POSITION STATEMENT
Can Software Design Benefit from Creative Computing?

Hong Zhu
Department of Computing and Communication Technologies
Oxford Brookes University, Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

I. Introduction

Design is one of the most elusive tasks in any engineering or creative activity. Design problems have been recognized as wicked [1] or ill-structured [2]. Software design is even more difficult because software is the most complicated man-made artifact and system. One of the key characteristics of design is its creativeness. The question is, therefore, can design benefit from creative computing? In other words, can software design be a successful subject of the research on creative computing?

For a long time, people have been search for design methodologies so that design can be mastered through education and learning, even become mechanical and automated. Researchers in the subject area of Computer Aided Design (CAD) have developed software tools to support various design activities in design process but the creativity, which still relies on human beings.

The research on software design in the past decades has made some breakthroughs that may lead to a revolutionary new type of software design tools so that software developers can benefit from creative computing. These research results combined with the general theory of design methodologies may make creativity in software design benefit from automation. The following briefly examine such theories and research results in the light of creative computing.

II. Design Space

A design space for a particular subject area is a space in which design decisions can be made. Each concrete design of an object in the subject domain is then a point in this space. Therefore, design can be regarded as navigation in this space. Understanding the design space of a particular domain plays a significant role in design. In a recent article [3], Shaw demonstrated the differences between designs without aware of the design space and designs with explicitly and systematically exploring a design space. Here, we assert that design space is also the key to the quest of design as a subject of creative computing.

A design space can be represented in three forms:

- *Multi-dimensional discrete Cartesian space*, where each dimension represents a design decision and its values are the choices of the decision. However, in practice, most interesting design spaces are too rich to represent directly in such a way. Design dimensions are not independent, so choosing an alternative for one decision might preclude alternatives for other decisions or make them irrelevant, as Shaw pointed out [3].
- *Instance list*, where a number of representative instances of the domain are listed with their design decisions.
- *Tree structure*: where nodes represent a design decision and alternative values of the decision are the branches, which could also be dependent design sub-decisions [4].

Since 1970s, computer science has used design spaces to organize software design knowledge to describe software architectural components and connectors as well as software architectural styles.

III. General Design Theory

The General Design Theory was proposed by Yoshikawa in 1980 [5], [6]. It generalizes the notion of design space in order to achieve design automation. Yoshikawa further divides a design space into two views: one for the observable/structural features of the designs, and the other for functional properties of the designs. These two views are linked together by the instances of the domain, which show which combination of properties in structural view is associated to the combination of properties in the functional view. By doing so, Yoshikawa demonstrated that two types of design problems could be solved automatically:

- *Synthesis problem*: when a set of functional features is given as requirements, the synthesis problem is to find a set of the structural features as a solution that meets the requirements.
- *Analysis problem*: when an objects structural properties are given, the analysis problem is to find out its functional properties.

In computer science, researchers have developed knowledge on how software architectural styles are related to software quality so that the structural design decisions can be made to achieve required quality attributes. These can be understood in the framework of General Design Theory [7]. Here, the architectural styles are a kind of abstract representation of design instances and the two views of design spaces are: the structural features are embodied in architectural style, while the functional properties are actually quality attributes. Such knowledge can be easily transformed into a design space and to apply the General Design Theory to achieve computational creativity.

However, many other aspects of software design are still in lack of systematic understanding of the design space.

IV. Design Patterns

Design pattern is again a notion borrowed from the research on general design methodologies, which stems from architectural design.

A design pattern contains encoded knowledge of design solutions to recurring design problems. Since 1990, much work has been reported in the literature on object-oriented design. Its success in improving OO code design has fostered research of design patterns in interface design, software architecture, security, requirements analysis, software process, fault tolerant design, etc. However, most of the design patterns are represented informally in the so-called Alexander format and thus suffering from ambiguity in interpretation and difficulty to apply.

In the past decade, much research effort has been reported in the literature on formalization of OO design patterns. More recently, we have proposed a set of operators on design patterns for pattern composition [8] and a complete set of algebraic laws that these operators obey [9]. In these operators, pattern oriented design decisions can be represented formally and the result of design decisions can be worked out automatically. Moreover, with the algebraic laws, the equivalence between different pattern expressions can be proven formally and automatically through a normalization process. This sheds a new light to the automation of pattern-oriented software design.

However, there is still a huge gulf to achieve software design as a successful subject of creative computing. First, existing formalization of design patterns has focused on the solutions of patterns. How to formalize the intensions of design patterns is still an open problem. Only if intensions of design patterns can be formalized and automatically recognized from the design problem, automatic selection of design patterns can be automated.

Second, the formalization of design patterns has been limited to object-oriented code design. How to formalize patterns of other design aspects remains open.

V. Directions of Research

We believe that formalization of design knowledge and representing it into an encoded format such as design space are necessary for software design as a subject of creative computing. Two approaches have been proposed and investigated, but mostly separately: the design space approach and the design pattern approach. Whats important for future research are

1) Formalisation of the intension of design patterns so that the knowledge of design patterns can fit into the framework of General Design Theory, thus enable both synthesis and analysis design problems be solved computationally.
2) Generalisation of the formal theory of OO design to the patterns of other software design aspects so that a wide range of software design issues and aspects can be computed.
3) Representation of software architectural styles in the framework of General Design Theory.

An approach to achieve these goals is to regard design patterns as hot spots in design spaces, so that design knowledge in the framework of the general design theory can be unified with the formal algebra of design patterns. A general purpose design space definition language and its computational implementation will provide a bridge to the future.

References

[1] H. J. Rittel and M. M. Webber, "Planning problems are wicked problems," in *Developments in Design Methodology*, N. Cross, Ed. Wiley, 1984, pp. 135–144.

[2] H. A. Simon, "The structure of ill-structured problems," *Artificial Intelligence*, vol. 4, pp. 181–200, 1973.

[3] M. Shaw, "The role of design spaces," *IEEE Software*, vol. 29, no. 1, pp. 46–50, Jan. 2012.

[4] F. J. Brooks, *The Design of Design: Essays from a Computer Scientist*. Addison-Wesley, 2010.

[5] H. Yoshikawa, "General design theory and a CAD system," in *Man-Machine Communication in CAD/CAM, Proceedings of the 1980 IFIP WG5.2-5.3 Working Conference, Tokyo, Japan*, T. Sata and E. Warman, Eds. North-Holland, 1981, pp. 35–57.

[6] Y. Kakuda and M. Kikuchi, "Abstract design theory," *Annals of the Japan Association for Philosophy of Science*, 2001.

[7] H. Zhu, *Software Design Methodology - From Principles to Architectural Styles*. Elsevier, 2005.

[8] I. Bayley and H. Zhu, "A formal language for the expression of pattern compositions," *International Journal on Advances in Software*, vol. 4, no. 3&4, pp. 354–366., 2011.

[9] H. Zhu and I. Bayley, "An algebra of design patterns," *ACM Transactions on Software Engineering and Methdology*, 2012, (In press).