Urban codes

Abstraction and case-based approaches to algorithmic design and implications for the design of contemporary cities

Anastasia Globa[1], Jules Moloney[2] and Michael Donn[3]

[1,3]Victoria University of Wellington, [2]Deakin University
globalnaya@gmail.com; jules.moloney@deakin.edu.au; michael.donn@vuw.ac.nz

Abstract. This paper reports on a comparative study that evaluates two approaches to support the learning and use of algorithmic design in architecture, and extrapolates from this to consider applications for the algorithmic design of cities. The study explored two methods to reduce the barriers of using programming and potentially improve design performance. The first is the reuse of abstract algorithmic 'patterns'. The second approach is the reuse of algorithmic solutions from specific design cases (case-based design). Reflecting on this research we outline how our findings discussed in relation to alternate thinking on the use of pattern, might inform a hybrid approach to the algorithmic design of cities.

Keywords: Case-Based, Design Patterns, algorithmic design, Urban Design

1 Introduction

The use of algorithmic design systems and programming environments offer architects immense opportunitiesbut it can also impose considerable challenges [Menges, Ahlquist, 2011]. Fundamentally, programming logic does not relate to conventional design approaches in architecture such as, hand sketching, building physical models and manual CAD modelling.Traditionally, programming has not been a part of the architectural syllabus, but increasingly visual programming has increased uptake in design education and practice.Despite this relative accessibility, a procedural approach is required, which can be frustrating and cause many difficulties for both novice and advanced users [Celani, Vaz, 2012; Woodbury 2010].Moreoverit is not easy for some people to use programming algorithms when translating their idea into form. Many designers find it difficult to integrate algorithmic thinking and programming techniques into the design process [Woodbury, 2010], because the algorithmic logic of the idea-to-form translation introduces novel principles of design thinking [Matcha, 2007].

Algorithmic design is a discipline which belongs to the field of design, including architectural design, interior, industrial and landscape design etc. At the same time, parametric design progresses through the use of programming. Consequently, it equally belongs to the field computer science. The reuse of programs, algorithms and codes (software artefacts) is an important part of programming practice and research in the field of software design [Krueger, 1992]. Typically thesereuse technique involve selection, specialisation and integration of artefacts, though the degree of involvement and the level of abstraction may vary depending on the reuse approach.

2 Testing Design Patterns and Case-Based Design

Thestudywhich forms the bulk of the paperexplores and compares two distinct approaches as a means of accessing and reusing existing design solutions. The first approach is the reuse of abstract algorithmic 'Design Patterns'. The second approach is the reuse of algorithmic solutions from specific design cases (Case-Based Design). The two methods of reusing abstract and case-based knowledge are not new. Over the past few decades the pattern and case-based design approaches have been adopted by educators and practitioners in various fields of design, architecture and software development.While the methodology and principles of 'abstract' and 'case-based' solutions adaptation differ, both approaches seek to make reuse of algorithmic design knowledge more effective. The core of this idea is that design is not properly invention and creation of something absolutely new, but is rather a process of rediscovery [Terzidis, 2006]. Naturally, this rediscovery can be directly drawn from existing algorithmic CAD knowledge, inventions and solutions, because it is highly possible that someone, somewhere really did already invent the wheel you are about to reinvent [Mann, 2005]

2.1 Research set-up

The research was set-up as an experimental comparative study between three test groups: the group using Design Patterns, the group using Case-Based Design and the control group using no approach. A total of 126 designers participated in the study providing sufficient numbers within each group to permit rigorous studies of the statistical significance of the observed differences. The study was organised in the form of two-day parametric modeling workshops using Grasshopper for Rhinoceros. In each of the days, participants were given one design assignment, which they were to develop on their own, after an introductory series of exercises focused on familiarisation with the code and with the approach to design reuse. It was suggested that participants model and submit their designs within a two hour period at the end of each workshop day. The collected data consisted of submitted 3D models, programming definitions and survey results.

2.2 The Level of Abstraction in the Reuse Approaches

To test the reuse of abstract algorithmic solutions in architecture, this study used the thirteen patterns for parametric design, developed and illustrated by Robert Woodbury [2010]. In his book ‘Elements of parametric design’ Woodbury states that designers who use parametric modelling tools tend to create algorithms anew, rather thanreuse them [Ibid]. The idea of design patterns is that instead of solving each new problem individually, architects can reuse the generalised algorithms (patterns) of existing, successfully implemented in the past, solutions [Gamma, Helm, Johnson, Vlissides, 1994]. Patterns refer to the solutions, described with a high level of abstraction. This way design patterns can be individually interpreted depending on a particular design context. In Woodbury’s book and a website dedicated to the patterns for parametric design [Designpatterns, 2014] each of the design patterns is explained using the ‘Name’, ‘Intent’, ‘Use When’, ‘Why’ and ‘How’ and is illustrated by a set of samples (specific solutions), which are shown as a sequence of images.

The Case-Based Design (CBD) approach is based on the reuse of design solutions from specific design cases. In the context of this study the CBD approach refers to the reuse of algorithmic solutions in architecture. This approach was tested using an online data-base system, specifically developed for this study, which contained over one hundred fifty programming solutions. The primary purpose of these re-usable solutions was to help designers and architects to solve their own (similar) design problems [Maher, de Silva Garza, 1997]. In various fields, including architecture and software programming, the use of Case-Based Design approach proved to be an effective method, helping designers and developers to solve problems by reusing previous solutions and experiences [Kolodner, 1991][Aamodt, Plaza, 1994][Riesbeck, Schank, 2013].

3 Results. Decrease of the Programming Barriers

In many aspects, such as for example 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 (Case Based Design). The use of CBD proves to be mostly effective in overcoming difficulties associated with the implementation of specific programming components and commands.

The diagrams in Fig.1 illustratethe overall amount of difficulties (the top diagram)and the typology and distribution of programming difficulties for each approach (the bottom diagram). In the diagram showing typology of difficultiesthe 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 algorithmic modelling tools wasthe ‘Idea-to-Algorithm Translation’. Itwas reported as a problem for 43-60% of workshop participants. Even on the second day of the workshops when participants were more experienced in algorithmic modelling 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 was problems with actual implementation of a particular programming component: 21-49% of participants (Fig.1). The reuse of solutions from the case-base proved to be an effective approach to overcome these types of difficulties. There were significantly less problems with particular programming components in the CBD group, compared to both the DP and the no approach group.

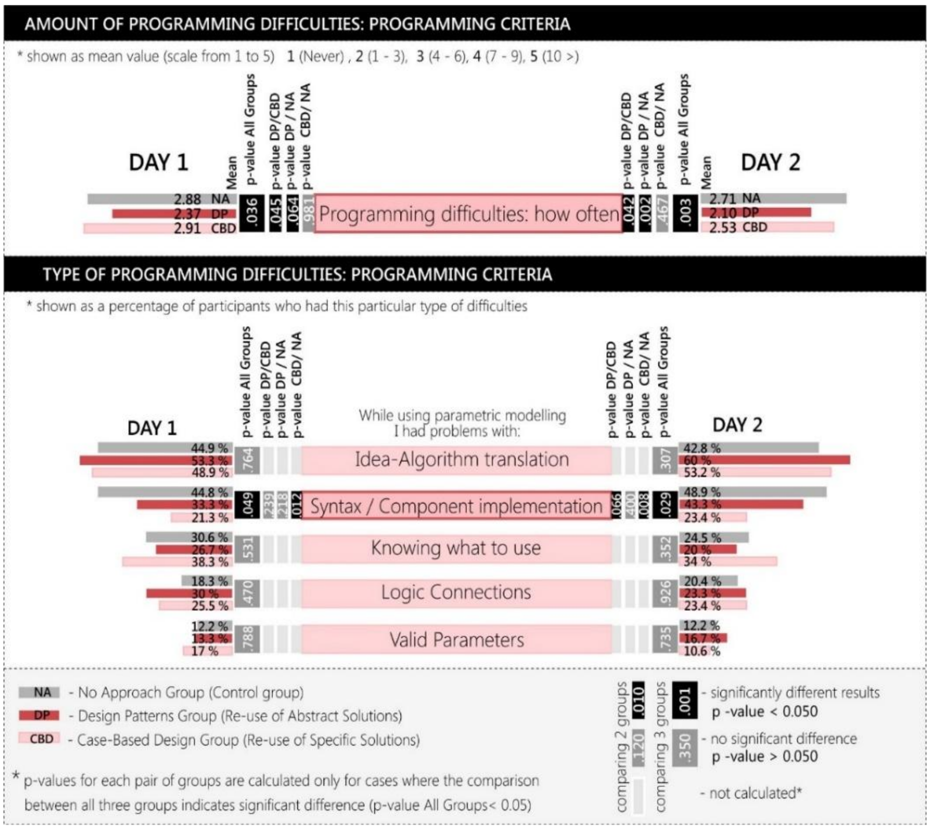

Fig. 1. Overall amount of difficulties. Typology and distribution of programming difficulties.

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 (Design Patterns) 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 (CBD) approach did help to overcome certain types of difficulties (Fig.1).

4 Effect of the Approaches on the Design Thinkingand Performance

4.1 Shift in Design Thinking

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% certainty level), including the ways designers' think and perform; and in what they ultimately produce. One of the most statistically significant differences is the major shift in the design objectives, caused by the use of approaches.

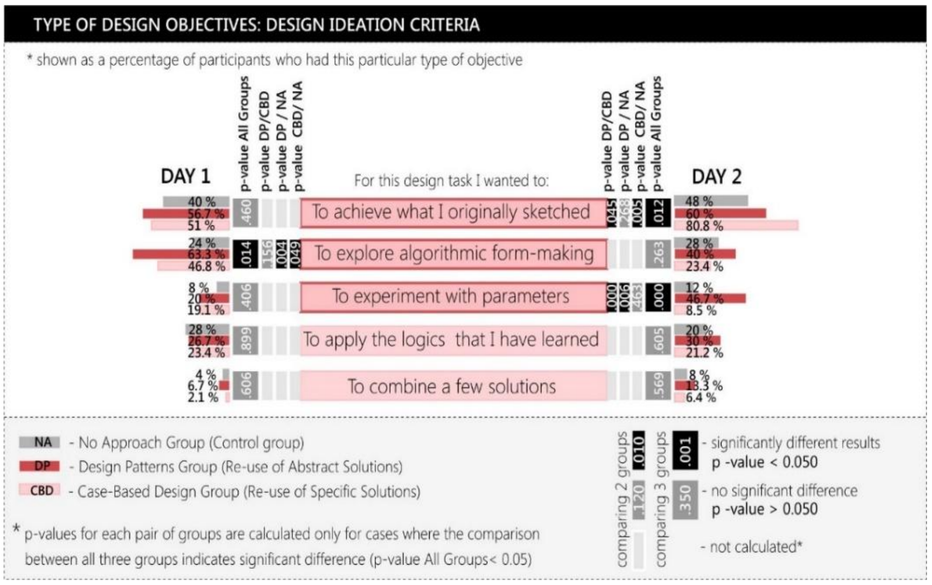

Fig. 2. Typology and distribution of design objectives.

The differences in objectives manifest themselves when designers gain more experience in algorithmic modelling. This can be seen in (Fig.2). It illustrates the measured differences in the design ideation criteria between the test groups on each day of the workshops. For three of these five criteria were the differences statistically significant: for these the p-value is highlighted in black, not grey Interpreting the measured responses, we can see that those designers who reused abstract solutions (the Design Patterns group) were more focused on experimenting with parameters (Fig.2).

Participants of both approach groups were much more likely to explore algorithmic form-making and to try out new programming logics compared to the participants of the control group (no approach (NA)). It should be noted that the group using the Case-Based Design approach was also more invested in the investigation of the capacity of algorithmic modelling (46.8%) compared to the control group (24.4%), however the DP group showed the biggest interest 'To explore algorithmic form-making' (63.3%) (Fig.2). Those who reused algorithmic solutions from specific design cases (Case-Based Design group) were more committed to realise the originally sketched design ideas and were less interested in explorations and experimentations (Fig.2).

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 (Fig.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 (Fig.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.

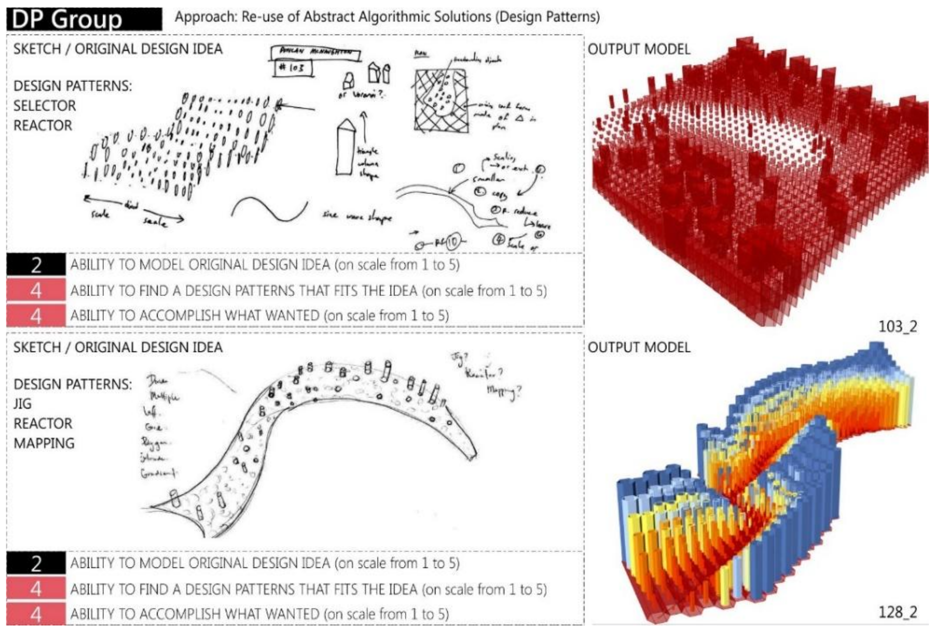
DP Group
Approach: Re-use of Abstract Algorithmic Solutions (Design Patterns)
SKETCH / ORIGINAL DESIGN IDEA
DESIGN PATTERNS:
SELECTOR
REACTOR
OUTPUT MODEL
2 ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)
4 ABILITY TO FIND A DESIGN PATTERNS THAT FITS THE IDEA (on scale from 1 to 5)
4 ABILITY TO ACCOMPLISH WHAT WANTED (on scale from 1 to 5)
103_2
SKETCH / ORIGINAL DESIGN IDEA
DESIGN PATTERNS:
JIG
REACTOR
MAPPING
OUTPUT MODEL
2 ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)
4 ABILITY TO FIND A DESIGN PATTERNS THAT FITS THE IDEA (on scale from 1 to 5)
4 ABILITY TO ACCOMPLISH WHAT WANTED (on scale from 1 to 5)
128_2

Fig. 3. Design Patterns group. Typical cases where designers have significantly changed their original idea and still reported that they were able to accomplish what they wanted.

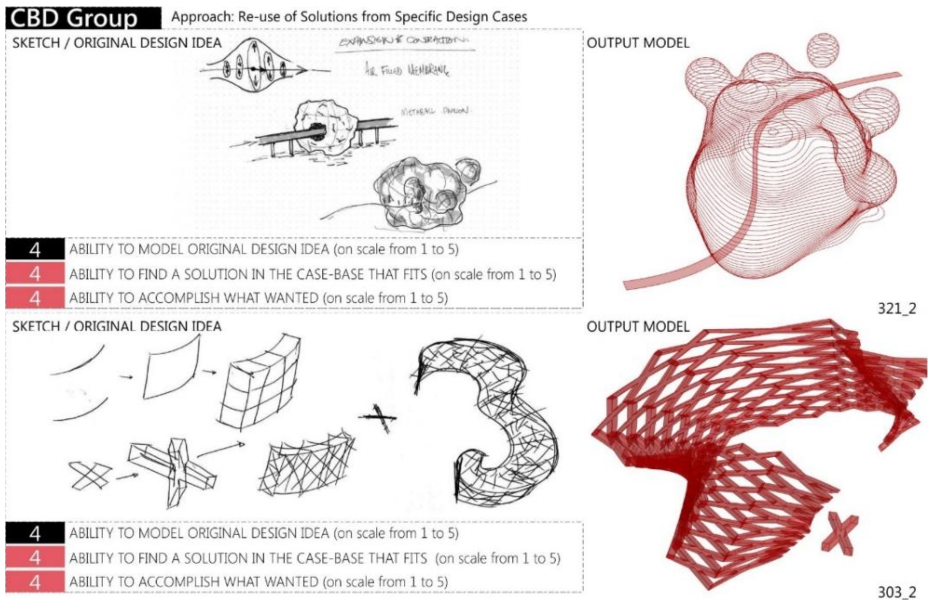
CBD Group
Approach: Re-use of Solutions from Specific Design Cases
SKETCH / ORIGINAL DESIGN IDEA
OUTPUT MODEL
4 ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)
4 ABILITY TO FIND A SOLUTION IN THE CASE-BASE THAT FITS (on scale from 1 to 5)
4 ABILITY TO ACCOMPLISH WHAT WANTED (on scale from 1 to 5)
321_2
SKETCH / ORIGINAL DESIGN IDEA
OUTPUT MODEL
4 ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)
4 ABILITY TO FIND A SOLUTION IN THE CASE-BASE THAT FITS (on scale from 1 to 5)
4 ABILITY TO ACCOMPLISH WHAT WANTED (on scale from 1 to 5)
303_2

Fig. 4. Case-Based Design group. Typical cases where designers developeda model that was close to their original idea and reported that they were able to accomplish what they wanted.

Statistical testing indicates that designers who used case-based reasoning while developing their parametric solutions tended to focus on modelling a particular design outcome. 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.

4.2 Change in Model Complexity

The evidence suggests that use of case-based reasoning in parametric design will most likely decrease the variety of programming components used to create algorithmic models. Designers who use CBD also tended to produce less novel (more typical) 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

The shift in design strategies caused by the use of abstract and case-based algorithmic 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 (Fig. 5. 6.). 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. Four example designs from this abstract group are shown in (Fig. 5. 6.). The score highlighted in black under each design has been developed as a means of systematically ranking the complexity of the programming algorithm. 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 (a score of 2 or 3 out of 5).

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.

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 0.463). Those DP designers, who managed to develop more complex programming algorithms also found the DP approach more helpful (correlation coefficient 0.417). Model and programming algorithm complexity are seen by these designers in a positive light.

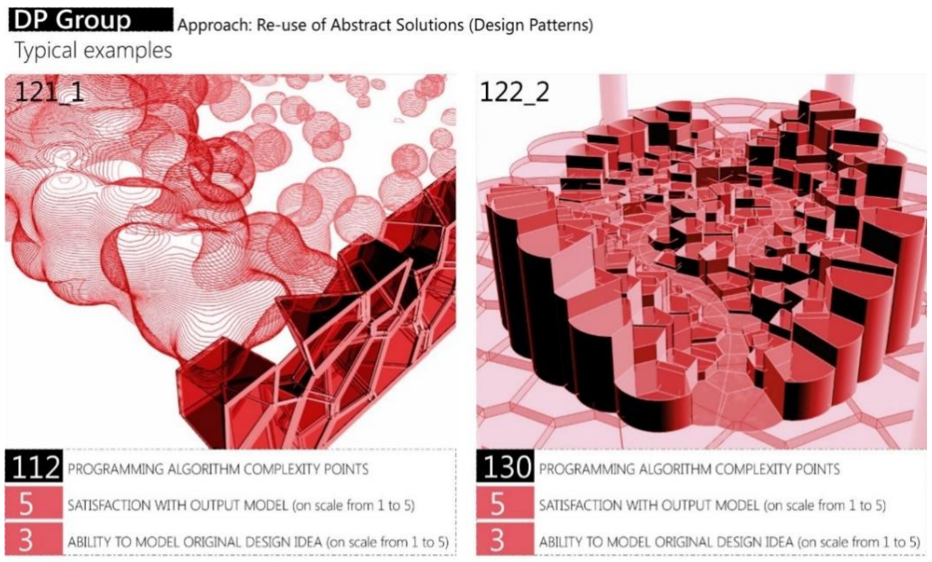
DP Group
Approach: Re-use of Abstract Solutions (Design Patterns)
Typical examples
121_1
112
PROGRAMMING ALGORITHM COMPLEXITY POINTS
5
SATISFACTION WITH OUTPUT MODEL (on scale from 1 to 5)
3
ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)
122_2
130
PROGRAMMING ALGORITHM COMPLEXITY POINTS
5
SATISFACTION WITH OUTPUT MODEL (on scale from 1 to 5)
3
ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)

Fig. 5. Examples of models, designed by the DP group participants, who were able to accomplish what they wanted; significantly changed the original idea; and developed more complex programming algorithms and output models.

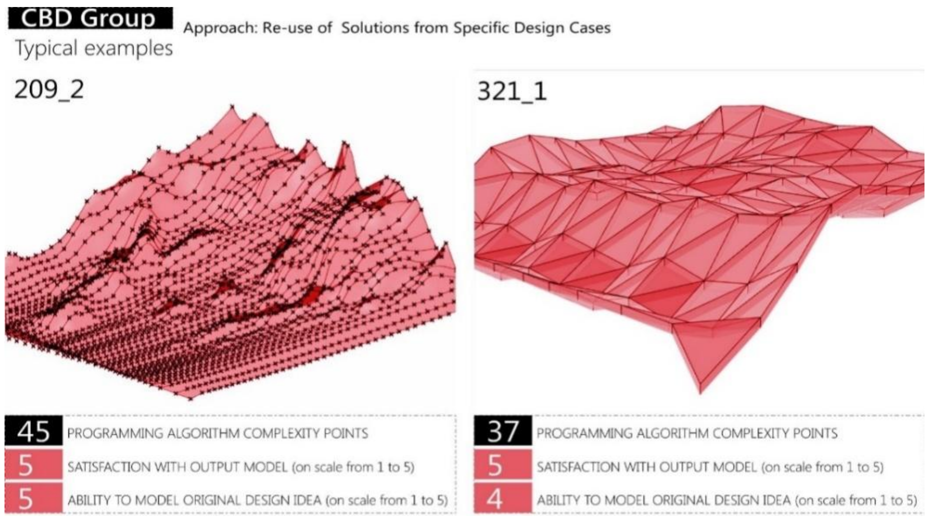
CBD Group
Typical examples
Approach: Re-use of Solutions from Specific Design Cases
209_2
321_1
45 PROGRAMMING ALGORITHM COMPLEXITY POINTS
5 SATISFACTION WITH OUTPUT MODEL (on scale from 1 to 5)
5 ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)
37 PROGRAMMING ALGORITHM COMPLEXITY POINTS
5 SATISFACTION WITH OUTPUT MODEL (on scale from 1 to 5)
4 ABILITY TO MODEL ORIGINAL DESIGN IDEA (on scale from 1 to 5)

Fig. 6. Examples of models, designed by the CBD group participants, who were able to accomplish what they wanted; managed to model the original idea; but developed more simple programming algorithms and output models.

In contrast to the abstract DP group, designers who reused algorithmic 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 score’ has a negative correlation (correlation coefficients -0.362 / -0.378) with ‘satisfaction with the design outcome’. When the CBD group participants managed to come up with more simple programming solutions, they were apparently more satisfied with the outcome. In summary, those who reused specific solutions saw complexity in a negative light, which is the exact opposite of what the group who reused abstract solutions tended to think.

4.3 Key Findings

The evidence presented in this paper demonstrates thatthe integration of knowledge reuse approaches, with learning and design processes, is beneficial. Both extremes of the knowledge reuse approach reduced barriers to using programming in design and improved design performance. Design Patterns developed by Robert Woodbury (an example of the abstraction reuse) proved to be an effective design support and learning method, significantly reducing learning barriers associated with the use of algorithmic modelling systems and programming languages. The use of abstract solutions (patterns) helps architects to understand and adopt algorithmic design methods better. Even though most of the participating designers and architects found the use of patterns to be less intuitive and less easy-to-use compared with the reuse case-based algorithmic solution, overall the pattern approach proved to be a more effective design support method, particularly at the initial stages of learning.

The use of the Case-Based Design approach (reusing specific algorithmic solutions) helps to reduce problems associated with use barriers (the implementation programming components and syntax), which often occur when designers know ‘what to use’, but do not know ‘how to use it’. However, the reuse of case-based solutions does not reduce the overall number of problems, and seems to discourage design exploration. It encourages more focused reasoning, oriented towards the realisation of the original design intention.

5.0 Some Implications for Conceiving the Parametric Design of Cities

While the methodology and principles of ‘abstract’ (DP) and ‘case-based’ (CBD) solutions adaptation differ, both approaches seek to make re-use of algorithmic design knowledge more effective. Reflecting on this research in the context of the theme of CAAD Futures 2015, we consider some implications for the next city. The use of abstract patterns for this study was directly informed by Robert Woodbury’s ‘Elements of Parametric Design’, which cites the seminal work on design patterns by Christopher Alexander. After a flurry of activity in the 70’s and 80’s Alexander’s approach to the reuse and combination of abstract design patterns has reappeared in recent CAAD discourse: researchers at TU Lisbon have directly referenced the principles of pattern language to develop a GIS based urban design tool; FelizOzel has examined the relevance of pattern language for Building Information Models [15]; while a position paper by Andersen and Salomon traces a genealogy of pattern thinking from Alexander to Gregory Bateson, to propose alternate ways to use patterns for design [16].

The design of cities is one of the most complex systems that architects and urban designers face, hence perhaps the interest in the use of a pattern approach such as those developed by Christopher Alexander. Seldom, however, are complete cities designed at one time by one group of designers. Rather the design of cities typically occurs over time, often with overlapping and contradictory master plans and transport systems. Moreover, in many cases typography provides a complexity in the third dimension that works against urban grids and other street patterns conceived in plan. These factors and the failure of some of the modernist urban patterns, such as the point blocks of Corbusier et al, have meant the use of patterns in architecture and urban design have a mixed reception for contemporary designers. How might a hybrid approach, which combines the processing capacity of the computer to utilize patterns with a more intuitive, responsive ‘case based’ approach, be conceived? Ouzel’s paper on the relevance of pattern language for BIM and the paper by Anderson and Salomon on pattern thinking will be discussed for the contrasting insights they shed on the potential of a hybrid ‘abstract – contextual’ approach to the algorithmic design of cities.

The focus of Ozel’s discussion of Alexanders’s pattern language is to identify the difference between the adaptation of the idea in software engineering *‘to model real world objects in a value free, impartial way’*. In contrast, at the object classification and structural levels Alexander embedded his experiences as a designer within the patterns. Relating this back to the above study, following the precedent of software engineering Woodbury considers parametric design patterns as neutral chunks of code. By comparison, in the case based approaches to developing parametric models, the intent of the designer who generated the case is embedded to the point where any major restructuring of relationships is constrained. In the case of urban design, as briefly discussed above, the level of complexity and contextual variation precludes the table rasa implied by abstract patterns. Nor would the superimposition of case based precedent be useful, given the difficulty of major restructuring (as would be required by the new urban context).We could predict that nether approach would be easily applicable for the parametric design in an urban context. Ideally, a parametric design approach for urbanism would enable exemplar urban typologies to be accessed and adapted within an open set of abstractpatterns. A hybrid approach,which would enable a designer to start with abstract patterns and embed these with design intent, through the integration of the existing context and as appropriate, the overlaying of urban exemplars. In this conception of the application of abstract and case based computation, the new city would be a subtle mix of old and new typologies tuned to specific contexts.

Anderson and Saloman take a wider perspective on pattern thinking,which provides an alternate take on how abstract and case based parametric design may be useful in conceiving the new city. Their critique of Alexander’s pattern language is that this privileged homeostasis, intended to maintain continuity and identity. They argue that this is at odds with the dynamism of natural and human systems (such as contemporary cities). They contrast Alexander’s pattern language with the dynamic conception of patterns by the chemist Ilya Prigogine. His research focused on forms of behavior that appeared chaotic but moved in and out of recognizable patterns. *‘From Prigogine we get an interpretation of the pattern as a condition of instability embedded in an entropic and unpredictable environment, rather than juxtaposed to it.’*

Computationally this conception of pattern would map to a self-organizing system, such as that achieved by agent and flocking algorithms. Such algorithms underpin software used to model human movement through urban systems (space syntax) and traffic modelling applications. Agent approaches have also been used to generate urban form, although these appear to have minimal practical applicability to contemporary cities. Returning to a discussion of a hybrid approach to a parametric city design system, the precedent of Prigogine suggests the integration of form and patterns of use *over* time. One way to conceive this would be the identification of fixed and dynamic aspects of a model. Just as architecture is increasing responsive to changes in environment and usage patterns through dynamic ventilation, sunshades and access control, the future city is likely to have a level of automation. In conceiving the components of an urban parametric system, a distinction could be made between fixed and dynamic components. The dynamism would be at a range of temporal scales: real time (such as traffic control systems): components that respond in relation to daily cycles of light, weather and occupation (e.g.automated lighting, sun screens, access control); and seasonal cycles where for example, an open piazza becomes a winter garden.

This final section has opened up the study of abstraction and case based approaches in relation to the complexity of designing urban systems. The study provided evidence of how young designers respond to the introduction of abstraction and case based parametric design. The context or urban design reveals neither approaches would be directly applicable to such a complex design context. While parametric design offers promise, there is a much more research – both software development and studies of designers engaging with complex urban contexts - to be undertaken. Our next stage of the project will be to explore such a hybrid approach to parametric design as that outlined above and undertake similar rigorous evaluation of the uptake and impact on design outcomes in a more complex design context.

References

1. Menges, A., &Ahlquist, S. (Eds.). (2011). Computational Design Thinking: Computation Design Thinking. John Wiley & Sons.
2. Celani, G., & Vaz, C. E. V. (2012). Cad scripting and visual programming languages for implementing computational design concepts: A comparison from a pedagogical point of view. International Journal of Architectural Computing, 10(1), 121-138.
3. Matcha, H. (2007). Parametric Possibilities: Designing with Parametric Modelling. In Predicting the Future: 25th eCAADe Conference Proceedings (pp. 849-856).
4. Woodbury, R. (2010). Elements of parametric design.
5. Krueger, C. W. (1992). Software reuse. ACM Computing Surveys (CSUR), 24(2), 131-183.
6. Terzidis, K. (2006). Algorithmic architecture. Routledge.
7. Mann, D. Someone, Somewhere Really Did Already Invent The Wheel You're About To Re-Invent., HKIVM 7th International Conference, 2005.
8. Gamma, E., Helm, R., Johnson, R., &Vlissides, J. (1994). Design patterns: elements of reusable object-oriented software. Pearson Education.
9. Designpatterns. (2014). Available from: Open Source Repository <http://www.designpatterns.ca> (accessed 1 October 2014)
10. Maher, M. L., & de Silva Garza, A. G. (1997). Case-based reasoning in design. IEEE Intelligent Systems, 12(2), 34-41.
11. Kolodner, J. L. (1991). Improving human decision making through case-based decision aiding. AI magazine, 12(2), 52.
12. Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI communications, 7(1), 39-59.
13. Riesbeck, C. K., &Schank, R. C. (2013). Inside case-based reasoning. Psychology Press.
14. Ko, A. J., Myers, B. A., & Aung, H. H. (2004, September). Six learning barriers in end-user programming systems. In Visual Languages and Human Centric Computing, 2004 IEEE Symposium on (pp. 199-206). IEEE.
15. Ozel, F. (2007) Pattern Language and Embedded Knowledge in Building Information Modeling, 25theCAADe conference proceedings, pp. 457-464.Anderson P. and Saloman D. (2010) Proceeding of ACADIA 2010, pp. 125-132.
16. Anderson, P. and Saloman D. (2010) The Pattern that Connects, ACADIA 2010 pp. 125-132