ABSTRACTION VERSUS CASED-BASED
A COMPARATIVE STUDY OF TWO APPROACHES TO SUPPORT PARAMETRIC DESIGN

PARAMETRIC MODELLING PERFORMANCE: DATA / CRITERIA

<table>
<thead>
<tr>
<th>DATA</th>
<th>OUTPUT MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROGRAMMING ALGORITHM</strong></td>
<td><strong>MODEL COMPLEXITY</strong></td>
</tr>
<tr>
<td>Algorithm complexity</td>
<td>Modelling Speed</td>
</tr>
<tr>
<td>Algorithm Variety</td>
<td></td>
</tr>
<tr>
<td>Algorithm Novelty</td>
<td></td>
</tr>
</tbody>
</table>

ABSTRACT

This paper describes an experimental study testing two approaches to support the learning and use of parametric modeling tools in architecture. The implementation of programming languages, both textual and visual, is often frustrating and causes difficulties for both novice and advanced users. This study explores two methods to reduce the barriers of using programming and potentially improve modeling performance through utilizing existing parametric design solutions. The approaches have been proposed as a means of accessing and reusing existing parametric design knowledge. The first is the reuse of abstract algorithmic ‘patterns’. The second approach is the reuse of parametric solutions from specific design cases (Case-Based Design). This paper outlines the principles and methods of ‘abstract’ versus ‘case-based’ approaches to reuse parametric solutions, and focuses on the results of their practical implementation. Results of this study illustrate how the reuse of abstract or specific programming solutions can aid designers and architects in learning and using parametric modeling tools more effectively.
INTRODUCTION

This paper investigates two approaches to supporting the learning and use of parametric design systems in architecture. The research focuses on knowledge reuse strategies. It explores methods to reduce the barriers to the use of programming in design and potentially improve modeling performance.

Parametric rules and strategies constitute the core of parametric design systems. They are operated through symbolic (scripting) or analogue (visual) programming languages, which are used as the means to actualize an idea-to-form translation (Mitchell 1978). In spite of the fact that the logics of human and computer translations do not follow the same patterns, the paradigm of applying the parametric design principles (variables, arithmetic, data structures and logical operations) to one’s idea seems rather easy to grasp: you compose a form-making algorithm and software generates a form. However, the implementation of programming can be frustrating and cause many difficulties for both novice and advanced users.

The use of parametric modeling tools requires an algorithmic, ‘step-by-step program’ type of design thinking. Fundamentally, programming logic does not relate to conventional design approaches in architecture, such as hand sketching, building physical models and manual CAD modeling. Visual programming software such as Rhino with Grasshopper or Generative Component’s Symbolic Diagram, significantly reduces barriers to the adoption of parametric modeling within architectural design (Celani and Vaz 2012). However, even with these apparently more accessible modeling methods, for novice and experienced users alike, finding the right algorithmic function and the correct syntax for the programming language is far from being resolved. While a number of alternative methods to improve and aid programming in design are being currently explored and developed, this research investigates the potential of approaches to the reuse of existing algorithms.

REUSE OF ABSTRACT SOLUTIONS VERSUS SOLUTIONS FROM SPECIFIC DESIGN CASES

Two different approaches have been proposed as a means of accessing and reusing existing parametric design knowledge. These are relevant not just to novice users of parametric design tools, but are also proposed as a means of reducing the frustrations of experienced users. The first approach is the reuse of abstract solutions to a design problem. An example of this approach is Robert Woodbury’s patterns for parametric design (Design Patterns) (Woodbury 2010). These patterns were developed to assist designers with structuring their own programming solutions on an abstract level. Woodbury states that designers who use parametric modeling tools tend to create algorithms anew, rather than reuse them (ibid). He argues that designers can make their designs much more effective by utilizing re-usable abstract parts (Design Patterns). The key concept of Design Patterns lies in the reuse of design knowledge (Alexander et al. 1977). Instead of solving each new problem individually, architects can reuse the patterns successfully implemented in the past (Gamma et al. 1995). Design Patterns are generalized reusable solutions, described with a high level of abstraction; and documented in such a way that they are broad enough to apply to a range of different design contexts. The DP (Design Patterns) approach was tested through the preparation of visual programming examples of each of the Woodbury ‘patterns’ (Woodbury 2010).

The second approach to accessing and reusing parametric design knowledge is reuse of specific programming solutions following case-based reasoning principles (Kolodner 1993). Case-Based Reasoning (CBR) is a problem solving approach, which utilizes specific knowledge from previous cases, instead of making assumptions based on generalized relationships between a description of a problem and conclusions (Aamodt and Plaza 1994). In CBR, a new problem is solved by finding and reusing an existing solution from a similar case from the past. There is evidence that when humans solve new problems, they predominantly rely on specific, previously encountered situations (Ross 1989). The CBD (Case-Based Design) design approach was tested through an online case-base of visually represented parametric models and corresponding downloadable programming algorithms developed specifically for this research.

While the methodology and principles of ‘abstract’ and ‘case-based’ solutions adaptation differ, both approaches seek to make reuse of parametric design knowledge more effective. The core of this idea is that architectural parametric design is not properly an invention and the creation of something absolutely new, but is rather a process of rediscovery (Terzidis 2006). Naturally, this rediscovery can be directly drawn from existing parametric CAD knowledge, inventions and solutions. It recognizes that it is highly possible that someone, somewhere really did already invent the wheel you are about to reinvent (Mann 2005).
RESEARCH FRAMEWORK & METHODOLOGY

This research was designed as a comparative study between three test groups:

1) a group which used no specific approach to the reuse of parametric design knowledge;
2) a group reusing abstract parametric solutions (Design Patterns (DP)); and
3) a group reusing solutions from specific design cases (Case-Based Design (CBD)).

The approaches were tested in a series of parametric modeling workshops for architects, and for landscape and interior architects who use parametric computer-aided modeling tools in their designs. Participants recruited to participate in the experimental part of the study are a diverse group of students, and practicing architects, with no age restriction, but a minimum one year experience in design. A total of 125 people participated in the study providing sufficient numbers within each group to permit rigorous studies of the statistical significance of the observed differences. Continuous variables were compared using the t-test (for two test groups), and ANOVA (for three test groups); binary data was compared with the chi-square test (all statistical testing was done using SPSS).

The study was organized in the form of two-day parametric modeling workshops. Each workshop offered an introduction to algorithmic computational design using Grasshopper (Grasshopper3D 2014) for Rhinoceros (Rhino3D 2014). On each day, participants were given one design assignment, which they were to develop on their own, after an introductory series of exercises focused on familiarization with the code and with the approach to parametric design reuse. Participants modelled and submitted their designs within a two-hour period. The collected data consisted of submitted 3D models, programming definitions and survey results. The 3D Rhino models were used to calculate the level of complexity of each model. The Grasshopper definitions were used to measure the complexity of each programming algorithm and to determine the explored solution space of each algorithm (Figure 1).

In the diagram (Figure 1) the image, labelled as ‘Data’, illustrates an example of a programming algorithm (box-and-wire diagram made in Grasshopper for Rhino3D) on the left. The image on the right illustrates the output 3D model that is generated by the programming algorithm (definition). The bottom row shows the respective ‘Criteria’ groups such as the ‘Algorithm Complexity’ and the ‘Explored Solution Space’, which were used to evaluate this programming algorithm. The explored solution space is determined by the variety and novelty of a programming solution (Shah, Smith and Vargas-Hernandez 2002). Variety refers to how many different programming components each algorithm has. Novelty evaluates how unusual (less frequently used at the group level) each programming component is (Ibid). The ‘Model Complexity’ criteria derive from the output model. The methodology for measuring the complexity of the output models, was informed by geometrical, combinatory and dimensional complexity criteria for model classification–Shape Grammars (Forrest 1974).

Questionnaires helped to determine the typology of programming difficulties and the number of reused algorithms. They sought feedback from workshop participants on the levels of satisfaction with the design outcome and their motivation to use parametric modeling systems in future. The participants also provided data regarding their design objectives, their ability to model the original design idea and the degree of change in the design due to programming difficulties.

The aesthetic and design qualities of the models were not judged directly. However, this issue was addressed indirectly. Each participant was asked to indicate his or her design intentions and, reflecting on the design outcome, evaluate the degree of satisfaction with the produced model. This strategy also provided insight into what each person intended versus what was actually achieved.

SHIFT IN THE DESIGN OBJECTIVES AND PARAMETRIC MODELING PERFORMANCE

The hypothesis being tested, is that the reuse of abstract and case-based parametric solutions can help designers and architects to overcome barriers associated with programming and can improve parametric modeling performance. One of the most significant differences, which was observed and confirmed by statistical testing, is the major shift in the design objectives caused by the use of approaches. The differences in objectives manifest themselves when designers gain more experience in parametric modeling (on the second day of the workshop). Those designers who reused abstract solutions (Design Patterns group) were more focused on experimentation, exploration of parametric form-making and trying out new programming logics and components (Design objective: proportion who reported their objective as ‘to experiment with parameters’, Day 2, No Approach (NA) group 12 per cent, Design Patterns (DP) group 40 per cent, Case-Based Design (CBD) group 8.5 per cent, p-value = 0.000) (Figure 2). Those who reused parametric solutions from specific design cases (Case-Based Design group) were more committed to realize the originally sketched design ideas and were less interested in explorations and experimentation (Design objective: ‘to achieve what I originally sketched’, Day 2, NA group 48 per cent, DP group 60 per cent, CBD group 80.8 per cent, , p-value = 0.012) (Figure 2).
The shift in design objectives and modeling priorities appeared to have a significant influence on the design process and, as a result, on the final design output. The group who reused abstract solutions (DP group) was less committed to a particular design goal. Participants in this group were more likely to experiment and try alternative options of programming logic and components, which in turn has influenced the way designers created their programming algorithms. Analysis of the programming algorithms showed that those who reused abstractions had a significantly greater explored solution space of algorithms, compared to the group who reused specific design solutions (Algorithm Variety/Day 1 (Mean Value): NA 12.4, DP 15.3, CBD 12.8; p-value = 0.008).

Statistical testing indicates that designers who used case-based reasoning while developing their parametric solutions tended to focus on modeling a particular design outcome (Figure 2). When browsing the online case-base, these workshop participants predominantly used key words associated with already familiar (used in the past) programming components, rather than using abstract key words, thus reducing the likelihood of developing alternative programming solutions.

The evidence suggests that the use of case-based reasoning in parametric design will most likely decrease the variety of programming components used to create parametric models. Designers who use CBD also tend to produce less novel (more typical) programming solutions (Algorithm Novelty/Day 2 (Mean Value): NA 50.8, DP 53.7, and CBD 43.6). However, it should be noted, that while the CBD group did use a substantially smaller range of programming
components and developed less novel programming solutions compared to both DP and no approach groups, they reported higher overall satisfaction with the design model and their ability to accomplish their design objectives than those who used the abstract approach (Figure 3) and (Figure 4).

CHANGE IN MODELING SPEED/MODEL COMPLEXITY

As the participants spent the same amount of time on the development of their design models in each workshop, it has been assumed that higher model complexity indicates greater modeling speed. The shift in design strategies caused by the use of abstract and case-based parametric solutions had a significant effect on the complexity of produced designs. Designers who reused specific programming solutions (CBD group) were likely to develop less complex output models, compared to both the abstract (DP) and no approach groups (Model Complexity/Day 2 (Mean Value): NA 13.94, DP 14.1, CBD 12.74; p-value = 0.031) (Figure 3 & 4). It would appear that the ‘abstract’ group’s greater interest in experimenting with forms and parameters produces designs less restrained by the limitations of the original design concept.

Designers who reuse particular programming solutions, seem to be more focused on modeling the form that they originally sketched, even though this might prove to be time-consuming. It is interesting that the No Approach workshop group were like the DP group in that they showed greater readiness than the CBD group to change their initial concepts, and to develop and experiment with their designs.

Figure 3 shows three designs from Day 1 and one from Day 2 for the DP group and (Figure 4) shows two designs from Day 1 and two from Day 2 for the CBD group (indicated by the suffix to the number code. The complexity scores - Shape Grammars (Forrest 1974) – are typical for each group, with the DP groups of significantly higher complexity. The ‘satisfaction with the output’ and ‘ability to model original idea’ ratings from the participants show a much greater satisfaction for the CBD group.

There is likely another reason that the CBD group participants might be slower than the abstract and No-Approach groups: it is related to the time spent by users accessing the case-base examples looking for programming sequences that allow them to generate the form they originally sketched. Analysis of the screen recordings indicates that participants who reuse solutions from the case-base tend to spend a considerable amount of time browsing the case-base. Designers rarely reused the first solution from the case-base. Instead, they tended to compare several design options, before...
deciding which solution they actually wanted to reuse. This search process can take a considerable amount of time, which inevitably slows down the overall speed of parametric modeling. Reuse of abstract solutions in this case has an advantage. Hypothetically, once designers and architects grasp the idea of a design pattern, they do not have to relearn it each time they implement it in a new design problem. Learning why and how to use a particular abstract solution is a one-time operation. In theory, when designers know a design pattern they might be expected to reapply it to a new design task straight away. Designers who reuse specific solutions are likely to search the case-base of algorithms every time before they chose to reuse (copy/modify) (Woodbury, 2011). The ‘modify’ part of this copy/modify approach is very important, as in most cases each reused solution has to be adapted to suit the new design context—to achieve the original sketch design outcome.

Correlational analysis was used to study the reasoning of the designers in each group. Higher complexity levels of the output models and of the programming algorithms are perceived positively by those who reused abstract solutions. The more complex the design models that these DP participants produced, the higher is their satisfaction with the output (correlation coefficient .463). Those DP designers, who managed to develop more complex programming algorithms also found the DP approach more helpful (correlation coefficient .417). Modeling and programming algorithm complexities are seen by these designers in a positive light.

In contrast to the abstract DP group, designers who reused parametric solutions from specific cases (CBD group) preferred to avoid complexity and tended to settle for the more simple programming algorithms. On both workshop days ‘algorithm complexity’ has a negative correlation (correlation coefficients -.362 / -.378) with ‘satisfaction with the design outcome’. When CBD group participants managed to come up with more simple programming solutions, they were apparently more satisfied with the outcome.

In summary, those who reuse specific solutions see complexity in a negative light, which is the exact opposite of the thinking of the group who reused abstract.

OVERCOMING BARRIERS ASSOCIATED WITH THE USE PROGRAMMING

Many designers find it difficult to integrate algorithmic thinking and programming into the design process (Woodbury 2010). Understanding and learning the programming framework syntax rules can be very frustrating, especially to novel users (Celani and Vaz 2012). Research on learning barriers in programming systems carried out by Ko et al. identified six types of most reoccurring barriers: design, selection, coordination, use, understanding, and information (Ko, Myers and Aung 2004). This study tested whether the reuse of abstract and specific parametric solutions can help designers and architects to overcome programming barriers. The participant designers were asked to indicate the overall amount of difficulties that they had while developing their design assignments and also to specify what type of difficulty it was.

Analysis of their responses identified the five most common categories of difficulty:

- Idea-to-algorithm translation (design barriers, figuring out how to get from a sketched idea to a programming algorithm, which generates a model);
- Problems with specific components (use barriers, when participants knew which programming component they needed, but struggled with how exactly to use it);
- Knowing what programming component to use and when (selection barriers);
- Logic Connections (coordination barriers, what is the correct sequence of programming logic, for example should ‘vector’ go before or after ‘move’); and
- Valid Parameters, unexpected errors (use and understanding barriers, for example, incorrect inputs or domains of numbers).

The diagram in (Figure 5) illustrates the first of these and one of the most common difficulties for novice and experienced users of parametric modeling tools: “Idea-to-Algorithm Translation” (Day 1/2, NA 44.9-42.8 per cent, DP 53.3-60 per cent, CBD 48.9-53.2 per cent). Even on the second day of the workshops when participants were more experienced in parametric modeling, the number of issues with translation of a design idea into a programming algorithm was still very high.

The second most common type of difficulty (Figure 5) was problems with actual implementation of a particular programming component. The reuse of solutions from the case-base proved to be an effective approach to overcome these types of difficulties. There are significantly less problems with particular programming components in the CBD group, compared to both the DP and the no approach group (Day 1: NA 44.8 per cent, DP 33.3 per cent, CBD 21.3 per cent, p=0.049, Day 2: NA48.9 per cent, DP 43.3 per cent, CBD 23.4 per cent, p=0.029).

Overall analysis of these difficulties shows that reuse of abstract solutions is an effective method to help designers reduce difficulties associated with use of parametric modeling tools (Figure 5). The DP group participants had significantly less programming difficulties...
compared to both the CBD and no approach groups (Day 1 (five point scale, Mean Value): NA 2.88, DP 2.37, CBD 2.91, F = 3.414, p = .036, Day 2: NA 2.71, DP 2.10, CBD 2.53, F = 6.200, p = .003).

Nevertheless, the case-based approach did help to overcome certain types of difficulties (Figure 5). As there were very few workshop participants with significant levels of experience with parametric modeling systems, it seems reasonable to conclude that in the initial stages of learning and using of these systems, the use of abstract solutions, such as Design Patterns, helps reduce the overall amount of difficulties. Abstractions help novices to better comprehend in principle the ‘when’, ‘what’ and ‘how’ a design problem can be solved. Designers themselves appear not to realize how helpful the use of abstractions (Design Patterns) is. When asked ‘how easy to implement’, ‘helpful’ and ‘intuitive’ each approach is, the Case-Based Design approach was identified by designers as significantly more intuitive, helpful and easy to use.

The reuse of abstract solutions in parametric design helps to reduce many of the barriers which designers and architects have when they use parametric modeling systems. It motivates designers and architects:

- to experiment more;
- to explore new programming solutions and commands; and
- to produce algorithms and output models with higher levels of complexity.

The reuse of parametric solutions from specific cases, is an effective tool to reduce difficulties associated with the implementation of specific programming components and commands. It is intuitive, helpful and easy to use; promotes the development of more simple and less novel design solutions; and motivates designers:

- to focus on realizing the initial design ideas; and
- to be less invested in exploration of alternative solutions and experimentation with new programming logics.

Further, planned research work focuses on the long-term effects of these two approaches and investigation of the effect of the approaches on professionals and people working with programming. It is also planned to develop and test the hybrid (abstract and case-based) methodology, focusing on the learning and overcoming barriers of programming in design.

### SUMMARY OF KEY FINDINGS

Although the problem of finding appropriate algorithms is common to novice and experienced users of parametric design methods, the focus in these workshops on novices limits the conclusions thus far to: for learners, the use of a systematic approach to the reuse of parametric design solutions is more beneficial than having no approach.
REFERENCES


Mann Darrell. 2005. “Someone, somewhere really did already invent the wheel you’re about to re-invent”, HKIVM 7th International Conference.


Woodbury, Robert. 2010. “Elements of parametric design”.

IMAGE CREDITS

Figure 1. Parametric Modeling Performance.
Figure 2. Typology and distribution of design objectives.
Figure 3. Examples of models, Design Patterns group.
Figure 4. Examples of models, Case-Based Design group.
Figure 5. Programming difficulties.


JULES MOLONEY’s research, creative practice and teaching span the fields of architecture, kinetic art and virtual environments. This hybrid practice is predicated on an equal mix of theory and intuitive engagement with digital technology. His book Designing Kinetics for Architectural faCADes: State Change (Routledge 2011) articulates a framework for the design of kinetics from first principles. He is currently Professor of Interdisciplinary Digital Design and Head of School at Victoria University of Wellington.