A Theoretical Framework for an Intelligent Design Catalogue

Paul Winkelman

*Department of Mechanical Engineering,
University of British Columbia
pwinkel@mech.ubc.ca*

Ian Yellowley

*Department of Mechanical Engineering,
University of British Columbia
yellowley@mech.ubc.ca*

Abstract

This paper outlines continuing work on the intelligent design catalogue. The intelligent design catalogue seeks to create a virtual design environment that is linked to a catalogue of standard parts. The theoretical framework for this research draws on several engineering areas. Within manufacturing, process plans can be developed in a virtual environment independently of the machines on the shop floor just as products can be conceptually designed independently of the standard parts available. The standard parts themselves can be grouped borrowing from classification schemes of Group Technology. Object-Oriented Programming (OOP) provides an environment for the development of the software that runs the intelligent design catalogue. As the objects of OOP parallel standard components, OOP also serves as a design paradigm after which the catalogue can be modelled. . Design theory suggests frameworks for developing a (semi-) hierarchical structure for cataloguing parts and design case studies offer insight into differences between novice and expert designers.

Keywords: standard components, object-oriented, virtual design

1 Introduction

Previous work [1] proposed a framework for the development of an intelligent design catalogue. The intent of the catalogue is threefold. First, to facilitate the retrieval of standard components, along with their specifications, from a traditional engineering catalogue. Second, to integrate the component retrieval feature with a virtual design environment. Third, to locate knowledge normally assumed to reside in the head of the designer and incorporate this knowledge into the virtual system.

The retrieval of standard component data implies that information that is normally presented to the client as a hard-copy catalogue must be transferred to a computer database. Ideally, the component data should be fairly comprehensive including geometry, material, weight, cost and so on. Additional data might include typical uses for the component in question or analytical tools associated with the component to assess its performance once it becomes part of a larger system.

The integration of the design environment with the catalogued data allows the user to send the selected components to the design environment via a list.

Within the design environment, the designer selects the components from the list and links them together. By relaxing the level of component specification required, the environment facilitates the more conceptual stage of the design process. A “component”, for example, may be merely a functional descriptions, or a part type to be refined to an increasingly more specific component as the design progresses. Once components are fully specified, only components with matching interfaces can be linked together. The design environment thus prevents the inadvertent assembly of mismatched parts.

To facilitate the retrieval of a specific component from a potentially large list, the intelligent design catalogue must provide the user with additional knowledge. This is particularly true if the user is not familiar with some of the components. A well structured list of design aids and tools allows the novice designer to develop systems that rival those of the more seasoned designer. This would help remove some of the uncertainty of design, and reduce the likelihood of re-design.

2 Objective

The objective of this research is to produce an intelligent design catalogue which combines a traditional catalogue with a virtual design environment and incorporates additional knowledge to allow novices to design more like experts.

3 Theoretical Framework

An engineering catalogue can potentially have a vast array of components. The greater the number and variety of the components, the more easily designers can create products tailored to their needs. The advantage of an extensive catalogue can be easily lost, however, if the designers have difficulty navigating through the information. The design of the intelligent catalogue itself depends on the development of a theoretical framework that can address the issues involved. Some of the framework has already been discussed in previous work [1]. The framework draws on models from general design theory, manufacturing and object-oriented programming.

3.1 General Design Theory

The Design Process: The engineering design process is typically modelled as passing through several phases or stages. Pahl and Beitz [2], for example, describe the process with respect to four phases. During the clarification of the task, information is gathered, leading to detailed problem specifications. The conceptual phase results in solution variants, establishing functions and principles. During the embodiment phase, form is assigned to the concepts and functions. In the detailed design phase, the required properties of the individual components are analysed and perhaps optimized.

Based on this model of the design process, the traditional design catalogue only becomes a viable resource during the last, or perhaps the last two, design phases as it is only then that the design begins to take on a physical, more concrete character. Hence, a significant amount of the design process is already carried out before any reference is made to product catalogues. Thus, catalogues make no direct contribution to the early stages of design. If the information contained in catalogues is to impinge on these early stages, designers should be able to quickly access the data even while working on design concepts. There needs to be a link between the concepts and the data or, alternately, between function and form.

Connecting Function to Form: Gero *et al.*, [3] have attempted to bridge the gap between form (what they call structure) and function, proposing behaviour as the intermediary step or layer. They refer to this as function-behaviour-structure (FBS) design model. Within this model, function represents the teleological part of design, encapsulating the designer's intent. Behaviour states how the structure hopes to achieve the desired function. Structure refers not only to the individual components, but how these components are interconnected. The designer starts with a function in mind and, via teleological knowledge, conceives of some desired set of behaviours. Through knowledge of the structural composition of various components and the behaviours associated with these structures, a design is synthesized. Using causal knowledge, the actual behaviour of the synthesized design is determined and then compared to the desired behaviour. Any mismatch leads to refinement.

Gero develops a F-B-S hierarchical network. Some of the features of this network include two layers between behaviour and structure, namely, behaviour variables and structure variables, resulting in five layers in all. Behaviour variables are akin to the performance descriptors or performance variables of the artefact. Structure variables refer to the geometry and material of the artefact. There are also crossovers within the network, meaning that one can arrive at a given node from more than one direction. This is to be expected as a given component can fulfill more than one function. The links within the network are also bidirectional. One can move in one direction from function to structure to select a component. Having selected a component, one can then move in the other direction to determine the implications of that selection.

Gero elaborates on the model using the example of the design of a window. One function is that of providing daylight. The corresponding behaviour is light transmission. A variable associated with this behaviour is the light flux transmitted. The structure consists of glazing and a frame. The structure variables fall into several categories. Examples of a structure variables include window length (regular structure variable), glazing length (structure component variable) and glazing area (structure component behaviour variable).

The designer may postpone assignment of values to variables almost indefinitely. When assignment is finally carried out, there is normally a particular order that follows as the implications of the assigned values are propagated throughout the design space. For

instance, if glazing area and glazing length are assigned, then glazing width is necessarily defined and the window frame itself is also largely defined.

In a later work, Gero *et al.* [4] build on the FBS model. They outline 8 steps in the design process. Of particular interest are the final three steps. These three steps are all reformulations, dealing with a refinement of the design if the initial design (or design from previous cycle) is found unsatisfactory. The first reformulation is concerned with changes to the product's structure; the second, with changes to the product's behaviour; the third, with changes to the functions. The model assumes that if the product is found unsatisfactory, designers first assume the shortcomings can be addressed by finding alternatives to the structure. Designers are unlikely to move on to behaviour issues unless structural changes prove insufficient. Functional changes are then the last resort.

The concept of behaviour as an intermediary between function and structure is not without its critics. Dorst and Vermaas [5] see several problems with Gero's approach. They view Gero *et al.*'s concept of behaviour as providing a link between intentional descriptions and structural descriptions. Where this shift from intentional to structural descriptors occurs is rather unclear. They also note that certain behaviours do not depend solely on the properties of the window itself. Window behaviour, for instance, is partly attributable to external factors such as wind pressure. They question whether logic is actually sufficient to determine function from a given structure. Finally, if cost is part of structure, then structure, in addition to function, has intentional features for the cost of an item depends upon, to some extent, what people are willing to pay for it.

Despite the difficulty of the concept, behaviour can provide a suitable building block for the intelligent design catalogue. As with Gero *et al.*'s network, behaviour can take the form of several layers to facilitate a smoother function-form transition. The bidirectional quality allows designers to proceed from concept to some embodiment and then return to the concept should the embodiment present unforeseen problems. This, of course, speaks of the refinement process. As the refinement may take place in any part of the design process, bi-directional movement is essential to support the way design is normally carried out. Crossovers also greatly facilitate the refinement process, for one may use the same component to fulfill a different function, or use a different component to fulfill the same function. Thus, the intelligent design

catalogue may be hierarchical in its structure, but it need not be so in a strict sense. Crossovers must be allowed.

Gero *et al.* [4] frequently refer to different kinds of knowledge that are needed as one proceeds from function to structure. They mentioned five types in all: teleological knowledge (whereby the designer determines the specific relevant behaviours), knowledge of structure composition, knowledge of components' behaviour, computational or causal knowledge, and refinement knowledge. Ideally, the virtual designing catalogue should have access to as many of these knowledge bases as possible, as it reduces the knowledge burden on the user. Some knowledge bases can, of course, be more readily accommodated within the design catalogue than others. Both "knowledge of the structure composition" and "causal knowledge" are obvious choices for the catalogue, representing the component specifications and analytical evaluation tools, respectively. "Knowledge of components' behaviour" would be a desirable feature as it implies a more general, abstract description of components. "Refinement knowledge" entails developing some means by which the software can suggest "better" alternatives to the designer. This kind of knowledge can perhaps be captured by developing historical databases and suggestions are made based on what has been designed in the past. Finally, "teleological knowledge" represents the intent of the designer and, as such, is the most difficult to formalize. As with refinement knowledge, teleological knowledge can perhaps be partially captured using a historical database.

The goal of these various knowledge bases is to assist designers as they move back and forth through the layers between form and function, searching for the best component. Designers draw on these knowledge bases according to their particular needs and their particular level of expertise.

Designer Types: Engineering designers are typically classified according to two very broad groups: novice and experienced. The distinction is normally made based on the number of years of experience. Ahmed and Wallace [6], for example, define an experienced designer as one having ten or more years experience; the novice, a year or less. Within this time span lies transitional designers who are on their way to becoming experienced. Ahmed and Wallace recognize the simplicity of the definition, pointing out that being categorized as "experienced" was not based on the assessment of design skills.

Table 1: Differences Between Novice and Expert Designers

Criterion	Novice	Expert
Knowledge Type	Common sense	Domain specific
Memory type	Small chunks	Large chunks
Reasoning	Backward	Forward and Backward
Representation	Concrete	Abstract
Goal-limited strategies	Weak	Strong
General strategies	Weak	Strong
Approach	Trial and error	Skilled memory
Range	Decomposition	Deep levels

Presumably, the use of time as the sole determining factor of designer expertise is due to its ease of measurement.

Popovic [7] conducted studies, examining differences among novice, intermediate and experienced designers. For her research, novice designers were first-year undergraduate students. Intermediate designers were second and third year undergraduate students, and experienced designers consisted of postgraduate students with 3-10 years product design experience.

She developed several criteria which she used to assess the performance of the various design groups. Table 1 summarizes some of her findings. The “Intermediate” category has been omitted for the sake of simplicity.

Popovic distinguishes between general and goal-limited strategies. General strategies are used to keep the design on track in a general direction. Goal limited strategies attempt to solve a particular problem within the design project. Both the goal-limited and general strategies are weak among novices, but strong among the experts. Novices prefer to decompose the problem rather than solving it in a more integrative fashion.

Designers can also be distinguished with regard to the amount of domain-specific knowledge they possess and how they extract, analyse and apply that knowledge. Memory is stored as “chunks” and differences can be detected, not by the number of chunks, but rather by the size of the individual chunks. Not surprisingly, expert designers have greater domain knowledge than novices, the latter resorting to what might be termed “common sense”. The memory “chunks” become larger with design expertise, resulting in patterns within which large amounts of knowledge are organized. Expert designers display a greater ability to think on the abstract level, allowing them to develop rich, qualitative analyses, while novice designers tend to concentrate on the more

concrete aspects of the design problem and preferred to use trial and error rather than more analytical approaches.

Novices also favour backward reasoning (bottom-top), beginning with an instance of a designed object which is then related to the problem at hand. Experts use more forward reasoning, although Popovic points out that experts continue to use backward reasoning as well. Expert designers are also more apt to skip design steps than novice designers.

Many of Popovic's expertise classification schemes and research findings can play a potential role in the development of the intelligent design catalogue in terms of addressing the specific needs of novices or experts. For novice designers, designing catalogue should provide lots of room for experimentation (trial and error) and display individual components with a fair amount of detail upon request (decomposition). As both goal-oriented and general strategies were weak among novices, the environment should be able to provide some form of guidance. The guidance may take the form of suggesting a specific, individual component (goal-oriented) or a family of parts that often work together (more general). These guidance groups might represent some of the “chunks” to which Popovic refers. Experts, on the other hand, must be accommodated differently. They need to be able to skip over steps (layers?) or, at least, move through them very quickly. The design environment must accommodate designing at an abstract or conceptual basis. Ideally, some kind of analysis tools might be helpful, even at these upper levels. As experts engage in both forward and backward reasoning, designers must be able to move back and forth between function and form or, equivalently, abstract and concrete. This requirement would, of course, be addressed with Gero *et al.*'s bidirectional concept. If the intelligent design catalogue is to be a learning tool as well, *i.e.*, help turn novices into experts, the environment must be sufficiently transparent to allow novices to see how

they might change their approaches in order to design better.

In addition to differences between novices and experts with respect to the direction of reasoning (*i.e.*, forward or backward), the type of reasoning can also play a role in how designers design. Ball *et al.* [8] conducted case studies investigating analogical reasoning of novice and experienced industrial designers. Analogising refers to using information from a “source” problem to attempt to solve the a new, “target” problem. They considered two basic types of analogising: schema-based and case-based. Schema-based analogising refers to the use of schemas to solve problems. Schemas are abstract knowledge structures which are used to identify a class of problem and suggest an appropriate solution procedure for the new problem. Case-based analogising refers to designers drawing on a previous instance of a similar design problem and mapping the features of that problem to the new problem. Contentful, rather than abstracted, knowledge is used. As they expected, their research revealed that both novices and experienced designers carry out analogising, but more instances occurred with the experienced designers. Furthermore, most of the analogising done by the experienced designers was schema-based, whereas the novices concentrated on the case-driven type. With case-driven reasoning, Ball *et al.* also noted that knowledge use was dominated by surface cues of the target problem, rather than more abstract cues underlying the source problem's structure. There was true regardless of expertise.

This type of research, examining analogising among designers, is typically carried out using frameworks from psychology. Dunbar and Blanchette [9] provide some of the background for Ball *et al.*'s work. Dunbar and Blanchette looked at how scientists and politicians use analogising in their work. Scientists tended to use superficial analogies, whereas politicians tend to use analogies based on structural features. “Superficial” refers to “in house” types of analogies, *e.g.*, a biologist uses analogies from biology. “Structural” is used when the analogy comes from outside the domain. In this case, the “source” and “target” may look quite different initially, but similarity becomes evident as one probes more deeply. Thus, scientists tend to use analogies from science, but politicians find most of their analogies from outside of politics itself. When making analogies, scientists may spell out the “source” problem in great detail, but the mapping of the “source” problem to the “target” problem is not normally stated. It is up to those hearing the analogy to draw their own inferences.

As a common means of reasoning, the intelligent design catalogue should support the use of analogy for solving design problems. The analogies employed may be domain specific. For instance, one may use an electrical analogy as an aid to the design of some piping/plumbing system (or vice versa). Thus, engineers attempting to design a product outside of their normal area of expertise can request analogies to help them better understand the implications of their choices. By having a range of analogies available, the system will be able to accommodate designers from a broad range of backgrounds.

Language Metaphor: Another possible addition to the virtual designing catalogue comes from the work of Fauvel and Winkelmann [10]. This concept builds on the language analogy of design and seeks to develop a thesaurus for engineering design. With the language analogy, the form of the product is a word, and the function of the product is the meaning of the word. The thesaurus is aimed at providing a starting point for novice designers who more readily relate to structure than function. An example function is fastening. A particular form of this function might be a bolt. Novice designers are typically familiar with the term “bolt” rather than the function that it alludes to. The thesaurus makes this connection explicit and offers replacement parts that may fulfill the same function. Novices, for example, look up the word “bolt”, and find its meaning (*i.e.*, function), “fastening”. They then cross-reference to the heading “fastening” under which they find the “synonyms” for the word “bolt”, such as “nail”, “rivet”, etc. The intent of the thesaurus is to help bridge the function-form gap and expand the vocabulary of novice designers.

3.2 Manufacturing

Group Technology: Group Technology was being developed in the USSR during the 1950s [11] and was further developed over the next two decades in Europe [12]. The idea here was to organize manufacturing floors in work cells where similar parts could be machined within a small area. Thus was born the idea of part families.

The machines appropriate for each cell was determined using a matrix, where the rows contain the parts to be manufactured and their operations, and the columns contain the machines that carried out the necessary machining operations. By rearranging the rows, smaller clusters within the matrix are formed and these clusters identify the machines within a given cell as well as the parts to be manufactured there [13].

Typically, some outliers remain and must be accommodated by having parts travel between cells.

The idea behind GT bears a striking resemblance to the work of Alexander [14] in the area of design. Using set theory, he grouped design requirements such that the interaction between these groups is minimized. These groups represent design modules. As the interactions were minimized, design changes made to one module would have minimal impact on other modules. The design module is thus akin to a manufacturing cell of GT.

We can conceive of the intelligent design catalogue as being a shop floor. A particular machine corresponds to a specific part and the cells of the shop floor become the categories of part families within the catalogue. On the manufacturing shop floor, family resemblances are based on how the parts are manufactured. These same categories are not necessarily appropriate for design using standard components. Classification according to function or particular features would be more helpful. Parts can also be grouped according to typical uses. For instance, a fan, electric motor, motor mount and electrical connections can be presented within a single group. Unlike the shop floor, the intelligent design catalogue need not be restricted to a single classification scheme. Thus, a single component can be located using one of many possible classification schemes. Within the virtual design environment, we can very easily relocate the “machine” to another “cell” on the “shop floor”, as it were.

Reconfigurable Systems: Oldknow and Yellowley [15] describe a reconfigurable, open-architecture system to control machine tools on a manufacturing shop floor. One of their main goals is to develop a system that can accept hardware (e.g., CNC machines) from any number of vendors (vendor neutrality). The system consists of a Virtual Machine Tool (hardware independent objects) and hardware-dependent methods. Between these two sides is the Binding Table which allocates hardware components to the functions demanded by the virtual environment. Thanks to the Binding Table, one can design a manufacturing process using the Virtual Machine Tool without any specific knowledge of the machines available. The idea of reconfigurability refers to the ease with which a machine in the system can be removed or added. Their system offers dynamic reconfigurability which means that reconfiguration can be carried out without necessitating a complete shut-down of the system. Another added feature of the system is that it allows for constraints to be added to

the machining process. Thus, the machining speed can be altered, for example, if torque limits are exceeded.

The intelligent design catalogue can be modelled after the reconfigurable system. As with GT, the catalogued components are the CNC machines that the system operates. Vendor neutrality means that the catalogue need not be confined to a single source of components. The Virtual Machine Tool becomes the virtual design environment where the designer can carry out conceptual and some embodiment design, sometimes unaware of the actual components that are available. Gero *et al.*'s behaviour or a similar concept replaces the Binding Table as it translates functional or conceptual specifications to increasingly well-defined components. Dynamic reconfiguration allows, first, new components to be added to the catalogue without interrupting catalogue use and, second, components to be added or removed from an assembly within the design environment without necessitating a complete redesign. As one adjusts the machining process to account for process constraints, one can also restrict the available components in the catalogue in keeping with designer-imposed constraints (e.g., cost, weight). The open architecture of the system means that those who would sponsor the intelligent design catalogue (e.g., suppliers) can change the database of components according to their own needs and are not obliged to engage the help of “experts” from the outside.

The system proposed by Oldknow and Yellowley [15] can be considered to be a Reconfigurable Manufacturing System (RMS). Mehrabi *et al.* [16] and Koren, *et al.* [17] compare RMSs to Flexible Manufacturing Systems (FMS). FMSs are designed to accommodate a wide range of possible machining processes. Unfortunately, in most FMS installations, the full flexibility offered by the system has gone unrealized as the number of parts manufactured in the long run failed to justify the initial investment [16]. RMS was developed to address this shortcoming of FMS. Although less flexible than FMSs, RMSs do provide a reasonable range of manufacturing capabilities at a reduced cost. RMS is thus an attractive alternative to many manufacturers.

The failure of many FMSs and the rise of RMS can serve as a warning to the intelligent design catalogue. The catalogue, though virtual, need not be comprehensive with every possible part listed. Catalogues may better serve their clients if they are confined in scope. This is particularly true of routine designs where designers tend to restrict themselves to

a limited range of fairly predictable components. As the catalogue content is extended to include a wider range of components, navigation becomes more difficult. However, as unique machining operations may justify an FMS, non-routine, more innovative design, requires a more comprehensive catalogue. A comprehensive catalogue is thus akin to a Flexible Manufacturing System, a limited catalogue to an RMS.

3.3 Object-Oriented Programming

Object-oriented programming (OOP), as the name implies, uses the metaphor of an object to develop a programming language. Hence, OOP focuses on the *what* of programming. By way of contrast, procedural languages attempt to address the question *how* [18].

As an object, it is a conceptual unit, combining both data and methods. A method is defined as a procedure or function that alters the state of an object or causes the object to send a message (*i.e.*, return values) [19]. A well designed object can serve many purposes, and fosters reuseability, often cited as one of the main benefits of OOP (some, however, remain skeptical of this claim).

Another important feature of OOP is encapsulation. Encapsulation refers to the hiding of information within an object, information not required by other objects. Encapsulation allows changes to be made to the internal workings of the object (implementation) while leaving its interface, where it receives messages from other objects, intact. Encapsulation thus promotes modularity.

Objects receiving the same message need not respond in the same way. This is referred to as polymorphism. The message “addition”, for example, may mean arithmetic addition for one object (addition of numbers), but concatenation to another (addition of words).

Another important feature of OOP is inheritance. Inheritance means that certain objects inherit attributes (but not data) from others. Inheritance necessarily creates various levels of abstraction, leading to a hierarchy consisting of superclasses and subclasses. (An object is not a class itself, but an instance of a class.) According to Wirfs-Brock [18], attributes should be as high up the hierarchy (more abstract) as possible. This ideal is promoted by allowing some of the attributes of the lower objects to override the inherited values, leading to polymorphism.

Typically, an object inherits all of its attributes from a single class (which may inherit from yet another class above it). Multiple inheritance refers to an

object or class inheriting from two or more classes at the same level of the hierarchy. Multiple inheritance, however, should be used sparingly as it may introduce some ambiguity [20], for if an object inherits from two classes that are both on the same level with methods that bear the same name, which method does the object inherit?

Many of the features of OOP map well to the intelligent design catalogue, for many of the issues OOP tries to address are design problems. Perhaps the most obvious feature of OOP is that of the object, analogous to a fully specified component. The close conceptual coupling of the object to the standard component suggests that OOP is a good choice for programming the virtual environment of the intelligent design catalogue. Like objects, reuseability is an important feature of standard components as a single component is expected to be part of many different assemblies. Components can be classified as belonging to part families, the classes of OOP. Increasing the level of abstraction leads to a functional description of the part. Inheritance within classes and objects describes how a single function can be fulfilled by any one of a number of components. Unlike OOP, the intelligent design catalogue must make use of multiple inheritance if it is to foster any kind of creative design. Within design, multiple inheritance refers to a single component fulfilling more than one function. A bolt, for instance, can be used as a fastener, or it can be used to position a mechanism. The need for multiple inheritance speaks to the complexity of design.

Within the intelligent design catalogue, encapsulation occurs at two levels. At the catalogue level, encapsulation refers to the internal workings of a standard component. Generally, when purchasing a component, the buyer considers the specifications (attributes) of the component, not its precise internal mechanisms. An important specification is the interface of the component as it determines to which other components it can be connected, just as the interface of an OOP object determines how it communicates with other objects. At the design level, encapsulation allows one to design in terms of concepts or functions. One may specify “fastener”, for example, without giving any thought to whether the fastener be a bolt or a blob of glue. In this case, the interface becomes more akin to the concept of behaviour.

Polymorphism applies to the intelligent design catalogue at two levels. On one level, polymorphism can be construed as the activation of various

descriptive or analytical tools depending on the component type. The message “show specifications”, for example, would be quite different for a bolt than an electric motor. Likewise, the “analyse” command would invoke different mathematical formulas, based on the type of component under analysis. As polymorphism in OOP is typically implemented by overloading, it speaks to a second level of application. Within the design environment, overloading (overriding) is carried out by imposing design constraints. A component normally selected for a given application may be deemed invalid if it violates a design constraint. Hence, design constraints parallel polymorphism.

4 Linking of Manufacturing, OOP and Design

Concepts from design, manufacturing and OOP have much to offer the development of an intelligent design catalogue. The foregoing discussion demonstrates that all three areas have much in common. The equivalent features of all three are summarized below and compiled in Table 2.

The basic unit of interest is the CNC machine in manufacturing, the object in OOP and the standard component of the catalogue. CNC machines are grouped together in manufacturing cells, objects in classes and components in part families. The sum total of CNC machines, objects and components become the shop floor, the program and assembly (or catalogue), respectively. Each machine performs a certain set of operations, just as objects have particular methods and components perform certain functions.

A good manufacturing system allows for reconfigurability, where one machine can be replaced

with another without rendering the entire shop floor inoperative. In OOP, objects can be replaced with other similar objects while leaving the rest of the program intact. The intelligent design catalogue allows a component to be changed in the assembly without disrupting the rest of the assembly or, equivalently, replacing segments of the catalogue of standard components.

The virtual machining environment allows for the specification of certain operations without specifying the machine. The assignment to the actual machines is carried out by the binding table. Within the OOP programming environment, one can conceivably develop a program to perform certain operations before the actual objects are created or selected from a library. The virtual design environment allows for the specification of certain functions without naming a part family or a particular component. Behaviour is used to translate the required function to a candidate component.

The process control to manufacture a specific component can be modified in keeping with process constraints. As a machining tool becomes dull, for example, the feed rate may be reduced to keep torque requirements below acceptable limits. Process constraints allow the normal process setpoints to be overridden. Within the class structure of OOP, methods can be overridden within particular classes, allowing for polymorphism. Within the intelligent design catalogue, functional specifications can be overridden if a given component violates certain design constraints, *e.g.*, the component is too heavy.

Within the virtual machining environment, the binding table assigns the high-level (abstract) machining operations to particular machines on the shop floor. It acts as the interface between the virtual

Table 2: Mapping manufacturing and object-oriented programming to the intelligent design catalogue.

Manufacturing	Object Oriented Programming	Intelligent Design Catalogue
CNC machine	Object	Specified component
GT, group cell	Class	Part family
List of machines	Object library	Catalogue
Shop floor	Program	Assembly
Virtual machining environment	Virtual programming environment	Virtual design environment
Reconfigurability	Modularization	Part replacement
Process constraints	Polymorphism	Design constraints
Binding table	Interface	Behaviour, interface
Operation	Method	Function
Machine unspecified	Encapsulation	Component/part family unspecified
One process, many machines	Simple inheritance	One function, many parts
One machine, many processes	Multiple inheritance	One part, many functions

Figure 1: Circular Representation of Theoretical Framework

and the real and allows an operator to work on the abstract process without being aware of the machines available. In OOP, objects, too, have interfaces and objects interact with each other through these interfaces. Encapsulation allows for the hiding of information not needed by other objects. Behaviour acts as the interface in the design environment, freeing the designer from the need of a thorough understanding of the catalogue contents. In the catalogue environment, the designer can assemble components based on their physical interfaces while having no knowledge of the internal mechanisms of the components themselves.

When designing a machining process for the manufacture of an individual part, one may assign any one of a number of machines to carry out the operations. Conversely, an individual machine can perform a number of operations. Inheritance in OOP means that superclass can pass its attributes to any one of a number of subclasses. Multiple inheritance means that a subclass can inherit attributes from any number of superclasses. Within design, a function can be carried out by many different components. Conversely, a single component may fulfill a number of functions.

5 Circular Representation

Although design, manufacturing and object-oriented programming contain many equivalent concepts, they also exhibit distinct differences. These differences help define the particular strengths and weaknesses with respect to each other. By drawing on the conceptual strengths of each, the intelligent design catalogue promises to have a firm theoretical base.

The three areas can be arranged in a circular fashion, with the areas joined by their shared conceptual strengths. The circularity reinforces the inter-connectedness of their conceptual frameworks. This idea is presented in Figure 1. Case study research has been separated from design theory as presented previously to produce four areas rather than three. The fourth area provides greater coherence within the circular arrangement.

Design theory is concerned with the design process, how one starts with a concept and develops a tangible product. Studying the differences between novices and experts may help our understanding of how a good design is carried out. Such research is normally conducted as case studies within an industrial setting.

Manufacturing can offer such an industrial setting conducive to testing theories and conducting case

studies. Within manufacturing, Group Technology offers approaches to the classification of both the parts and the machines to create them. Much of the process design is now done on computers in a virtual environment and a binding table serves to link the virtual environment with the shop floor. The virtual environment facilitates reconfigurability.

OOP is one such virtual environment where the manufacturing process can be designed. It promotes the idea of object, where data and methods are inherently linked together. Objects are also conducive to classification. As a type of programming language, OOP naturally emphasizes language itself, for the entire program is composed of words.

Similar to OOP, design is very much concerned with the combining of objects and methods, or components and functions. Language is also a very common metaphor in design. The thesaurus mentioned above is one of many examples.

Thus, all four groups can be conceptually linked together in a circular fashion to provide a coherent framework from which to develop the intelligent design catalogue.

6 Summary

The intelligent design catalogue combines a virtual design environment with catalogues of standard engineering components. Components are assembled within the design environment. A particular feature of this environment is that one may assemble functional concepts which are then successively refined to increasingly concrete forms until one arrives at fully specified components from the catalogue. Potentially, analytical tools assess the performance of the various components within the assembly. Within the catalogue environment, design aids are presented to facilitate the retrieval of the appropriate component. These aids are aimed at the more novice designers; experts may ignore these aids and select components directly from the catalogue.

Combining a design environment with an engineering catalogue presents many challenges, for design itself is a very complex activity. Fortunately, issues similar to those encountered in the intelligent design catalogue have been investigated in other areas, thus offering potential solutions. For this particular research, we looked to design, manufacturing and object-oriented programming to provide some guidance.

Theories from design offer ways to facilitate the movement from initial design concepts to standard components, *i.e.*, from function to form. Behaviour

provides a way that one might bridge the gap between the two. Case study research in design provides insights into the different approaches of novice and expert designers, suggesting design aids for novice designers. One such aid is to use analogies so that the designer can better relate to the unfamiliar component or design problem.

From manufacturing, Group Technology points to ways that parts can be organized in groups. In GT, parts are grouped to facilitate machining; in the intelligent design catalogue, components may be grouped a number of ways, such as by function or assembly patterns, to facilitate retrieval. The binding table of the reconfigurable manufacturing system reminds us that a great deal of design can be carried out with limited knowledge of the actual components available. The inclusion of process constraints in the model suggests that the intelligent design catalogue incorporate design constraints.

Object-oriented programming focuses on the object. An object combines both data and methods and maps well to the standard component and its function of the intelligent design catalogue. Encapsulation, like the binding table of manufacturing, tells us that some information is best hidden from the user as hiding can simplify tasks and promote reuseability.

Design, manufacturing and object-oriented programming have a great deal in common and equivalent concepts can be seen across the three. They complement each other as each has its particular strengths and weaknesses. By drawing on the strengths of each, the intelligent design catalogue promises to reach its goals.

7 References

- [1] I. Yellowley and P. Winkelman, "A simple proposal for the development of Intelligent Design Catalogs", presented at the Second CDEN International Conference on Design Education, Innovation, and Practice, Kananaskis, Alberta, July, 2005.
- [2] G. Pahl and W. Beitz, *Engineering Design: A Systematic Approach*, trans. by Arnold Pomerans and Ken Wallace, Springer-Verlag, London, 1988.
- [3] J.S. Gero, K.W. Tham and H.S. Lee, "Behaviour: a link between function and structure in design", in D.C Brown, M. Waldron and H. Yoshikawa (eds.), *Intelligent Computer Aided Design*, North Holland (Elsevier), Amsterdam, pp. 193-220, 1992.
- [4] John S. Gero, Udo Kannengiesser, "The situated Function-Behaviour-Structure framework", *Design Studies*, Vol. 25, pp. 373-391, 2004.
- [5] Kees Dorst and Pieter E. Vermaas, "John Gero's Function-Behaviour-Structure model of designing: a critical analysis", *Research in Engineering Design*, Vol. 16, pp.

- 17-26, 2005.
- [6] S. Ahmed and K.M. Wallace, "Understanding the knowledge needs of novice designers in the aerospace industry", *Design Studies*, Vol. 25, pp. 155-173, 2004.
 - [7] Vesna Popovic, "Expertise development in product design—strategic and domain-specific knowledge connection", *Design Studies*, Vol. 25, pp. 527–545, 2004.
 - [8] Linden J. Ball, Thomas C. Ormerod and Nicola J. Morley, "Spontaneous analogising in engineering design: a comparative analysis of experts and novices", *Design Studies*, Vol. 25, pp. 495-508, 2004.
 - [9] Kevin Dunbar and Isabelle Blanchette, "The in vivo/in vitro approach to cognition: the case of analogy", *Trends in Cognitive Sciences*, Vol. 5, pp. 334-339, 2001.
 - [10] O. Fauvel and P. Winkelman, "Organization of technological information for the novice mechanical designer", presented at the International Workshop on Pedagogics in Design Education, Pilsen, Czech Republic, 1998.
 - [11] S.P. Mitrofanov, *Scientific Principles of Group Technology*, English Translation, National Library for Science and Technology, Washington, D.C., 1966.
 - [12] H. Opitz, *A Classification to Describe Workpieces*, Pergamon Press, Oxford, 1970.
 - [13] C.C. Gallagher, W.A.Knight, *Group Technology Production Methods in Manufacture*, Ellis Horwood, Limited, Chichester, England, 1986.
 - [14] Christopher Alexander, *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, 1964.
 - [15] K.D. Oldknow and I. Yellowley, "The design, implementation and validation of a system for the dynamic reconfiguration of open architecture machine tool controls", *Int. J. Mach. Tools and Manufacture*, Vol 41, pp. 795-908, 2001.
 - [16] M.G. Mehrabi, A.G. Ulsoy, Y. Koren and P. Heytler, "Trends and perspectives in flexible and reconfigurable manufacturing systems", *Journal of Intelligent Manufacturing*, Vol. 13, 135-146, 2002.
 - [17] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy and H. van Brussel, "Reconfigurable manufacturing systems", *Annals of the CIRP*, Vol. 48, pp. 527-540, 1999.
 - [18] Rebecca Wirfs-Brock, Brian Wilkerson and Lauren Wiener, *Designing Object-Oriented Software*, P T R Prentice Hall, Englewood Cliffs, New Jersey, 1990.
 - [19] Ian Graham, *Object Oriented Methods*, Addison-Wesley Publishing Company, Wokingham, England, 1991.
 - [20] Mark Lutz and David Ascher, *Learning Python*, O'Reilly, Sebastopol, CA, USA, 2004.