Abstraction versus Case Based: A Comparative Study of Two Approaches to Support Parametric Design

Anastasia Globa, Michael Donn and Jules Moloney
Abstraction versus Case Based: A Comparative Study of Two Approaches to Support Parametric Design

Anastasia Globa, Michael Donn and Jules Moloney

We describe an experimental study testing the reuse of design knowledge as a method to support learning and use of parametric design in architecture. The use of parametric design systems and programming environments offer architects new opportunities, providing a powerful means to create geometries and allowing dynamic design exploration, but it can also impose substantial challenges. The proposition tested in this study is that the reuse of design knowledge can improve architects’ ability to use parametric modelling, and reduce the barriers to using programming in a design context. The paper explores and compares two approaches as a means of accessing and reusing existing design solutions: the reuse of abstract parametric ‘Design Patterns’ [1]; and secondly 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 through the statistical analysis of a comparative study involving 126 designers. In conclusion, it is proposed the outcomes from this study can be applied to inform the methodology for introducing parametric design in architecture and design disciplines.
I. INTRODUCTION

A comparison with practices in software engineering suggest the design disciplines could benefit from ‘reuse’ methods, which may help to overcome programming barriers that currently limit the uptake of parametric modelling software. We describe an experimental study that has evaluated the reuse of design solutions as a support method for learning and using parametric design software in architecture. The focus of the study is to test whether the reuse of design knowledge can improve designers’ ability to understand and use parametric modelling systems effectively, and to help them to overcome difficulty associated with the implementation of programming languages.

An increasing number of designers choose to learn and use parametric modelling methods [2], attracted by the capacity to explore multiple output models by relatively simple changes to the values of parameters and their interrelationships. Algorithmic 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 actualise an idea-to-form translation [3]. Another factor in the growing interest in parametric modelling techniques, is that the use of programming offers a means to overcome some of the limitations of predefined commands and interfaces of typical CAD software. It allows designers more freedom and flexibility in the face of software constraints. While the use of parametric modelling software provides architects and designers with new opportunities, it can also impose considerable challenges [2]. Many designers find it difficult to integrate algorithmic thinking and programming techniques into the design process [1]. The principles of human and computer reasoning do not follow the same patterns, hence it is not easy for some people to use programming algorithms when translating their design ideas. This research compares two methods for reducing these difficulties: the reuse of abstract solutions [1] and deriving solutions from a database of specific design cases.

The reuse of programs, algorithms and codes (software artefacts) is an important part of programming practice and research in the field of software design [4]. Typically every reuse technique involves selection, specialisation and integration of artefacts, though the degree of involvement may vary depending on the reuse approach. The objective of the reuse of programming artefacts is usually to reduce time and effort required to design systems [4]. Software engineers and architects using parametric modelling systems share similar challenges. However, the systematic reuse of design solutions is not a part of algorithmic design practice in architecture, and software programming practices offer architects some useful lessons [1, 5].

The use of parametric modelling tools requires a procedural approach to design. There is a distinct gap between traditional design principles and algorithmic modelling methods and rules. Most architects and architectural
students find it difficult to shift from conventional freehand drawing and modelling (including manual CAD modelling) to describing their ideas through the language of algorithms and codes. Traditionally programming has not been a part of the architectural syllabus, but increasingly visual programming methods and disciplines are being explored within architectural and design education. This shift in design education and practice towards new parametric technologies and design approaches is an on-going process, as much as the development of parametric CAD technology itself. Recent studies indicate that some barriers have significantly decreased with the development of such software as Grasshopper and Generative Component’s Symbolic Diagram, which support visual programming [2]. However, even with these more accessible visual programming interfaces, both the algorithmic functions and the syntax of programming cause problems for designers. While software developers work towards improving the characteristics of design systems (making more intuitive and flexible programming languages and interfaces), this study proposes to explore this issue from the perspective of the systematic reuse of design solutions.

2. REUSE OF ABSTRACT SOLUTIONS VERSUS SOLUTIONS FROM SPECIFIC DESIGN CASES

Two different approaches are proposed to test the idea of design knowledge-sharing and the reuse of the solutions. The primary research strategy is to work with two radically different reuse methods that are well established and discussed in the literature. The first is to learn and reuse abstract solutions (Design Patterns) [1], the second is to reuse case-based solutions using a database system.

The abstraction reuse approach is tested using Design Patterns developed by Robert Woodbury [1]. These Design Patterns focus on generalised methods of structuring programming solutions, and address both problems with programming, as well as with solving problems specific to architecture [5]. According to Woodbury, Design Patterns are a theory, which is yet to be tested [1] – one of the ambitions of this research is to evaluate the theory in the context of a systematic comparative study.

The key concept of Design Patterns lies in the reuse of design knowledge [6]. Instead of solving each new problem individually, architects can reuse the patterns successfully implemented in the past [7]. The Design Patterns method has been adapted and tested in various disciplines including the field of object-oriented design (software development). This is particularly relevant, because both software design and parametric design operate using programming languages. It has been suggested that an effective reuse technology implies the use of a high level of abstraction [5, 7, 4]. The idea is that a designer should know ‘what’ the reusable artefacts do rather than ‘how’ they do it. However, there are difficulties associated with the
reuse of abstractions, because in order to use abstract solutions a designer must be familiar with the abstractions prior to the design process, which requires time to study and understand these abstractions [4]. This suggests that for a reuse technique to be effective it should be easier to reuse an existing solution than it is to develop a new one from scratch [4].

Design Patterns are generalised re-usable 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. Woodbury has outlined the following principles of patterns for parametric design [1]:

- Explicit. Others should be able to read (understand) your patterns in your absence.
- Partial: separate solutions to problem parts;
- Problem focused: a pattern should solve a shared problem;
- Abstract. Patterns are abstract and represent a general concept.

Through reference to the works of Alexander [6] and Gamma [7], Woodbury identifies the following structure: each design pattern has to be explained using the ‘Name’, ‘Intent’, ‘Use When’, ‘Why’ and ‘How’ and it should be illustrated by a set of samples (examples) [8]. For example, Design Pattern ‘Reactor’ [1] are described in the following way.

- Intent: ‘Make an object respond to the proximity to the other object’.
- When: ‘Use this pattern … to make an object respond to the presence of another object’.
- Why: Designers often use the metaphor of response, when one part of a design (result) depends upon the state of the other (interactor). For this particular pattern the proximity (reference) factor drives the response.
- How: ‘Connect an interactor to a result through a reference’ [1].

The second approach is the reuse of specific programming solutions, employing case-based reasoning principles [9]. Case–Based Reasoning (CBR) is a problem solving approach which utilises specific knowledge from previous cases, instead of making assumptions based on generalised relationships between a description of a problem and conclusions [10]. In CBR a new problem is solved by finding and reusing an existing solution from a similar case from the past. In other words, in order to solve a new problem one finds a previous situation and reuses the knowledge of its solution in a new context. Case-based reasoning is a cognitive model proposing that thinking by analogy is consistent with natural patterns of problem solving [9]. It is argued that CBR is used by people as a primary mechanism for common reasoning on a daily basis; there is evidence that when humans solve new problems they predominantly rely on specific, previously encountered situations [9]. Research on human cognition shows that people tend to use previous cases as models both when they are novices [11] and when they are experts [12].
Studies on the use of case-based design in architecture indicate that designers can benefit from past cases, by adapting similar design solutions [13]. One of the fundamental strategies in acquiring knowledge is to learn by example. In architecture, examples are design cases, however, there is a fundamental difference between learning by example and case-based reasoning. In CBR, cases ‘are generalised with respect to the context of a specific problem during each problem solving process’ [14]. Traditionally, in the field of design, knowledge has been recorded and formalised as examples of successful design outcomes, rather than generalised in the form of principles [14]. The approaches using case-based reasoning incorporate the following principles [10].

• Identification of a new problem (new case);
• Finding a similar past case (existing solution in a case-base);
• Use this past case to solve (suggest a solution for) a new problem;
• Evaluate your solution and update the case base by learning from your new experience (new solution)

In this study, the CBD (Case-Based Design) approach was tested through an online case-base of visually represented parametric models and corresponding downloadable programming algorithms. These cases, and their illustrations, were developed specifically for this research and were labelled according to the design concept, shape and programming logic.

While the methodology and principles of ‘abstract’ and ‘case-based’ solutions 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 invention and creation of something absolutely new, but is rather a process of rediscovery [15]. Naturally, this rediscovery can be directly drawn from existing parametric CAD knowledge, inventions and solutions. It recognises that it is highly possible that someone, somewhere really did already invent the wheel you are about to reinvent [16].

3. RESEARCH FRAMEWORK AND METHODOLOGY

This research was designed as an experimental comparative study between three test groups: a control group (No Approach - NA); a second group reusing abstract solutions (Design Patterns - DP); and a third group reusing solutions from specific design cases (Case-Based Design - CBD). The approaches were tested in a series of parametric modelling workshops for architects, and landscape and interior architects. Participants recruited to participate in the experimental part of the study were a diverse group of students, and practicing architects, with no age restriction, but a minimum of one year experience in design. A total of 126 people participated in the study providing sufficient numbers within each group to permit rigorous studies of the statistical significance of the observed differences. The term statistically significant is used as a means to indicate the probability of the
results occurring by chance alone. A probability level of 0.05 has been
established as an acceptable level of confidence. The 0.05 level indicates that
there are at least 95 out of 100 chances that the results obtained from the
study sample would be similar, when tested on the entire population.
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 organised in the form of two-day parametric modelling
workshops. Each workshop offered an introduction to parametric design
using Grasshopper [17] for Rhinoceros [18]. On each day, participants were
given one design assignment, which they were to develop on their own. This
was preceded by an introductory series of exercises focused on
familiarisation with the software and the DP and CBD groups were
additionally taught how to use the respective reuse approach. Participants
modelled and submitted their designs within a two-hour period at the end
of each day. 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.

In “Figure 1” the image on the left (labelled ‘Data’) illustrates an example
of a programming algorithm (box-and-wire diagram made in Grasshopper
for Rhino3D). The image on the right illustrates the output 3D model that is
generated by the programming algorithm. The bottom row shows the
respective ‘Criteria’ groups that were used to evaluate this programming
algorithm. For example the ‘Explored Solution Space’ is determined by the
variety and novelty of a programming solution [19]. 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 [19]. The ‘Model Complexity’ criteria are
derived 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
based on Shape Grammars [20].

Questionnaires helped to determine the typology of programming
difficulties and the number of reused algorithms and sought feedback from
workshop participants on the levels of satisfaction with the design outcome
and the motivation to use parametric modelling systems in the 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.

In order to evaluate and compare how each approach influences various
aspects of parametric design, this study identified five criteria.

• parametric modelling performance (ability to effectively use
  parametric modelling systems):
• programming criteria (ability to overcome barriers associated with programming),
• design ideation (ability to realise an idea-to-form translation within the parametric modelling environments);
• motivation (the level of satisfaction with the design output and motivation to use parametric modelling in future) and;
• approach characteristics (easy to use, intuitive and helpful each approach is).

These five criteria formed the evaluation metrics by which this study measured the effect of the reuse of abstract and case-based parametric solutions. The aesthetic and design qualities of the models were not judged directly. However, these issues were addressed indirectly. Each participant was asked to indicate their design intentions and, reflecting on the design outcome, to evaluate the degree of satisfaction with their produced model. This strategy also provided insight into what each person intended relative to what was actually achieved.

4. SHIFT IN THE DESIGN OBJECTIVES AND PARAMETRIC MODELLING PERFORMANCE

This research tested whether the reuse of knowledge (either the reuse of abstract or case-based parametric solutions) can help designers and architects overcome barriers associated with programming and can improve parametric modelling performance. Compared to the control group participants (No Approach), participants in both abstract and case-based reuse approach groups demonstrated improved performance. The
differences in results were statistically significant (at 95% confidence level), including the ways designers’ think and perform; and in what they ultimately produce. The study also revealed differences between abstract and case-based approaches. One of the most statistically significant is the major shift in the design objectives, which manifest when designers gain more experience in parametric modelling. This can be seen in “Figure 2”. It illustrates the measured differences in the design ideation criteria between the abstract and the case-based study approaches on each day of the workshops. For three of these five criteria the differences were statistically significant: for these the p-value is highlighted in black, not grey, and most of these differences showed themselves to be statistically significant on day 2 of the workshops.

![Figure 2: Typology and distribution of design objectives. Grey % bar – No Approach group, Red % bar – Design Patterns (Abstract Parametric Solutions), Pink % bar – Case-Based Design (Specific Parametric Solutions).](image)

TYPE OF DESIGN OBJECTIVES: DESIGN IDEATION CRITERIA

* shown as a percentage of participants who had this particular type of objective

<table>
<thead>
<tr>
<th>DAY</th>
<th>To achieve what I originally sketched</th>
<th>To explore algorithmic form-making</th>
<th>To experiment with parameters</th>
<th>To apply the logics that I have learned</th>
<th>To combine a few solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY 1</td>
<td>40%</td>
<td>51%</td>
<td>28%</td>
<td>24%</td>
<td>6%</td>
</tr>
<tr>
<td>p-value</td>
<td>All Groups</td>
<td>DP/NA</td>
<td>CBD/NA</td>
<td>DP/NA</td>
<td>CBD/NA</td>
</tr>
<tr>
<td>4.0%</td>
<td>0.01</td>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40%</td>
<td>51%</td>
<td>28%</td>
<td>24%</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>DAY 2</td>
<td>48%</td>
<td>50%</td>
<td>28%</td>
<td>30%</td>
<td>5%</td>
</tr>
<tr>
<td>p-value:</td>
<td>DP/NA</td>
<td>CBD/NA</td>
<td>NA/All Groups</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>0.00</td>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48%</td>
<td>50%</td>
<td>28%</td>
<td>30%</td>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>

NA - No Approach Group (Control group)
DP - Design Patterns Group (Re-use of Abstract Solutions)
CBD - Case-Based Design Group (Re-use of Specific Solutions)

* p-values for each pair of groups are calculated only for cases where the comparison between all three groups indicates significant difference (p-value All Groups < 0.05)

Participants of both approach groups were much more likely to explore parametric form-making and to try out new programming logic, compared to the participants of the control group (NA). It should be noted that the group using the CBD approach were also more invested in the investigation of the capacity of parametric modelling (46.8%), as compared to the control group (24.4%); however, the DP group showed the biggest interest ‘To explore algorithmic form-making’ (63.3%). Those who reused parametric solutions from specific design cases (CBD) were more committed to realise the originally sketched design ideas and were less interested in explorations and experimentations. Interpreting these responses, we can argue that those
designers who reused abstract solutions (the Design Patterns group) were more focused on experimenting with parameters.

The shift in design objectives and modelling priorities appeared to have a significant influence on the design process and, as a result, on the final design output. The test group who reused abstract solutions (DP group) were less committed to a particular design goal. This is illustrated in “Figure 3” by two designs from the DP group where the two participants reported a score of 2 (out of a maximum 5) on their ability to model their original design idea. The figure shows the original hand sketch and the output model from their Day 2 DP workshop. These two participants also reported a 4 (out of 5 again) on their ability to find a Design Pattern that fitted their idea and a 4 on their ability to accomplish what they wanted. As shown in “Figure 2”, participants in this group were more likely to experiment and try alternative options of programming logic and components. This in turn has apparently 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 the algorithms, compared to the group who reused specific design solutions.

\[ \text{Figure 3: Examples of sketches (original design ideas) and corresponding output models, designed by participants using Design Patterns. Typical cases where designers have significantly changed their original idea and still reported that they were able to find a Design Pattern(s) that fit and were able to accomplish what they wanted.} \]
Statistical testing indicated that designers who used case-based reasoning while developing their parametric solutions tended to focus on modelling a particular design outcome. This is shown in “Figure 4” by two designs from the Case Based Design (CBD) group where the two participants reported a score of 4 (out of a maximum 5) on their ability to model their original design idea. As a group, the analysis in “Figure 2” suggests they were less interested in exploring different programming options and new strategies. Instead, those who used CBD tended to implement components that they already knew (and which were explained during the workshop tutorials). 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 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 programming solutions. 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 with the abstract approach “Figure 3-6”. These conclusions further confirm other research on the implementation of CBD tools in design where it was reported -  

‘The major disadvantage of the case-based method is that the solution space is not fully explored and as a result, there is no guarantee of an optimal solution’ [9].

5. CHANGE IN MODELLING SPEED / MODEL COMPLEXITY

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 “Figure 5, 6”. It would appear that the ‘abstract’ (DP) group’s greater interest in experimenting with forms and parameters produces designs less restrained by the limitations of the original design concept. Four example designs from this abstract group are shown in “Figure 5”. The suffix to the participant id number shows that three of these are from Day 1 of the workshop, and one from Day 2. The score highlighted in black under each design has been developed as a means of systematically ranking the complexity of the programming algorithm [21]. All four of the participants whose work is illustrated reported high (5 out of 5) satisfaction with their output model, but were far less satisfied with their ability to model their original idea (2 - 3 out of 5).

Designers who reuse particular programming solutions, seem to be more focused on modelling a specific design outcome. “Figure 6” shows four example outputs from this group laid out in the same manner as “Figure 5”. Two of the outputs are from Day 1 of the workshop and two from Day 2. The overall programming complexity of these examples is much lower than for the DP group in “Figure 5”. The four examples in each figure were selected to be clustered close to the average for each approach, but to all have a score of 5 on each workshop participant’s satisfaction with the output model.

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. The CBD group participants were more likely to try and develop a particular programming sequence, which would generate the form that they originally sketched, even though this might prove to be time-consuming.
Figure 5. Examples of models, designed by participants, who used Design Patterns and were able to accomplish what they wanted; explored alternative design options; significantly changed the original idea; and developed more complex programming algorithms and output models.
There is likely another reason that the CBD group participants might be slower in modeling 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

**CBD Group**  
Approach: Re-use of Parametric Solutions from Specific Design Cases

![Examples of models](image)

**Figure 6**: Examples of models, designed by participants, who used Case-Based Design approach and were able to accomplish what they wanted: managed to model the original idea; and developed more simple programming algorithms and output models.

<table>
<thead>
<tr>
<th>209_2</th>
<th>321_1</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Model 209_2" /></td>
<td><img src="image" alt="Model 321_1" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>217_1</th>
<th>313_2</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Model 217_1" /></td>
<td><img src="image" alt="Model 313_2" /></td>
</tr>
</tbody>
</table>

| 45 | 37 |
| PROGRAMMING ALGORITHM COMPLEXITY POINTS | PROGRAMMING ALGORITHM COMPLEXITY POINTS |
| 5 | 5 |
| SATISFACTION WITH OUTPUT MODEL (on scale from 1 to 5) | SATISFACTION WITH OUTPUT MODEL (on scale from 1 to 5) |
| 5 | 4 |
| ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5) | ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5) |

Anastasia Globa, Michael Donn and Jules Moloney
participants who reuse solutions from the case-base, tend to spend a considerable amount of time browsing the case-base and exploring various programming solutions. It was observed that designers rarely reuse the very first solution from the case-base which they chose to probe. Instead, they tend to compare several design options, before deciding which solution they actually want to reuse. Observation of the group that used case-based design shows that the search process for the most fitting specific solution can take a considerable amount of time, which inevitably slows down the overall speed of parametric modelling.

It seems likely that once designers and architects grasp the idea of a design pattern, they do not have to re-learn it each time they implement it in a new design problem. Learning why and how to use a particular abstract solution (design pattern) is a one-time operation. In theory, when designers know a design pattern they might be expected to re-apply 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) [1]. 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 (Design Pattern (DP) group). The more complex the design models that 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). Model and programming algorithm complexity 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 what the group who reused abstract solutions tend to think.
Figure 7: Examples of models, designed by participants of the control group, who used no reuse approach (NA group).
In the No Approach group, minimal dependency was identified between the levels of complexity of algorithmic solutions and participants’ ability to realise their design ideas and their satisfaction with the result (“Figure 7”).

6. OVERCOMING BARRIERS ASSOCIATED WITH THE USE OF PROGRAMMING

Many designers find it difficult to integrate algorithmic thinking and programming into the design process [1]. Understanding and learning the programming framework syntax rules can be very frustrating, especially to novel users [2]. This study tested whether the reuse of abstract and specific parametric solutions can help designers and architects to overcome these barriers. The participants were asked to indicate the overall degree of difficulty they encountered while developing their design assignments and also to specify what these were. Analysis of their responses identified five common types.

- 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 need, 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’);
- Valid Parameters, unexpected errors (use and understanding barriers, for example, incorrect inputs or domains of numbers) [22].

The diagram in “Figure 8” illustrates the degree to which all five of these parameters were a problem for each approach. The length of the pairs of bars either side of the central list of difficulties represents the percentage of workshop participants who reported each difficulty. The most common difficulty for people learning to use parametric modelling tools is immediately clear: ‘Idea-to-Algorithm Translation’ was reported as a problem for 43-60% of workshop participants. Even on the second day of the workshops when participants were more experienced in parametric modelling the number of issues with translation of a design idea into a programming algorithm was still very high. The workshop participants expressed this in a variety of ways:

“You understand the end product, but the way to derive it is confusing and challenges the way you think about your form.”
“Not able to translate concept into script logic.”
“Struggling to find a method to put what I wanted to do into reality.” [21]
**Figure 8** reveals the second most common problems were 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, with significantly less problems with particular programming components when compared to both the DP and the control group. The difference in the average number of participants reporting difficulties in day 2 workshops was the only statistically significant difference observed on these particular criteria (refer to the bottom of “Figure 8”). The top (How Often) portion shows an overall analysis of the number of programming difficulties encountered by workshop participants.

Assigning a score of 1 for no difficulties, a score of 2 for 1-3 difficulties and so on to a score of 5 for 10 or more difficulties produced the three bars to the right for ‘No Approach’, ‘Abstract Approach’ and ‘Case Based Design Approach’. The average score (number of difficulties) on day 1 and on day 2 is significantly less for the reuse of abstract solutions approach.
Reuse of abstract solutions is therefore an effective method to help designers reduce difficulties associated with use of algorithmic modelling tools. The DP group participants had significantly less programming difficulties compared to both the CBD and No Approach groups. Despite this clear difference, it is worth remembering the case-based approach did help to overcome certain types of difficulties.

As there were very few workshop participants with significant levels of experience with parametric modelling 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 to reduce the overall amount of difficulties. Abstractions help novices to better comprehend, in principle, ‘when’ and ‘how’ a design problem can be solved. However, in terms of initial impressions, rather than output produced, designers themselves appear not to realise how helpful the use of abstractions 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.

7. SUMMARY OF KEY FINDINGS

The primary observation to be made is that when learning computational design methods, the use of a systematic approach to the reuse of parametric design solutions is more beneficial than having no approach.

In many aspects, such as the ability to overcome programming difficulties, the reuse of abstract (Design Patterns) solutions is more helpful than the reuse of solutions from a case-base. The use of Case Based Design proves to be mostly effective in overcoming difficulties associated with the implementation of specific programming components and commands.

The reuse of abstract solutions in parametric design helps to reduce the barriers that designers and architects have when they use parametric modelling systems and motivates designers and architects:

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

The reuse of parametric solutions from specific cases (CBD), 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 realising the initial design ideas;
- to be less invested in exploration of alternative solutions and experimentation with new programming logic.

A range of practical lessons were learned throughout the course of this study, testing the reuse of design knowledge as a method to support...
learning and use of parametric design in architecture. From a teaching perspective, the systematic inclusion of Design Patterns and Case-Based reasoning into the learning narrative of programming in architecture and design proves to be highly beneficial. Their inclusion can improve learners’ ability to overcome programming barriers and help to enable a transition to algorithmic design thinking. Since the DP and CBD approaches were tested on novice programmers, the findings of this study can be used to provide the basis for strategic teaching approaches, which utilise the reuse of programming artefacts. In conclusion, we propose that the outcomes from this study can be applied to inform the methodology for introducing parametric design within architecture and other design disciplines [21].

REFERENCES

16. Mann D. “Someone, somewhere really did already invent the wheel you’re about to re-invent”, *HKIVM 7th International Conference*, 2005.
17. http://www.grasshopper3d.com [1-4-2014]

Anastasia Globa¹, Michael Donn¹ and Jules Moloney²

¹Victoria University of Wellington, School of Architecture
Globalnaya@gmail.com, Michael.Donn@vuw.ac.nz

²Deakin University, School of Architecture and Built Environment
Jules.Moloney@deakin.edu.au