they had, and if it affected their modeling behavior. Participants in the later workshops were also asked to compare their experiences using FGOs *versus* standard geometry.

Modeling Tasks

The first workshop tested the usability of PARTs by designers. Participants were tasked with creating a cup-holder using PARTs. Modelers had access to all assertions and *integrators*, but no pre-existing FGOs to reuse in their design. Successful completion of the task required that the resulting model fit the provided cup, and that the holder was stable when the cup was filled.

The second workshop focused on modelers, comparing PARTs to Fusion360. Participants were asked to reuse models to create cup-holders that could be mounted on a bike. They were placed in two conditions, *Fusion360* (alone) or *PARTs* (Fusion360 with PARTs). Condition ordering was randomly assigned to ensure that participants were not primed by the first task to perform better in the second. In the *Fusion360* condition, we provided parameterized models based on a cup-holder (T:2271973) and bike-mount (T:1012573) found in our Thingiverse survey. In the *PARTs* condition, we provided a parameterized cup-holder and bike-mount FGO through the PARTs library. Participants could also use any Fusion360 capabilities they wished.

The necessary tasks participants had to complete in both conditions to create a successful model were: correctly parameterize the cup-holder, bike-mount, and fastener to fit the corresponding real world objects; position the cup-holder and bike-mount components; position and cut space for the fastener. If these tasks were successfully completed, researchers attempted to print and test the resulting model by fitting it to the bike and cup filled with water.

Findings

Anecdotes from participants in both studies show a perceived benefit of the PARTs' approach. For example, A1 noted that he often modeled real world objects to help with his modeling process. His process was manual and informal and he appreciated the automated support for capturing these process ideas, saying "it needs to be in your model for the design or simulation to mean anything". A5 commented on the value of sharing such process information with other users: "here's your spec, and you send them around."

The ability to delay integration was also valued, in comparison to a standard process where "you really have to know what you intend to build everything right" [A6]. In A2's words, "It's essentially a 3D version of layers. It's like a non-destructible field on one side and a visual debugger on the other" He was excited by the idea that he could use *integrators* as place holders for changes he wanted to make later, just as he uses layers in photo editing to prepare changes to the final image.

Workshop 1 Outcomes

4 of 6 participants completed the PARTs design task in the first workshop, and 3 of those models were completely stable. All participants used similar design patterns to create their model: each represented their cup as an interference assertion. They also created cut integrators with a copy of the same geometry. The three failure cases diverged when creating a base for the cup holder. Participants A3 and A6 both failed because they spent the majority of their time creating an accurate representation of the cup, requiring more time to learn Fusion360. A4 completed the design task but did not accurately measure the cup, causing a slightly unstable print. Images of the resulting cup holders are shown in Table 4, Row 1.

Workshop 2

We broke the second workshop up into 6 sub task related to setting parameters, positioning, and combining components and finally resulting in a failed model. Participant failure rates are shown in Table 3. Under the *Fusion360* condition, no participant created a printable model, and participants averaged completion of 1.5/6 steps. Under the PARTs condition, B2 and B3 were able to create printable and functional models, and participants averaged 5/6 steps. Within subjects, they improved by an average of 3.5 steps.

All participants in the second workshop voiced frustration or confusion about Fusion360 because it lacked information included in PARTS' assertions: B3 complained about the Fusion360 cup holder model, "I guess that's where the cup goes, I'm not sure how to fit it to this cup". No similar questions arose while using PARTs.

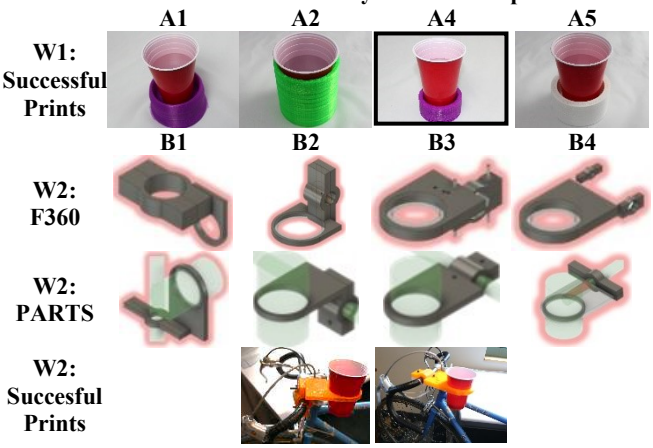
Participants also noted the lack of integration geometry and its associated benefits in Fusion360. B4 asked, "Where do you get that bolt thing in Fusion?" when they assumed the fasteners FGO had been taken from Fusion360; they were frustrated when told that it did not exist and they must find some other way to create the same result.

Parameterization: The main source of failure for participants under the *Fusion360* condition was adjusting the parameters of the pre-existing models to fit the new objects. This was a challenge for two reasons. First, despite being shown parameter definitions in the tutorials, participants could not figure out how to change the dimensions of the models and instead used a scaling feature to adjust the model size as a whole. Second, participants did not think about other effects on the parameters besides the objects dimensions, such as leaving space for the bolt causing each participant to fail at that task. Neither of these were challenges for the PARTs condition, where all participants succeeded in all parameterization tasks. This is likely because parameters are more salient with FGOs. They are displayed at instantiation and broken down into relevant components including clearance and object size.

Table 3. Participant success and failure in modeling tasks broken down by participant and condition (Fusion 360 or PARTS).

Participant	Condition	Fits Cup	Fits Bolt	Fits Bike	Component placement	Fastener Placement	Printable and Functional
B1	**PARTs**	**Pass**	**Pass**	**Pass**	**Pass**	Fail	Fail
	Fusion 360	Fail	Fail	Fail	Pass	Fail	Fail
B2	**PARTs**	**Pass**	**Pass**	**Pass**	**Pass**	**Pass**	**Pass**
	Fusion 360	Fail	Fail	Pass	Pass	Pass	Fail
B3	**PARTs**	**Pass**	**Pass**	**Pass**	**Pass**	**Pass**	**Pass**
	Fusion 360	Pass	Fail	Fail	Fail	Fail	Fail
B4	**PARTs**	**Pass**	**Pass**	**Pass**	**Pass**	Fail	Fail
	Fusion 360	Pass	Fail	Fail	Fail	Fail	Fail

Table 4. Workshop model and print results. Failures are shown with a red halo and the partially functional model from W1 has a black frame (A4). Unsuccessful designs by A3 and A6 are not shown because they could not be printed.



Positioning: B3 and B4 were further challenged by attaching the cup holder and handlebar mount and ran out of time trying to join the two models together. Compounding errors and a rush to complete the tasks in time resulted in B3 and B4 not attempting to place their bolts, and B1 placing the bolt in an infeasible position. These challenges were not completely surmounted with PARTs but all participants were able to position and combine the cup-holder and bike-mount FGOs. This is likely because they did not feel rushed, because the parameterization tasks were easier and quicker. B1 and B4 found it difficult to position the fastener in both conditions. These participants struggled with Fusion360's move command, which is used in both conditions.

Discussion

Participants found the following features of PARTs most accessible: adjusting parameters, integrating models, and making functional changes to another modeler's design. While PARTs helped participants complete their task more effectively, it was not without frustrations. Participants complained that assertions were only occasionally helpful and wanted to turn the visualizations off. Since these workshops, we have made control of assertion visualization and instantiated FGO parameters more salient.

There are some limitations to the second workshop that may affect our results. For instance, participants struggled to contextualize the Fusion360 models, which may not be true if they had discovered them online, as they would in the real world. Further, users with more Fusion360 experience may have shown less of a difference between conditions, or had we given participants more time they may not have rushed and compounded their errors.

CONCLUSIONS AND FUTURE WORK

3D printing enables users to fabricate new objects. However, creating geometry that will function in the real-world is difficult, and so many users print models that others have created. Unfortunately, as shown in our workshop, non-experts can find it difficult to customize such models, even when they are carefully parameterized. Standard tools do not explicitly express design intent, leaving a gap between form and function.

PARTs is at its essence one of the first 3D modeling tools to support end-user programming concepts, thus helping to bridge this gap. PARTs brings design intent to the forefront of the design process. With functional geometry, designers can create reusable designs that capture design intent and support future modelers.

In the future, we hope to build on lessons learned and explore how PARTs can support different modelers by creating customized domain specific libraries and implement more complex systems. By building more complex ways of expressing and acting on design intent, we hope to increase the engineering quality of results in parts. At the moment, PARTs is a powerful first step in enabling non-experts to make use of their common practices in a reusable fashion.

ACKNOWLEDGEMENTS

We thank: our reviewers, participants, and HiCapacity Makerspace. This work was funded in part by NSF grants IIS-1718651 and DGE1745016, by DoD Contract No. FA8721-05-C-0003 (or FA8702-15-D-0002) with CMU for the operation of the Software Engineering Institute, a federally funded research and development center, by a Google Faculty Research Award, a gift from Autodesk, and the Distributed Research Experiences for Undergraduates program, a project of the CRA-W and the Coalition to Diversify Computing (NSF BPC-A #1246649).

REFERENCES

1. Mathieu Acher, Benoit Baudry, Olivier Barais, and Jean-Marc Jézéquel. 2014. Customization and 3D Printing: A Challenging Playground for Software Product Lines. In *Proceedings of the 18th International Software Product Line Conference - Volume 1* (SPLC '14), 142–146. https://doi.org/10.1145/2648511.2648526
2. Daniel Ashbrook, Shitao Stan Guo, and Alan Lambie. 2016. Towards Augmented Fabrication: Combining Fabricated and Existing Objects. In *The 2016 CHI Conference Extended Abstracts*, 1510–1518.
3. Hugh Beyer and Karen Holtzblatt. 1997. *Contextual Design: Defining Customer-Centered Systems*. Elsevier.
4. P Blikstein. 2013. Digital fabrication and 'making'in education: The democratization of invention. *FabLabs: Of machines*.
5. A Boating. 1979. *Thinglab-a constraint-oriented simulation laboratory*. SSL-79-3, Xerox PARC, Palo Alto, CA.
6. Alan Borning. 1981. The programming language aspects of ThingLab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 3, 4: 353–387.
7. Erin Buehler, Niara Comrie, Megan Hofmann, Samantha McDonald, and Amy Hurst. 2016.

Investigating the Implications of 3D Printing in Special Education. *ACM Trans. Access. Comput.* 8, 3: 11:1–11:28. https://doi.org/10.1145/2870640

8. Erin Buehler, Shaun K. Kane, and Amy Hurst. 2014. ABC and 3D: Opportunities and Obstacles to 3D Printing in Special Education Environments. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility* (ASSETS '14), 107–114. https://doi.org/10.1145/2661334.2661365
9. Jack B Dennis. 1973. Modularity. In *Software Engineering*. Springer, 128–182.
10. Chang-Xue Feng and Andrew Kusiak. 1995. Constraint-based design of parts. *Computer-Aided Design* 27, 5: 343–352.
11. Christoph M. Flath, Sascha Friesike, Marco Wirth, and Frédéric Thiesse. 2017. Copy, transform, combine: exploring the remix as a form of innovation. *Journal of Information Technology* 32, 4: 306–325. https://doi.org/10.1057/s41265-017-0043-9
12. Sean Follmer, David Carr, Emily Lovell, and Hiroshi Ishii. 2010. CopyCAD: Remixing Physical Objects with Copy and Paste from the Real World. In *Adjunct Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology* (UIST '10), 381–382. https://doi.org/10.1145/1866218.1866230
13. Rajaram Ganeshan, James Garrett, and Susan Finger. 1994. A framework for representing design intent. *Design Studies* 15, 1: 59–84.
14. Michelle Gantt and Bonnie A. Nardi. 1992. Gardeners and Gurus: Patterns of Cooperation Among CAD Users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '92), 107–117. https://doi.org/10.1145/142750.142767
15. Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What Would Other Programmers Do: Suggesting Solutions to Error Messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10), 1019–1028. https://doi.org/10.1145/1753326.1753478
16. Tyson R Henry and Scott E Hudson. 1988. Using active data in a UIMS. In *Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software*, 167–178.
17. Nathaniel Hudson, Celena Alcock, and Parmit K Chilana. 2016. Understanding Newcomers to 3D Printing: Motivations, Workflows, and Barriers of Casual Makers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 384–396.
18. Michael D. Jones, Kevin Seppi, and Dan R. Olsen. 2016. What You Sculpt is What You Get: Modeling Physical Interactive Devices with Clay and 3D Printed Widgets. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16), 876–886. https://doi.org/10.1145/2858036.2858493
19. Jeeeun Kim, Anhong Guo, Tom Yeh, Scott Hudson, and Jennifer Mankoff. 2017. Understanding Uncertainty in Measurement and Accommodating its Impact in 3D Modeling and Printing. In *Proceedings of the 2017 ACM Conference on Designing Interactive Systems*.
20. Fumihiko Kimura and Hiromasa Suzuki. 1989. A CAD system for efficient product design based on design intent. *CIRP Annals-Manufacturing Technology* 38, 1: 149–152.
21. Siniša Kolarić, Halil Erhan, Robert Woodbury, and Bernhard E. Riecke. 2010. Comprehending Parametric CAD Models: An Evaluation of Two Graphical User Interfaces. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (NordiCHI '10), 707–710. https://doi.org/10.1145/1868914.1869010
22. FB Kong, Xin Guo Ming, L Wang, XH Wang, and PP Wang. 2009. On modular products development. *Concurrent Engineering* 17, 4: 291–300.
23. Yuki Koyama, Shinjiro Sueda, Emma Steinhardt, Takeo Igarashi, Ariel Shamir, and Wojciech Matusik. 2015. AutoConnect: Computational Design of 3D-printable Connectors. *ACM Trans. Graph.* 34, 6: 231:1–231:11. https://doi.org/10.1145/2816795.2818060
24. Vladislav Kreavoy, Dan Julius, and Alla Sheffer. 2007. Model Composition from Interchangeable Components. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (PG '07), 129–138. https://doi.org/10.1109/PG.2007.43
25. Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. 2011. Converting 3D Furniture Models to Fabricatable Parts and Connectors. In *ACM SIGGRAPH 2011 Papers* (SIGGRAPH '11), 85:1–85:6. https://doi.org/10.1145/1964921.1964980
26. Ghang Lee, Charles M Eastman, Tarang Taunk, and Chun-Heng Ho. 2010. Usability principles and best practices for the user interface design of complex 3D architectural design and engineering tools. *International Journal of Human-Computer Studies* 68, 1–2: 90–104.
27. Ming Li, Frank C Langbein, and Ralph R Martin. 2010. Detecting design intent in approximate CAD models using symmetry. *Computer-Aided Design* 42, 3: 183–201.
28. Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. 2014. Build-to-last: strength to weight 3D printed objects. *ACM Transactions on Graphics (TOG)* 33, 4: 97–10.
29. Thomas Ludwig, Oliver Stickel, Alexander Boden, and Volkmar Pipek. 2014. Towards sociable technologies: an empirical study on designing appropriation infrastructures for 3D printing. In *DIS '14:*

Proceedings of the 2014 conference on Designing interactive systems.
30. Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. 1990. User-tailorable Systems: Pressing the Issues with Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '90), 175–182. https://doi.org/10.1145/97243.97271
31. Satoshi Matsuoka, Shin Takahashi, Tomihisa Kamada, and Akinori Yonezawa. 1992. A general framework for bidirectional translation between abstract and pictorial data. *ACM Transactions on Information Systems (TOIS)* 10, 4: 408–437.
32. Jarkko Moilanen, Angela Daly, Ramon Lobato, and Darcy Allen. 2014. Cultures of sharing in 3D printing: What can we learn from the licence choices of Thingiverse users?
33. Catarina Mota. 2011. The rise of personal fabrication. In *C&C '11: Proceedings of the 8th ACM conference on Creativity and cognition*.
34. Don Norman. 1988. *The Psychology of Everyday Things*. Basic Books, New York, NY, USA.
35. Lora Oehlberg, Wesley Willett, and Wendy E. Mackay. 2015. Patterns of Physical Design Remixing in Online Maker Communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (CHI '15), 639–648. https://doi.org/10.1145/2702123.2702175
36. Dan R Olsen Jr and Kirk Allan. 1990. Creating interactive techniques by symbolically solving geometric constraints. In *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, 102–107.
37. Colin Potts and Glenn Bruns. 1988. Recording the reasons for design decisions. In *Software Engineering, 1988., Proceedings of the 10th International Conference on*, 418–427.
38. Valkyrie Savage, Sean Follmer, Jingyi Li, and Björn Hartmann. 2015. Makers' Marks: Physical Markup for Designing and Fabricating Functional Objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (UIST '15), 103–108. https://doi.org/10.1145/2807442.2807508
39. R Shewbridge, A Hurst, and S K Kane. 2014. Everyday making: identifying future uses for 3D printing in the home. *ACM Conference on Designing Interactive Technology*.
40. Ivan E Sutherland. 1964. Sketchpad a man-machine graphical communication system. *Transactions of the Society for Computer Simulation* 2, 5: R–3.
41. Robert W Taylor, Allan Heydon, and Greg Nelson. 1994. The Juno-2 constraint-based drawing editor. In *Technical Report 131a, Digital Systems Research*.
42. Juraj Vanek, Jorge AG Galicia, and Bedrich Benes. 2014. Clever support: Efficient support structure generation for digital fabrication. In *Computer graphics forum*, 117–125.
43. AR Venkatachalam. 1994. Automating manufacturability evaluation in CAD systems through expert systems approaches. *Expert Systems with Applications* 7, 4: 495–506.
44. Haijun Xia, Bruno Araujo, Tovi Grossman, and Daniel Wigdor. 2016. Object-Oriented Drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16), 4610–4621. https://doi.org/10.1145/2858036.2858075
45. Xiang "Anthony" Chen, Jeeeun Kim, Jennifer Mankoff, Tovi Grossman, Stelian Coros, and Scott E. Hudson. 2016. Reprise: A Design Tool for Specifying, Generating, andCustomizing 3D Printable Adaptations on Everyday Objects. In *Proceedings of the 29th Annual ACM Symposium on User Interface Software and Technology*. https://doi.org/10.1145/2984511.2984512
46. Brad Vander Zanden, Brad A Myers, Dario Giuse, and Pedro Szekely. 1991. The importance of pointer variables in constraint models. In *Proceedings of the 4th annual ACM symposium on User interface software and technology*, 155–164.
47. *Tinkercad.com Features | Tinkercad*. Retrieved March 21, 2016 from https://www.tinkercad.com/about/features
48. Interference Check | 3D CAD Solutions | SOLIDWORKS. Retrieved April 4, 2017 from http://www.solidworks.com/sw/products/3d-cad/interference-check.htm
49. Start a solid body using basic shapes. Retrieved August 29, 2017 from http://help.autodesk.com/view/fusion360/ENU/?guid=GUID-9DECFD77-D3AF-4EA8-98D6-CA8A647E5BA4
50. Design Intent | Fusion 360 | Autodesk Knowledge Network. Retrieved August 29, 2017 from https://knowledge.autodesk.com/support/fusion-360/learn-explore/caas/CloudHelp/cloudhelp/ENU/Fusion-Form/files/GUID-CE02E095-6A90-4DCC-B89F-4D70185BC69D-htm.html
51. Learning | How to create components. Retrieved August 29, 2017 from http://help.autodesk.com/view/fusion360/ENU/?guid=GUID-5966BF6B-4135-49B3-B5BF-29A6577C9E72
52. 2016 SOLIDWORKS Help - Hole Wizard Overview. Retrieved September 13, 2017 from http://help.solidworks.com/2016/english/solidworks/sldworks/c_hole_wizard_overview.htm
53. Smart Components and Smart Fasteners | SOLIDWORKS. Retrieved August 30, 2016 from http://www.solidworks.com/sw/products/3d-cad/smart-components-and-smart-fasteners.htm