5
Classes of Design-Classes of Tools

Gerhard Schmitt
Architekturabteilung
ETH Zurich

It is unrealistic to expect one computer-aided design tool to sufficiently support any one given design process. Yet it is necessary to define new CAD programs that give semantic support in design. To this end, the paper first differentiates phases and classes of design and then attempts to establish relations between the defined classes and appropriate computer-aided design tools. In three main sections it describes (i) routine, innovative and creative design, (ii) a set of corresponding prototype design tools, and (iii) two examples of routine and innovative design which use these tools. The purpose of the paper is to make a contribution to the definition of domain specific aspects of CAD and to propose a mapping between processes and tools.

Introduction
Architects employing computers in teaching design have encountered a number of severe conceptual problems in applying the new tool to established design methods and theories. The reasons are two-fold: obvious weaknesses of current CAD systems in representing qualitative and functional aspects of design and the lack of appropriate teaching and design methods which will be fundamentally different with the new tools. The object representation and manipulation capabilities of state-of-the-art CAD systems are far more advanced than functional modeling capabilities or the necessary accompanying design methodology [Schmitt 88].

There seem to exist two alternatives to solve this problem. The first would be to adjust design teaching to the capabilities of hardware and software thus, creating a constant dependency on commercial developments. The second
alternative would be to attempt a classification of the design process itself and to search and define a set of conceptual tools each of which support a particular class of design. This approach will guarantee some independence from commercial developments and has the potential to create a general body of design knowledge which is comparable to summaries of methods and experiences in other disciplines, such as medicine.

Most often the design process is broken down into several stages [Laseau 80], reaching from program development to construction. Beyond this however, there are obviously different types of design which Gero has appropriately coined as routine, innovative and creative [Gero 88]. A class of design, then, is defined as the intersection of design types and design stages that share common characteristics, such as routine program development. A class of tools, accordingly, is a set of programs that share common characteristics, such as 3D solids modelers. Upon closer examination of existing and planned tools for design, a certain mapping between classes of design and classes of tools begins to emerge. This paper claims that general CAD tools are best suited for the general production of drawings and that the few emerging parametric programs are well suited for routine design. There seems to be very few support tools for creative and innovative design which in part explains the absence of any new architecture which involves the use of computers in design process.

**Classes of Design**

The architectural design process includes several stages, as proposed by Laseau: program development, schematic design, preliminary design, design development, contract documents, shop drawings, and construction [Laseau 80]. While this differentiation is useful for understanding and possibly formalizing stages of design, for the development of a new design methodology a classification of different design types is necessary.

It seems appropriate to differentiate at least three types of design, as proposed by Gero, Maher, and Zhang: routine design, innovative design, and creative design [Gero 88]. While this is a useful classification for research purposes, it does not quite reflect architectural reality and must therefore, be seen as a means of abstraction. From the intersection of types and stages we derive design classes. For lack of a better alternative, design classes are represented using prototypes which at the moment do not well express important factors necessary for successful real-world design such as the socio-economic environment, existing context, and client profiles.

The challenge is to describe the roots or qualitative and functional aspects of the prototype and not the external representations. The roots are almost always functional whereas the external representation is a well described object. We are experimenting presently with a different
representation of design, treating existing architecture as cases and new design as a combination of case-based reasoning and rule-based inferencing, but an in-depth description of this approach would be beyond the scope of this paper.

**Parametric Design**
Parametric design, also often called routine design [Gero 88], is the major focus for commercial programs in defining and supporting the design process. Routine design assumes that:

- The design problem is well defined and client requirements are well understood.
- There exists a parameterized prototype for the type of design that is to be developed and a data base of parameter variations.
- The final design may be derived by refining, but not fundamentally changing the prototype.

This type of design is characterized by "prototype refinement", a goal-directed activity. Beginning with a given prototype of, for example, a piece of furniture or equipment, the designer adjusts a number of parameters to the specifications of the design program. The parameters, typically geometric properties or materials, are normally well understood and are manipulated either in the designer's memory or with advanced modeling systems. The functional requirements of the design are known and the semantics or the teleology (the purpose of each element) of the design are not changed but accepted from previous examples. This type of design and design process rely heavily on instantiation of designs from a catalogue of parameterized examples [Kramer 88].

**Innovative Design**
Innovative design, also referred to as prototype adaptation [Gero 88] and prototype combination [Faltings 88] is used when the refinement of a known prototype will most likely not lead to a satisfactory solution. Prototype adaptation and prototype combination can potentially lead to creative design. The designer has a general idea of the desired object and the design process is still a goal-directed activity. It cannot be completed with routine design because either the functional description or the object properties are not achievable utilizing a given prototype. Therefore, combination of two or more prototypes which each have some of the desired properties is necessary. In some cases, innovative design is achievable with prototype modification. An example would be the development of an advanced type of intelligent office building for which some information infrastructure needs, such as new requirements for communication are still unknown. Case-based reasoning (the
entire building being the case) and explanation-based learning systems (functions or building parts being the examples) are of particular interest in innovative design. Once a prototype is adapted or several prototypes have been combined, the design process becomes routine design.

**Creative Design**

Creative design is rare and defined by the development of new solutions that may only be partially defined at the outset. Both functional requirements and the objects properties are not necessarily and completely known and the final design may in fact influence the original problem definition or even render it partially irrelevant. It is possible that a unique solution (Ronchamps, Sydney Opera House) may be found to a problem in which case the result would be an 'archetype' In most cases, "prototype creation is necessary, which later can be combined and modified (innovative design) and instantiated (routine design). An example is the invention of a new machine, such as the personal computer. Although creative design must be mentioned as the perhaps most important design class, the attempt to formalize creative design is outside the scope of this paper.

**Gasses of Tools**

Different tools support different types of design. As a general rule, object presentation and manipulation tools such as word processing, drafting and three-dimensional modeling programs-in the following referred to as generic tools-have a larger application base and a more robust and commonly understandable user interface than experimental programs for parametric modeling or grammar generators. The following classes of tools are to be considered:

- **Generic tools.** Those include word processing (e.g., WORD, Emacs), spreadsheets (i.e., EXCEL), painting (e.g., MacPaint), drafting (i.e., AutoCAD), three-dimensional modeling (e.g., Personal Architect), hypermedia (e.g., Hypercard). Generic tools are particularly strong in representing objects, rather than qualitative or functional properties and requirements.
- **Parametric tools.** Those include two-dimensional parametric design tools (Synthesis) and teaching programs (Topdown) [Mitchell 89] which make use of formalized design knowledge about machine parts, classical orders, and trees, as respective application examples.
- **Prototype editors.** Those include tools to interactively establish, manipulate, and combine prototypes through interactive input or forms of machine learning.
The GPC and TARTAN prototypes described below are examples with particular emphasis on supporting conceptual design development.

- Grammar editors. Those include tools to interactively define and edit shape, structure, or later possibly knowledge grammar generators, to manipulate their control structure, and to add domain-specific knowledge. The FROEBEL and GRC programs described below are examples for programs intended to support experimental design development.

There is no simple mapping between generic tools and routine design, for example, or between complex tools and creative design. In some cases, the opposite holds true. Graphic, symbolic, and alpha-numeric information are not readily made compatible at the moment. The following table attempts an overview of a mapping between phases of the design process, types of design, and design tools.

<table>
<thead>
<tr>
<th></th>
<th>Routine</th>
<th>Innovative</th>
<th>Creative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program development</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>Schematic design</td>
<td>P</td>
<td>P E S</td>
<td>S</td>
</tr>
<tr>
<td>Preliminary design</td>
<td>PE</td>
<td>P E S</td>
<td>P E S</td>
</tr>
<tr>
<td>Design development</td>
<td>PE</td>
<td>P E S</td>
<td>P E S</td>
</tr>
<tr>
<td>Contract documents</td>
<td>G</td>
<td>G</td>
<td>G E P</td>
</tr>
<tr>
<td>Shop drawings</td>
<td>G</td>
<td>G</td>
<td>G E P</td>
</tr>
<tr>
<td>Construction</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
</tbody>
</table>

C  Generic tools
P  Parametric modeling tools
E  Prototype editors
S  Shape grammar generators

The following two sections present examples from two tool classes: prototype editors and grammar editors. Seen in isolation, the programs described are rather simplistic. They become interesting if used in combination and through their strong visual feedback and real time evaluation capabilities.

**Prototype Editors**

Prototype editors primarily support tasks that relate to the category of routine design. The prototype to be manipulated is either built into the program as it is in the case of ARCHPLAN [Schmitt 88], or it is defined by the designer, a possibility offered by the Edisyn program [Maher 87]. The manipulation of
parameters of the prototype can be performed manually, in which case the visual feedback and some information from attached attributes determines the value of each new object variation, or it may occur automatically through local optimization of parameters and the necessary local and global conflict resolution. The following examples represent partial approaches for prototype definition and manipulation.

**GPC: a graphical prototype constructor**

As a simplification, a prototype for standard or routine design tasks may be defined as an object of non-trivial complexity in the sense of object-oriented programming. As such, GPC allows the graphical construction of objects which in isolation or as combinations of instances describe the prototype for design. To build the prototype of a conference room, for example, GPC needs different types of knowledge, all of which may be input graphically:

- **Geometric knowledge.** This includes the position of the object in three-dimensional space, and its dimensions.
- **Hierarchical knowledge.** It includes the knowledge of belonging to a named assembly or a part-of relation.
- **Relational knowledge.** It includes relations between parts of the design that are established (and, if desired, enforced) graphically by pointing. Examples are on-top-of, north-of, south of, etc.
- **Functional knowledge.** This includes the definition of the main function of the object, which could describe static (fixed location for a certain part or activity), as well as dynamic (move along a certain path) functions.
- **Constraint knowledge.** Constraints can be defined graphically or numerically and include, for example, less_than, greater_than, or geometric constraints.

The program could be seen as a graphical shell that allows the construction of the basic units of a prototype and their combination. Figure 1 shows examples of the user interface. GPC has at the moment very limited inferencing techniques. It is possible, however, to produce OPS5 rules with the program so that more complex inferencing may be executed externally in OPS5. GPC is programmed in C, using frames and the graPHIGS [IBM 86] package, on the IBM RT 5080 graphics workstation operating under AIX.

**Tartan: a three-dimensional grid editor**

A tartan grid is a representation of three-dimensional objects which uniquely defines each corner, edge, intersection, plane, and plane orientation or direction of objects in the grid. The TARTAN grid editor allows a designer to
- define 11w tartan grid in three dimensions.
- insert into and delete three-dimensional building blocks from the tartan grid.
- inquire about attributes of building blocks.
- write to and read from an input file.
- display and manipulate the building in real time in sections, elevations and perspectives.

The designer manipulates the primary building elements and the grid with a set of function keys and direct keyboard input. A digitizer supports the selection of objects; analog dials allow real time view and scaling operations. Examples from a typical session are shown in figure 2. The design begins with the presentation of a default three-dimensional grid which is the average of grids from previous sessions. The grid itself may be manipulated by sets of integers which represent the distance between horizontal and vertical grid lines. Subsequently, building blocks are filled in by specifying areas and heights with digitizer and numerical input. Although the described sequence corresponds to an additive method, a subtractive approach is available as well in which individual or groups of building blocks are selected, moved, or deleted. The ultimate effect is one of carving or direct modeling of a large block.

The tartan grid editor is most appropriate for the development of massing models with an underlying three-dimensional grid structure. Object manipulation feedback is instantaneous and encourages direct visual judgement. Alternatives that appear satisfactory are written to a data base and can be retrieved for comparison. A disadvantage is the restriction of the grid. The program employs

Figure 1 User interface of GPC, the Graphical Prototype Constructor. Object definition and frame assembly.
direct object manipulation, that is, by modifying the screen object, the data structure is manipulated as well.

**Shape Grammar Editors**
Shape grammars area subclass of production systems and consist of graphical rules with a left hand side (the condition) and the graphical results of a transformation as the right hand side (the action). Shape grammars in design have been popularized mainly by the work of Stiny, Mitchell and Flemming [Flemming 86]. Chinese gardens, Gothic church windows, and Victorian residences name a few of numerous successful implementations. In most cases, shape rules are defined manually or in source code and cannot be easily edited. The following two examples provide a real time graphical user interface for the application of pre-defined graphical rules, and an interface for the construction and application of graphical rules. In both cases the construction of graphical "sentences" (3D objects) consisting of simple "words" (shapes) is possible. Prototypes are either the rules themselves or the "sentences" formed with the macro language.

**Froebel: a graphical production system**
FROEBEL generates three-dimensional objects from a set of twelve three-dimensional shape grammar rules. The production system was inspired by Stiny's work on Froebel's Kindergarten grammars [Stiny 80]. A three-dimensional object can be defined either by applying shape rules step by step or input of a simple "sentence", using the program's macro parser. The "sentences", constructed by a

![Figure 2 TARTAN, the three-dimensional shape editor. Two interactive screens](image-url)
combination of a string of rule names, recursion levels, and geometric constraints, maybe seen as the actual prototypes. The rules can be executed within the tartan grid described in the previous section.

A typical working session begins with the display divided into six windows as shown in figure 3. Window 1 contains twelve three-dimensional rules, window 2 shows the initial shape. Window 3 displays error messages and current information, window 4 prompts the user and accepts answers. Window 5 lists pre-defined function keys, and window 6 presents the generated results in three dimensions.

The designer may choose an initial shape from window 2 and any rule with the digitizer from window 1. The result of applying the rule to the initial shape appears in window 6. Any rule in any order may be applied to the object in window 6. The dials allow real time transformations such as rotation, translation, and scaling in all axes.

A simple macro language allows the formation of objects by combining individual rules (words) into complex sequences (sentences). Constraints in each orthogonal direction guide and terminate the execution of rules and "sentences."

A typical application is the construction of a stair. The user selects the appropriate rule (one step with its lower left edge on the upper right edge of the previous step), inputs the floor to floor height as a constraint, and requests the system to begin executing this rule. Once the stair reaches the next floor, it will automatically stop. In other words, each rule becomes an object in the sense of object-oriented programming and is therefore, able to handle simple constraints. The program is most useful.

Figure 3  FROEBEL: a graphical production system. Snapshots from the interactive screens.
for the exploration of rule combination and constraint application. Its strengths are the convenient interface and the direct object manipulation capabilities. Disadvantages at the moment are the restrictions of rectangular shapes and the limitations in the "intelligence" of objects.

**GRC: a graphical rule constructor**

The graphical rule constructor is a program for the graphical definition and execution of two-dimensional production rules. It supports the routine and to a degree the innovative design process as it enables the designer to define own rules, produce simple shape "words", and produce graphical "sentences" by combining these words into "sentences" manually or with the built-in macro parser. "Sentences" and combinations of "sentences" may be stored as a new prototype, or existing "sentences" may be edited.

The screen is divided into six windows as shown in figure 4: window I stores and displays user defined rules, window 2 displays the result of rule application, window 3 is the rule definition window, window 5 shows error messages and current information, window 6 contains dialogue, and window 7 presents user options. Keyboard, function keys and dials are used in the same way as in the previous examples.

A typical session begins when the user defines a new shape rule in window 2. The initial shape (left hand side) is specified. Then the first rule (right hand side) may be input graphically. The result appears immediately in window I and is available for application. Following the input of one or more rules, the interface works.

![Figure 4: GRC, the graphical shape grammar rule constructor. Typical interactive screens.](image-url)
exactly as in FROEBEL. At the moment, the program is restricted to two dimensions, as too many possible cases must be considered for three-dimensional input. The program is also compatible with FROEBEL and TARTAN. It is implemented, similar to the previous programs, in C and graPHiCS.

Research and Teaching
The previous sections described examples for applications of domain specific tools to routine and innovative design in theory. They are research programs with limited practical importance yet they begin to explain the notion of domain specific design tools. The following two examples are conceptually not as well defined but relate more closely to practice.

Traditional Taiwanese housing-parametric design
The example of the knowledge-based design of Taiwanese housing is based on the careful study of geometric properties of this building type. The shape-grammar paradigm is used to generate the components of traditional Taiwanese villages such as buildings, courts, rooms, walls, and components. The project is an example of routine design because the parameters of a prototype of the traditional Taiwanese house are known, formalized, and merely manipulated within well defined boundaries.

Figure 5 Axonometric of a five-bay house, showing some foundations, columns, doors, and wall panels
The prototypes consist of AutoLISP functions and association lists which together form the "design object." The program prompts the user at the beginning and during the design development for the value of important parameters which vary within certain limits for different family sizes. The types, however, do not change. The program performs on any IBM AT class machine but the resulting buildings and building blocks, designed and displayed three-dimensionally, are so complex that as a minimum, a SUN3 class machine is recommended. The program may be used to study the effects of parameter variation on buildings and building agglomerations, without going beyond the underlying prototype.

The significance of the program lies in the actual level of interaction between designer and computer: the dialogue takes place on a conceptual level in which family size and proportional aspects determine the shape and size of the building. Whereas the built-in knowledge may be seen as a limitation, it can be changed, and the end result of a session may he edited on the level of AutoCAD entities.

**Maggi House by Mario Campi-Innovative Design**
The Maggi house by the Swiss architect Mario Campi (built in 1980) is an example of the combination of prototypes and may be categorized under the heading of innovative design. The Ticino farm house prototype was combined with a rational and contemporary organizational language prototype. One could argue that either one of the prototypes was modified or adapted to the degree that innovative design resulted. However, the Maggi house is better explained by prototype combination, because the result clearly shows characteristics of both types.

The Maggi house represents a piece of architecture in spatial isolation yet set in a regional context for which the application of a knowledge-based system

![Figure 6](image-url) Front and back elevation of the Maggi house by Mario Campi, with precedent prototype.
for innovative design is feasible. The accompanying program is still under development and is implemented in parallel in AutoLISP, Hypercard/Hypertalk and Smalltalk. In AutoLISP, lists of functions represent prototypes. The Hypercard implementation takes advantage of the graphical and object-oriented-like programming style of Hypertalk. The Smalltalk application makes use of the object-oriented and graphical programming environment.

Conclusions
To divide design processes into different classes is useful for understanding the complexity of design and leads to the isolation and exposure of unsolved research questions. Present commercial CAD programs and tools developed in CAD laboratories best support the routine design process. It would be wrong to assume that any one program could support a given design process. The different phases and types of design require a variety of conceptual and modeling aids. The second large research question, that of principled representation and reasoning techniques is not touched upon in this paper but must be solved for the communication of the individual design tools. Our experiences also show that prototype and shape grammar editors require very high computing power for visualization and execution to be perceived as the dynamic modeling methods that differentiate them from traditional methods.

The time has come to develop appropriate tools to support each type of design and to end the mismatch between application requirements and program offerings: Drafting systems are just not built for conceptual design and paint systems are unfit in the long term for detail development. As in both examples, excessive implicit knowledge is required from the viewer. All resulting interpretations are therefore ambiguous and port human problems to the machine rather than taking advantage of the computers capabilities. Knowledge implicit in details or larger planning units should finally be made available in useful form. The definition of unified standards for product modeling may be a first step in this direction [Gielingh 89].
Acknowledgments
The author would like to thank Heng Tzu Wu, Chen-Cheng Chen, Shen-Guan Shih and Annelies Zeidler. Special thanks to Sibylla Sprecher and Laura Lee. Research funding for Graphical Knowledge Acquisition, of which GPC, FROEBEL, TARTAN, and GRC are a part, is provided by IBM.

References


