AI Magazine Volume 11 Number 4 (1990) (© AAAI)

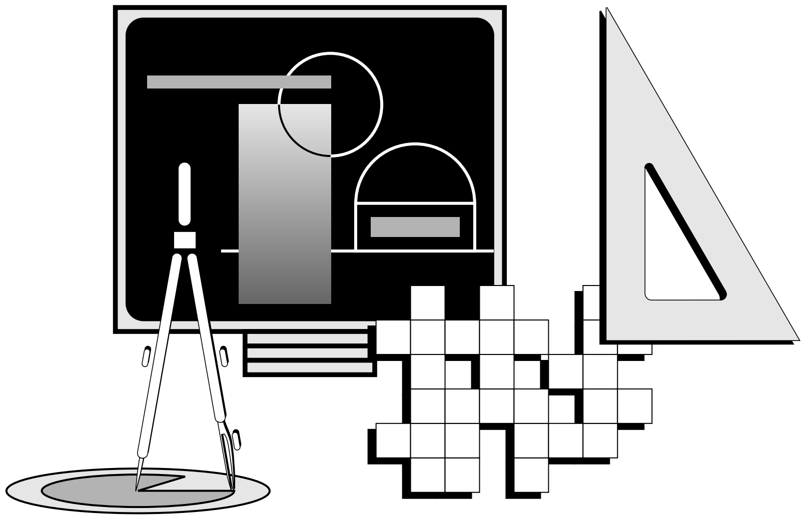
Modeling Design Processes

Hideaki Takeda, Paul Veerkamp, Tetsuo Tomiyama, and Hiroyuki Yoshikawa

This article discusses building a computable design process model, which is a prerequisite for realizing intelligent computer-aided design systems. First, we introduce general design theory, from which a descriptive model of design processes is derived. In this model, the concept of metamodels plays a crucial role in describing the evolutionary nature of design. Second, we show a cognitive design process model obtained by observing design processes using a protocol analysis method. We then discuss a computable model that can explain most parts of the cognitive model and also interpret the descriptive model. In the computable model, a design process is regarded as an iterative logical process realized by abduction, deduction, and circumscription. We implemented a design simulator that can trace design processes in which design specifications and design solutions are gradually revised as the design proceeds.

One of the major problems in developing so-called intelligent computer-aided design (CAD) systems (ten Hagen and Tomiyama 1987) is the representation of design knowledge, which is a two-part process: the representation of design objects and the representation of design processes. We believe that intelligent CAD systems will be fully realized only when these two types of representation are integrated. Progress has been made in the representation of design objects, as can be seen, for example, in geometric modeling; however, almost no significant results have been seen in the representation of design processes, which implies that we need a design theory to formalize them.

According to Finger and Dixon (1989), design process models can be categorized into a descriptive model that explains how design is done, a cognitive model that explains the designer's behavior, a prescriptive model that shows how design must be done, and a computable model that expresses a method by which a computer can accomplish a task. A design theory for intelligent CAD is not useful when it is merely descriptive or cognitive; it must also be computable. We need a general model of design to clarify and define what design is from a theoretical point of view, which is a role of the descriptive model. However, descriptive models are not necessarily helpful in directly deriving either the architecture of intelligent CAD or the knowledge representation for intelligent CAD. For this purpose, we need a computable design process model that should coincide, at least to some extent, with a cognitive model that explains actual design activities.



GDT is a descriptive model that tries to explain how design is conceptually performed in terms of knowledge manipulation.

We start this article with the descriptive model. We present our descriptive theory called general design theory (GDT). It is a mathematical formulation of design processes and explains how design is conceptually performed in terms of knowledge manipulation. Next, we show a more concrete descriptive model called the evolutionary design process model. It is based on GDT and shows how the design object is manipulated during the design process in an intelligent CAD environment. We then analyze design processes based on experimental results. We conducted design experiments to extract the features of design processes from which a cognitive design model was derived.

These discussions lead to a logical formalization of design processes in terms of a design process model that is not only consistent with the descriptive and cognitive models but also expected to be computable. In this model, a design process is regarded as a logical process in which both the theory (that is, axioms) and the goal are gradually revised as the design proceeds, using abduction, deduction, and circumscription. *Abduction* is used to expand the designer's thought, *deduction* is used when the designer wants to get all obtainable facts from currently available design knowledge and design object descriptions, and *circumscription* is applied to solve an inconsistency found during deductive reasoning. We developed a design simulator to demonstrate that this model is computable and also appropriate to represent design processes. An example is shown in which given design knowledge and specifications, the design simulator logically simulates a design process based on protocol data obtained by a real design experiment. In the last section, we discuss the relationship between the descriptive, cognitive, and computable design process models.

General Design Theory

GDT's major achievements are a mathematical formulation of the design process and a justification of knowledge representation techniques in a certain situation (Yoshikawa 1981; Tomiyama and Yoshikawa 1987). GDT is a descriptive model that tries to explain how design is conceptually performed in terms of knowledge manipulation. In GDT, a design process is regarded as a mapping from the function space to the attribute space, both of which are defined over the entity concept set. Based on axiomatic set theory, we can mathematically derive interesting theorems that can well explain a design process. (Interested readers can refer to Yoshikawa [1981] and Tomiyama and Yoshikawa [1987] for detailed discussions. In this article, we only present the theorems.)

Design in the Ideal Knowledge

GDT deals with concepts that only exist in our mental recognition. In this sense, GDT is an abstract theory about knowledge. We make a distinction between an entity and an entity concept. An *entity* is a concrete existing thing, and an *entity concept* is its abstract mental impression conceived by a human being. An entity concept might be associated with its properties, such as color, size, function, and place. These properties are called *abstract concepts* and include attributes.

Axiom 1 (axiom of recognition): Any entity can be recognized or described by attributes, other abstract concepts, or both.

Axiom 2 (axiom of correspondence): The entity set S' and the set of entity concepts (ideal) S have one-to-one correspondence.

Axiom 3 (axiom of operation): The set of abstract concepts is a topology of the set of entity concepts.

Axiom 2 guarantees the existence of a superhuman who knows everything. In other words, it defines an ideal, ultimate state at which our knowledge should aim. Axiom 3 signifies that it is possible to logically operate abstract concepts as if they were just ordinary mathematical sets. Accordingly, we get set operations, such as intersection, union, and negation.

We can then introduce ideal knowledge that knows all the elements of the entity set and that can describe each element by abstract concepts without ambiguity. Theorem 1 mathematically describes this situation.

Theorem 1: The ideal knowledge is a Hausdorff space.

In GDT, a design specification is given as an abstract concept that the design solution must belong to. Thus, the specifications can be given by describing an entity with only abstract concepts (for example, functionally). The *function space* is the entity concept set with a topology of functions, and the *attribute space* is the one with a topology of attributes. Therefore, a *design specification* is a point in the function space, and a *design solution* is a point in the attribute space. The most significant result of having the ideal knowledge that can be further proven from the three axioms is that design as a mapping from the function space to the attribute space successfully terminates when the specifications are described.

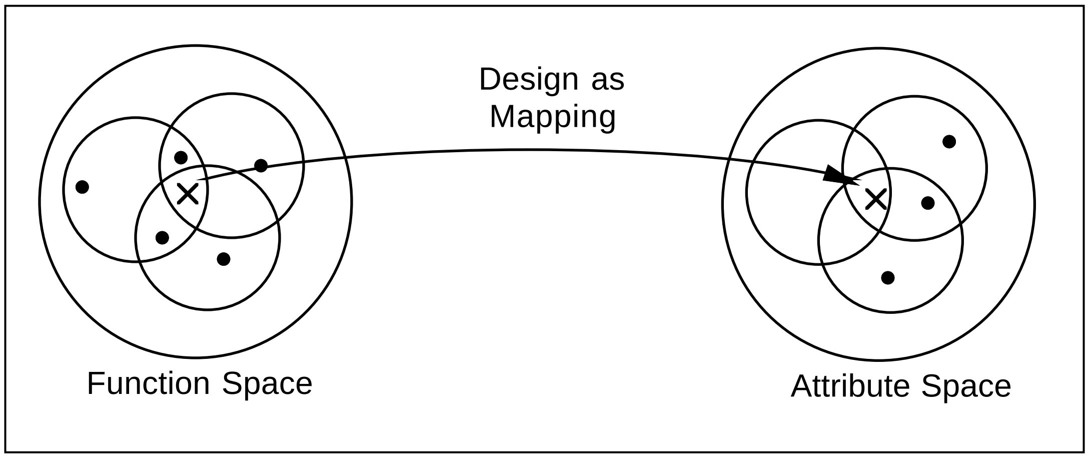


Figure 1. Design Process in the Ideal Knowledge.

Theorem 2: In the ideal knowledge, the design solution is immediately obtained after the specifications are described.

Because we know everything perfectly in the ideal knowledge, when we finish describing the specifications, they converge to a point in the function space. Because the function space and the attribute space are built on the same entity concept set, this point (that is, an entity concept) can also be considered in the attribute space. The ideal knowledge is also perfect to describe an entity concept in the attribute space. Thus, the design solution will be fully described by attributes; that is, the design in the ideal knowledge is a mapping process from the function space to the attribute space (figure 1).

Design in the Real Knowledge

The situation in the ideal knowledge is not the case in the real design, and we need to consider the following characteristics: First, design is not a simple mapping process but rather a stepwise refinement process where the designer seeks the solution that satisfies the constraints. Second, the concept of function is difficult to objectively formalize because it includes a sense of value that can vary from person to person. Instead, we use behavior to deal with function. Third, the ideal knowledge does not take physical constraints into consideration, and it can produce design solutions such as perpetual machines.

These restrictions are considered in the real knowledge, where design is regarded as a process in which the designer builds the goal and tries to satisfy the specifications without violating physical constraints. To formalize the real knowledge, we first define a *physical law* as a description about the relationship between physical quantities of entities and the field. The concept of physical laws is one of the abstract concepts formed when one looks at a physical phenomenon as a manifestation of physical laws. Physical laws constrain entities in the real world; in other words, any feasible entity must be explicable by physical laws. This fact can be proven as a theorem.

Theorem 3: The set of physical law concepts is a base of the attribute concept topology of the set of (feasible) entity concepts.

This theorem states that attributes can be measured by using physical laws. An interesting fact about the real knowledge is that we can prove finiteness or boundedness of our knowledge with the following hypothesis.

Hypothesis: Finite subcoverings exist for any coverings of the set of feasible entity concepts made of sets chosen from the set of physical law concepts.

Basically, this hypothesis says that a feasible entity is explicable not by an infinite number but a finite number (as small as possible) of physical laws. From this hypothesis, we can prove an interesting theorem that explains that an attribute has a value if it is possible at all to measure the distance. We can also mathematically prove theorems 4 and 5.

Theorem 4: The real knowledge is a compact Hausdorff space.

Theorem 5: In the real knowledge, if we can produce a directed subsequence from the given design specifications, this subsequence converges to a single point.

Theorem 5 indicates that in the real knowledge, a design process can be regarded as a convergence process if we can pick up some

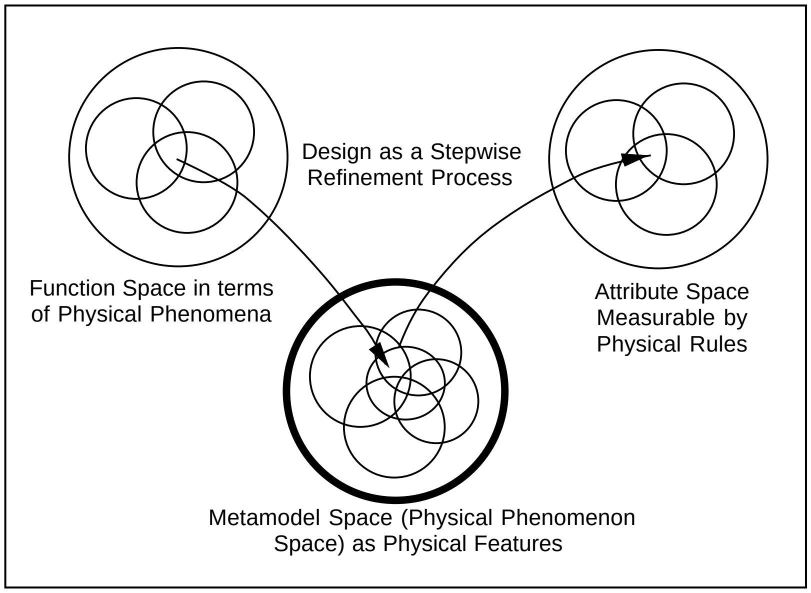

Design as a Stepwise Refinement Process
Function Space in terms of Physical Phenomena
Attribute Space Measurable by Physical Rules
Metamodel Space (Physical Phenomenon Space) as Physical Features

Figure 2. Design Process in the Real Knowledge.

(but not necessarily all) specifications that seem to yield meaningful solutions.

Evolutionary Design Process Model

The next step is to formalize design processes in the real knowledge. For this purpose, the concept of metamodels is formally introduced, where a *metamodel* is a finite set of attributes, and the *metamodel set* is the set of all metamodels. Such a metamodel can be evolved; that is, we mean to increase the number of attribute concepts.

Theorem 6: If we evolve a metamodel, we get an entity concept as the limit of evolution.

Theorem 6 is a corollary to theorem 5, and it does not guarantee that we obtain a design solution as the result of this evolution. However, if we restrict ourselves to consider function only in terms of behavior, theorem 7 can be proven.

Theorem 7: If concepts explained by physical law concepts are chosen for the metamodel, then a design solution exists that is an element of this metamodel.

Theorem 7 guarantees that we are able to design as long as specifications are given in terms of (physical) behaviors and solutions are described in terms of attributes that can be measured by physical laws. Furthermore, solutions contain only those attributes that can be physically realized; in other words, we are not allowed to consider objects that contradict physical laws. Figure 2 depicts a design process in the real knowledge in which we design, in fact, physical behaviors of the design object.

Theorem 6 also indicates that a design process is a stepwise transformation process, and solutions are obtained in a gradual refinement manner because the metamodel can be evolved only by increasing the number of attributes. In the ideal knowledge, design is a direct mapping process from the function space to the attribute space, but in the real knowledge, design is a stepwise, evolutionary transformation process. The difference is elaborated in the next section.

Descriptive Design Process Model

In this section, we present a descriptive design process model that shows design as a gradual transformation from the function space to the attribute space. This model is used to build a framework that directs the design process to be implemented in an intelligent CAD system. Knowledge about the design process is embedded in the framework. Thus, the system is always informed about the current stage of the design process and the current state of the design object. The descriptive design process model is presented in system-oriented terminology. The knowledge about how to perform the design process and which tools are applicable at a certain instant is denoted by means of scenarios. These scenarios represent design schemes that describe what kind of actions must be carried out at a specific phase of the design process. For each phase of the design process, there is a different set of scenarios.

The descriptive model is used as a basis for developing an intelligent CAD system, and the computable model, discussed later, simulates the reasoning aspect of design. Both of them can be implemented on a computer. However, the descriptive model is meant as a guide to discuss the architecture of an intelligent CAD system, and the computable model is more interesting for illustrating that design as a reasoning process can be performed by a computer.

From Specification to Solution

We use GDT as a basis for giving a descriptive design process model. The basic ideas behind the descriptive design process model are derived from the evolutionary design process model, as follows: First, from the given functional specifications, a candidate for the design solution is selected and refined in a stepwise manner until a complete solution is obtained. This approach is used rather than trying to directly obtain the solution from the specifications because a nontrivial design problem involves a complex object with a multitude of parts. Second, the design process is regarded as an evolutionary process that transfers the model of the design object from one state to another, gradually obtaining a more detailed description. The number of attributes grows as the design process proceeds, and a growing number of the functional specifications are met. Finally, to evaluate the current state of the design object model, various interpretations of the design object model need to be derived to see whether the object satisfies the specifications.

We call these interpretations of the design object model *contexts*. Contexts allow a designer to model the current state of the design object in a certain environment; that is, they represent an aspect model. More information about the design object is obtained through these contexts, and hence, the number of attributes grows. Contexts are created by means of scenarios that contain the design knowledge and data necessary to build an aspect model. Scenarios perform the reasoning about a context, and they lead the dialogue with the designer.

Stepwise Refinement of Metamodel

Considering GDT, a designer starts with the functional specification of a design object and continues the design process until a design solution is obtained. During this process, the design object model is refined in a stepwise manner. The central description of the design object is regarded as *metamodel* in GDT because it is used during the design process as a central model from which aspect models are derived.

A metamodel is a description of the design object that is independent of a context. It contains all entities the design object is composed of, and it includes the relationships and dependencies among these entities. Data that are used in one of the contexts derived from the metamodel are stored apart from the metamodel. For instance, geometric data of the design object are not found in the metamodel but in a geometric aspect model.

The stepwise refinement process shown in figure 3 behaves as follows: At a certain stage of the design process, the metamodel M_{i-1} is the current, incomplete description of the design object. To get a detailed description, an aspect model is derived from the metamodel. Through this aspect model, some new information about the design object is obtained. After this refinement, the new information from the aspect model is merged into the metamodel M_{i-1}. If the merge is successful—that is, the new information is consistent with the current M_{i-1}—then the result of the merge is a new state of the metamodel, M_i. This process continues, obtaining M_{i+1}, and so on, until the design object model finally becomes a complete and satisfactory description of the desired artifact. Here, *complete* means satisfying all initial requirements. As a matter of fact, a design can never be complete; something can always be improved or made cheaper. In this context, the term complete has a rather subjective meaning.

. . . we present a descriptive design process model that shows design as a gradual transformation from the function space to the attribute space.

Metamodel Evolution Scheme

The descriptive design process model, as discussed so far, gives a global outline of the design process without going into details. We now focus on how to transfer from one state of the metamodel to the next; that is, how do we perform a design step? Here, we elaborate on the concept of a context. Through the creation of a context, the designer provides an interpretation of metamodel in a certain environment; that is, s/he generates an aspect model. For each design step, an associated scenario exists that creates a context that the design object is modeled in. In a context, new information about a design object is obtained. The current metamodel, together with the new information, form the next metamodel.

The metamodel mechanism is a sequence of design steps (Veth 1987; Veerkamp, Pieters Kwiers, and ten Hagen 1990). A design step is performed by executing a design scenario. A *scenario* consists of design procedures and design rules that describe the procedural and declarative design knowledge, respectively. The rules and procedures are applied to the metamodel and the aspect data. In such a manner, we create a context that consists of (a part of) the metamodel together with knowledge about the design object valid in this particular context. The designer can interact with the context and change its contents. After the session, the contents of the context being evaluated are mapped to the next state of the metamodel. Details of the

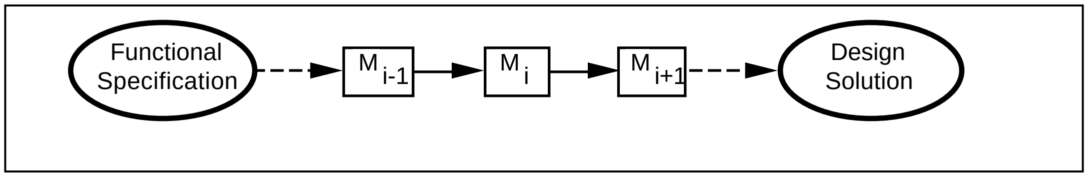


Figure 3. Stepwise Refinement of the Metamodel.

relation between the metamodel mechanism and aspect models are given in Veerkamp, Kiriyama, Xue, and Tomiyama (1990).

The metamodel mechanism transfers the metamodel from its current state to the next, according to the stepwise refinement model. The mechanism is driven by scenarios. The designer selects a design scenario that is appropriate for the current state of the metamodel M_i. The scenario is executed in a co text c_i and performs some action on the context in dialogue with the designer. The execution of a scenario continues as long as new information can be obtained in the context. The acquisition of new information is accomplished through the design procedures and the design rules in a scenario.

The contents of a context are evaluated when the execution of its scenario is completed. The evaluation checks the context c_i for consistency with the metamodel M_i.; that is, no facts can contradict each other, and all constraints over the design object model must be met. The metamodel M_i is transferred to M_{i+1} if the evaluation succeeds. In case of failure, all results of ci are discarded, and the process restarts from M_i (figure 4). This backtracking is performed in dialogue with the designer so that the next attempt can be made more successful.

A consecutive application of the metamodel evolution mechanism enables the designer to perform a design job. Hence, the design process consists of the execution of a series of design scenarios. For each state of the metamodel, an appropriate scenario exists. The descriptive design process model presented here is derived from GDT, providing a framework that allows for design process representation. The metamodel mechanism is part of this framework and depicts the evolutionary nature of the design process; that is, it represents a design step. The design process, therefore, can be modeled as a stepwise refinement process.

In this and the previous three subsections, we obtained a descriptive design process model. This model clarifies the representation of the design object that dynamically changes. We also suggested a way to apply this model to building intelligent CAD systems. It seems reasonably interesting to extend this descriptive model to a computable model that can clarify the reasoning process during the design process. However, we do not just discuss design processes from a theoretical viewpoint. Because both the design object and the designer play a crucial role in design, we have to consider another viewpoint, that is, a cognitive viewpoint, which is presented in the next section.

Design Experiment and a Cognitive Design Process Model

In this section, we discuss design processes from an experimental point of view. We used an experimental method called *design experiment* (Yoshikawa, Arai, and Goto 1981; Yoshikawa 1983; Takeda, Tomiyama, and

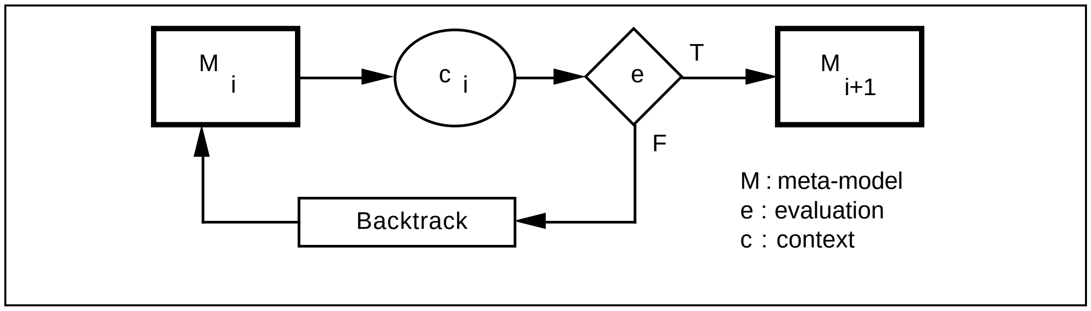


Figure 4. Metamodel Evolution.

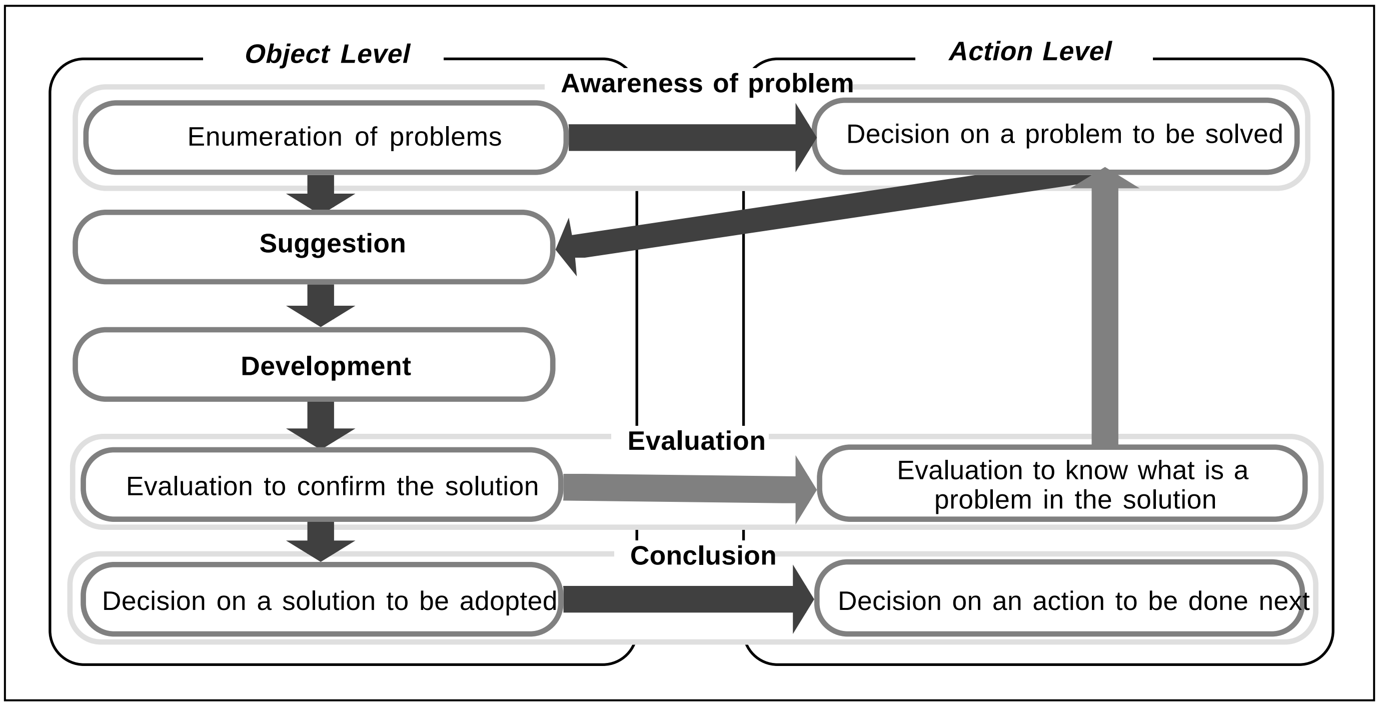

Object Level
Action Level
Awareness of problem
Enumeration of problems
Decision on a problem to be solved
Suggestion
Development
Evaluation
Evaluation to confirm the solution
Evaluation to know what is a problem in the solution
Conclusion
Decision on a solution to be adopted
Decision on an action to be done next

Figure 5. Design Cycle.

Yoshikawa 1990). It is a kind of psychological experiment in which designers are asked to design an artifact from a given set of specifications. The whole designing session is videotaped and analyzed by protocol analysis methods. There are various differences between an experiment in psychology and one in design, not only in task but also in environment, and so on. We discussed how the experiment must be prepared to obtain useful results and performed several experiments based on this discussion (Takeda, Hamada, Tomiyama, and Yoshikawa 1990). This experimental approach is also studied by Ullman, Dietterich, and Stauffer (1988).

One of the major results obtained from the experiments is a cognitive model of design processes when examining a design process from a problem-solving point of view. This model is constructed from unit design cycles.

A *design cycle* consists of five subprocesses: (1) awareness of the problem: to pick up a problem by comparing the object under consideration with the specifications; (2) suggestion: to suggest key concepts needed to solve the problem; (3) development: to construct candidates for the problem from the key concepts using various types of design knowledge (when developing a candidate, if something unsolved is found, it becomes a new problem that should be solved in another design cycle); (4) evaluation: to evaluate candidates in various ways, such as structural computation, simulation of behavior, and cost evaluation (if a problem is found as a result of the evaluation, it becomes a new problem to be solved in another design cycle); and (5) conclusion: to decide which candidate to adopt, modifying the descriptions of the object.

Utterances in the protocol data are categorized into these subprocesses, and then a design cycle is composed of these subprocesses. Basically, a single design cycle solves a single problem, and sometimes new problems that must be solved in other design cycles arise during the suggestion and evaluation subprocesses.

We can distinguish two levels in the design process when we consider the designer's mental activity. One is the object level, where the designer thinks about design objects themselves, for example, what properties the design object has and how it behaves in a certain condition. The other is the action level, where the designer thinks about how to proceed with his(her) design, that is, what s/he should do next. The designer seems to perform his(her) design mutually using these two types of thinking. When looking at the design cycles with respect to this aspect, they also contain these two levels (figure 5).

In this section, we presented the cognitive model of design processes. In the next section, we look at the reasoning aspect of the cognitive and descriptive models, which will result in the computable design process model.

A Logical and Computable Model of the Design Process

Here, we reconstruct the cognitive and descriptive models into a computable model. Logic is a useful framework for this purpose because it is developed from human thought and is closely related to inference. Some results toward understanding design in logic are discussed in Coyne (1988) and Treur (1990), but they are only concerned with design processes at the conceptual level, not at the decision-making or computable levels.

The Logical Foundation for the Computable Model

In logically describing design processes, it is reasonable to assume the following simple model as a first step:

$S \cup K \vdash Ds$,

where *S, K,* and *Ds* are a set of formulas that denote the specifications, knowledge used in design, and the design solutions, respectively. The solutions are derived from the specifications and knowledge as a result of deduction. However, this formalization has three difficulties in interpreting the whole of the design process. First, design is not always performed with complete information, which implies that refinement of specifications and designing of objects are mutually performed in design. We not only use deduction but also abduction (Fann 1970) for the formalization of design processes to refine the specifications. Second, knowledge in the previous assumption is concerned with how to design objects. A typical example is, “If there is a specification S_1, then use a design object D_1.” Although it might be useful for routine design, it is not appropriate for more flexible and creative designs in which knowledge about object properties and behaviors plays an important role. The third problem, which is related to the second, is knowledge incompleteness and inconsistency. When the designed object does not satisfy the specifications, the knowledge base should be regarded as initially incomplete rather than inconsistent.

In this article, we regard design solutions and knowledge as the premise and the specification as the conclusion. Based on this assumption, we formalize design processes as bidirectional and iterative processes as follows:

$$\underleftarrow{\overrightarrow{Ds \cup Ko \vdash P}} \,,$$

where *Ds* is a set of logical formulas describing a design candidate, *Ko* is knowledge of object properties and behavior, and *P* are properties of the design candidate. Required specifications are included in *P*. Given the design knowledge *Ko* and the required properties *P* as the specifications, the designer tries to find a candidate. Deduction is then performed to see (1) what properties the candidate has and (2) whether the candidate contradicts the given constraints, including the specifications. The result is that the description of the obtained candidate becomes complete by detailing. If the candidate does not satisfy the specifications, the designer either tries an alternative candidate or modifies the design knowledge and the specifications. Trying an alternative candidate recalls further abductive or deductive processes, and finally, a complete description of the solution and the specifications are obtained. If there is no other way to explore, the design process terminates.

The previous formalism solves the first and the second problems but not the third one, which is related to the incompleteness of knowledge bases. We introduce circumscription (McCarthy 1980) for this problem. We assume that every piece of knowledge is valid only when it is used in certain situations. However, we can only identify the applicability of knowledge when detecting a contradiction. For a given set of logical formulas, circumscription can be used to compute exceptions that caused the contradiction. By doing so, the original knowledge is modified, and it is able to handle the incompleteness.

The Computable Model

In the previous subsection, we discussed the idea of employing three different types of reasoning, that is, deduction, abduction, and circumscription, for formalizing design processes. In this subsection, we show a computable model of design processes based on these three types of reasoning, and we interpret the cognitive model in the framework of the computable model.

Let us assume that a design process changes its state step by step. In each state, the following formula holds:

$$Ds_c \cup Ko_c \vdash P_c \,,$$

where Ds_c, Ko_c, and P_c are the description of the current design candidate, the knowledge available at the current state, and the properties of the current design candidate, respectively. Now, we can make a computable model by interpreting the cognitive model in the logical framework discussed previously. First, we concentrate on the object level.

During the suggestion subprocess, the designer tries to find a feasible candidate. The purpose of this subprocess is to obtain Ds_c from P_c and Ko_c; it can be regarded as an abduction process. Both the development and evaluation processes are regarded as deduction. In these subprocesses, the designer applies his/her knowledge to the candidates and obtains what is known at the current state. The difference between these two subprocesses lies in the kind of knowledge that is applied. The development subprocess uses knowledge to find out what properties the design object has, whereas the evaluation subprocess uses knowledge to compare those properties obtained in the development subprocess with expectations. The purpose of these two processes is to obtain P from Ds and Ko:

$$Ds_c \cup Ko_c \vdash P' \ .$$

P' can be different from Pc if and only if Ds_c or Ko_c changes. Ds_c changes as the result of abduction, and Ko_c changes as the result of circumscription.

While the designer is developing or evaluating, s/he sometimes encounters a difficulty about the candidate and defines a new problem to solve it. It is a jump from a development or evaluation subprocess to an awareness-of-problem subprocess. We interpret it as circumscription.

The continuation of the design process can be disturbed for two reasons. One is lack of information, which is dealt with by metalevel reasoning. The other is a contradiction in the theory that happens not because the knowledge contains false information but because it is incomplete. We can avoid this contradiction by defining exceptions for these pieces of knowledge using circumscription.

If a contradiction is detected in the theory, we gather formulas that cause the contradiction. We then add literals that are composed of predicates that are abnormal to them and circumscribe these abnormal predicates with the theory. We obtain modified formulas in which their abnormal predicates are substituted by non-empty formulas. As a result, the contradiction is removed from the theory. Because the formulas are modified, we cannot always derive the formulas that represent the required specifications from the current design candidate. Some new formulas must be added to the design candidate to satisfy the specifications. In this process, the contradiction creates a new problem, that is, the awareness-of-process subprocess.

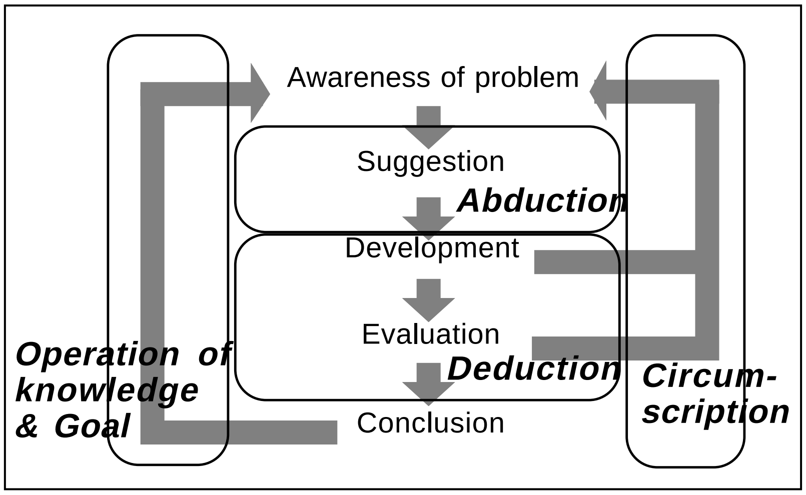


Figure 6. Reasoning on Design Cycle.

To discuss the action level of the logical framework, we introduce metalevel operations to deal with this problem. In our approach, operations on building theories and on how to perform reasoning are introduced as metalevel operations, including setting up of *Ds*; setting up of *P*; setting up of *Ko*; revision of *Ds* by abduction on *P* and *Ko*; revision of *P* by deduction on *Ds* and *Ko*; and revision *Ko* by circumscription on *Ds*, *P*, and *Ko*.

Starting the design cycle is interpreted as executing the setting up of *P* and the setting up of *Ko* operations. Similarly, we can interpret the suggestion, the development, and the evaluation subprocesses as the executions of these operations. Knowledge about how to design can be represented as a sequence of the operations on this level.

Reasoning on the action level is, therefore, to obtain a sequence of the operations on the object level. This reasoning is performed by using the status of the object level and knowledge about how to design. The status of the object level is determined by what kind of information is obtained on the object level and how this information is obtained. Knowledge about how to design is used only on this level and includes procedural knowledge about how to proceed with the design, the design strategy, and so on. It is possible to discuss reasoning on the action level in the same way we did for the object level. Actually, however, only deductive reasoning, such as rule-based reasoning, is needed because most of this type of knowledge is procedural.

We do not interpret the conclusion subprocess with the logical framework. It is a decision-making process that considers all information obtained by other types of rea-

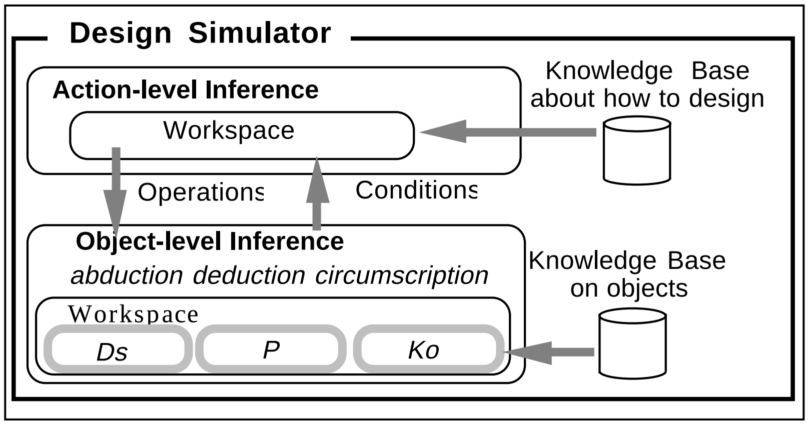
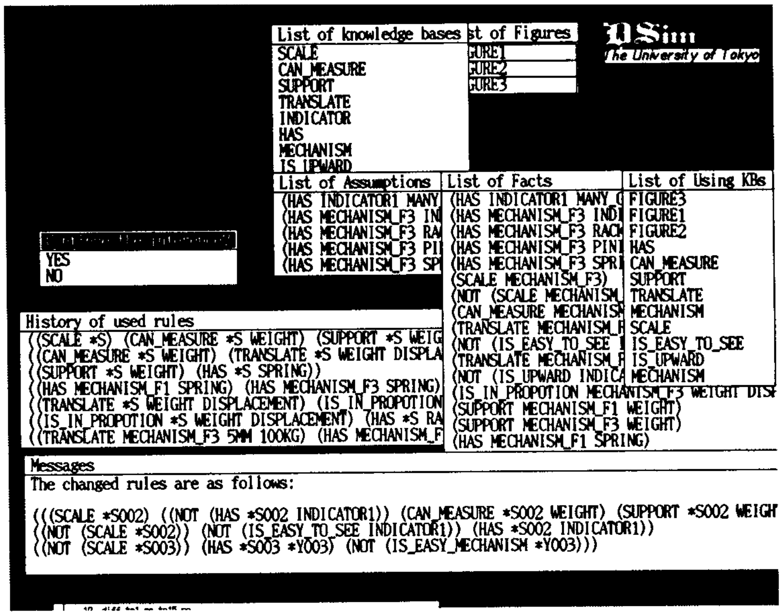


Figure 7. Design Simulator.

soning, and therefore, it seems to be reasoning on a higher level than those types of reasoning. We leave this subprocess unformalized. Figure 6 summarizes the discussion and shows the correspondence between the subprocesses and the types of reasoning in the logical framework.

The Design Simulator

We implemented a prototype of a design simulator that realizes the inference previously discussed. We call it the design simulator because it is designed to track the design processes performed by designers.



```
List of knowledge bases
SCALE
CAN_MEASURE
SUPPORT
TRANSLATE
INDICATOR
HAS
MECHANISM
IS_UPWARD

st of Figures
GURE1
GURE2
GURE3

List of Assumptions
(HAS INDICATOR1 MANY
(HAS MECHANISM_F3 IN
(HAS MECHANISM_F3 RA
(HAS MECHANISM_F3 PI
(HAS MECHANISM_F3 SP

List of Facts
(HAS INDICATOR1 MANY_(
(HAS MECHANISM_F3 INDI
(HAS MECHANISM_F3 RACK
(HAS MECHANISM_F3 PINI
(HAS MECHANISM_F3 SPRI
(SCALE MECHANISM_F3)
(NOT (SCALE MECHANISM_
(CAN_MEASURE MECHANISM
(TRANSLATE MECHANISM_F
(NOT (IS_EASY_TO_SEE I
(TRANSLATE MECHANISM_F
(NOT (IS_UPWARD INDICA
(IS_IN_PROPOTION MECHANISM_F3 WEIGHT DIS
(SUPPORT MECHANISM_F1 WEIGHT)
(SUPPORT MECHANISM_F3 WEIGHT)
(HAS MECHANISM_F1 SPRING)

List of Using KBs
FIGURE3
FIGURE1
FIGURE2
HAS
CAN_MEASURE
SUPPORT
TRANSLATE
MECHANISM
SCALE
IS_EASY_TO_SEE
IS_UPWARD
MECHANISM

YES
NO

History of used rules
((SCALE *S) (CAN_MEASURE *S WEIGHT) (SUPPORT *S WEIG
((CAN_MEASURE *S WEIGHT) (TRANSLATE *S WEIGHT DISPLA
((SUPPORT *S WEIGHT) (HAS *S SPRING))
((HAS MECHANISM_F1 SPRING) (HAS MECHANISM_F3 SPRING)
((TRANSLATE *S WEIGHT DISPLACEMENT) (IS_IN_PROPOTION
((IS_IN_PROPOTION *S WEIGHT DISPLACEMENT) (HAS *S RA
((TRANSLATE MECHANISM_F3 5MM 100KG) (HAS MECHANISM_F

Messages
The changed rules are as follows:

(((SCALE *S002) ((NOT (HAS *S002 INDICATOR1)) (CAN_MEASURE *S002 WEIGHT) (SUPPORT *S002 WEIGH
((NOT (SCALE *S002)) (NOT (IS_EASY_TO_SEE INDICATOR1)) (HAS *S002 INDICATOR1))
((NOT (SCALE *S003)) (HAS *S003 *Y003) (NOT (IS_EASY_MECHANISM *Y003)))
```

Figure 8. A Snapshot of the Design Simulator.

The Architecture

The design simulator consists of two main parts: the action-level inference system and the object-level inference system. The object-level inference consists of the workspaces *Ds*, *P*, and *Ko* and three inference subsystems, that is, deduction, abduction, and circumscription (figure 7).

The metalevel inference is performed by a rule-based system. Knowledge used on this level is about how to design, for example, knowledge about selecting a knowledge base and scheduling reasoning according to the condition of the object level. The result of this deduction is a sequence of operations on the object level.

The object-level inference is performed by abduction, deduction, and circumscription to the workspace, which modify the current state of *Ds, P,* and *Ko*. *Ko* contains knowledge as a horn clause, and *P* and *Ds* contain objects and their properties as atom formulas that only have a literal. The inference on the object level modifies the contents of *Ds, P,* or *Ko* and sometimes causes contradictions, which are reported to the metalevel as a condition of the object level.

The abduction system is implemented based on the algorithm proposed by Poole (1988). The circumscription system is implemented using the algorithm proposed by Nakagawa and Mori (1987) for computing circumscription (Lifschitz 1985) on clausal forms.

The design simulator simulates design cycles by repeatedly applying abduction, deduction, and circumscription. Their order is specified by the action level, which asks the users to choose when there are some alternatives and when it encounters a conclusion subprocess. This system is implemented in Allegro Common Lisp and X11 on a Sun-4.

Example

We simulated a design process composed of protocol data obtained by a design experiment. Figure 8 shows a snapshot of the design simulator solving this problem. The task was to design a weighing scale; a part of the protocol data is shown in figure 9. This example demonstrates that the design simulator is capable of replaying design sessions by tracking protocol data obtained by design experiments. The performance of the design simulator indicates that the computable model presented in the previous section shows the same operational aspects as the cognitive model. It also justifies that it is crucial to employ different types of reasoning for implementing design subprocesses of the cognitive model as a computable model.

(1) What mechanism does a standard scale use?
(2) It measures the weight like this (Figure A).
...
(3) If we can use a rack and pinion (Figure B), we can measure the weight because the displacement is in proportion to the weight.
(4) Anyway, we think the indicator first.
(5) As it translates 5mm of the displacement
to the 100kg weight, the displacement per 1kg is 0.05mm.
(6) It is impossible to realize it with Figure B.
(7) If we don't mind the accuracy, it is possible by using many gears.
...
(8) But a standard scale must use a more simple mechanism.
...

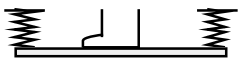


Figure A

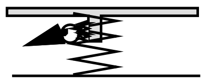


Figure B

Figure 9. Examples of the Protocol Data.

Conclusions

In this article, we presented three different models: a descriptive model, a cognitive model, and a computable model. In the descriptive design process model, we presented a general framework of the design process as an evolutionary refinement process of the metamodel. Because we put emphasis on the representation of the design object and the design process rather than on reasoning aspects in the discussion of the descriptive model, the mechanism of reasoning in design did not become clear. These aspects are highlighted for the discussion on the cognitive design process model. We examined the design processes of designers at work and presented how the designer proceeds with his(her) design.

We discussed the relationships among the descriptive, the cognitive, and the computable models. The computable model uses the framework of the descriptive model, and the reasoning in the computable model is an interpretation of the cognitive model. A unit design cycle in the cognitive model corresponds to a refinement step in the descriptive model. The framework of the computable model is the evolutionary process of the design object discussed in the descriptive model; that is, we realized it as a revision process of the theory in logic. This revision process of the theory is performed by three different types of logical reasoning: abduction, deduction, and circumscription. Thus, we succeeded in integrating the descriptive and cognitive models using a logical framework on which the computable model is based. Furthermore, this computable model resulted in a system called the design simulator.

Acknowledgments

We would like to thank Paul ten Hagen for various discussions on design process models, Kazuto Hayashi for useful discussions on logical formalization, and Juro Baba for contributions to the implementation of the design simulator.

References

Coyne, R. 1988. *Logic Models of Design.* London: Pitman.

Fann, K. T. 1970. *Peirce's Theory of Abduction.* The Hague, The Netherlands: Martinus Nijhoff.

Finger, S., and Dixon, J. R. 1989. A Review of Research in Mechanical Engineering Design. Part I: Descriptive, Prescriptive, and Computer-Based Models of Design Processes. *Research in Engineering Design* 1(1): 51–67.

Lifschitz, V. 1985. Computing Circumscription. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 121–127. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

McCarthy, J. 1980. Circumscription—A Form of Non-Monotonic Reasoning. *Artificial Intelligence* 13:27-39.

Nakagawa, H., and Mori, T. 1987. Computable Circumscription in Logic Programming (in Japanese). *Transactions of Information Processing Society of Japan* 28(4): 330–338.

Poole, D. 1988. A Logical Framework for Default Reasoning. *Artificial Intelligence* 36:27–47.

Takeda, H.; Tomiyama, T.; and Yoshikawa, H. 1990. A Logical Formalization of Design Processes. In *Intelligent CAD, II,* eds. H. Yoshikawa, and D. Gossard. Amsterdam: North-Holland. Forthcoming.

Takeda, H.; Hamada, S.; Tomiyama, T.; and Yoshikawa, H. 1990. A Cognitive Approach of the Analysis of Design Processes. In Proceedings of the Second ASME Design Theory and Methodology Conference. New York: The American Society of Mechanical Engineers. Forthcoming.

ten Hagen, P. J. W., and Tomiyama, T., eds. 1987. *Intelligent CAD Systems, I: Theoretical and Methodological Aspects.* Berlin: Springer-Verlag.

Tomiyama, T., and Yoshikawa, H. 1987. Extended General Design Theory. In *Design Theory for CAD—Proceedings of the IFIP Working Group 5.2 Working Conference,* eds. H. Yoshikawa and E. A. Warman, 95–124. Amsterdam: North-Holland.

Treur, J. 1990. A Logical Framework for Design Processes. In *Intelligent CAD Systems, III: Practical Experience and Evaluation,* eds. P. J. W. ten Hagen and P. J. Veerkamp. Berlin: Springer-Verlag. Forthcoming.

Ullman, D. G.; Dietterich, T. G.; and Stauffer, L. A. 1988. A Model of the Mechanical Design Process Based on Empirical Data: A Summary. In *Artificial Intelligence in Engineering Design,* ed. J. S. Gero, 193–215. Amsterdam: Elsevier.

Veerkamp, P. J.; Pieters Kwiers, R. S. S.; and ten Hagen, P. J. W. 1990. Design Process Representation in ADDL. In *Intelligent CAD Systems, III: Practical Experience and Evaluation,* eds. P. J. W. ten Hagen and P. J. Veerkamp. Berlin: Springer-Verlag. Forthcoming.

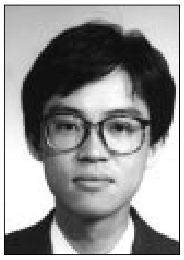
Veerkamp, P. J.; Kiriyama, T.; Xue, D.; and Tomiyama, T. 1990. Representation and Implementation of Design Knowledge for Intelligent CAD—Theoretical Aspects. In Proceedings of the Fourth Eurographics Workshop on Intelligent CAD—The Added Value of Intelligence, 184–205. Compiegne: Springer-Verlag.

Veth, B. 1987. An Integrated Data Description Language for Coding Design Knowledge. In *Intelligent CAD Systems, I: Theoretical and Methodological Aspects,* eds. P. J. W. ten Hagen and T. Tomiyama, 295–313. Berlin: Springer-Verlag.

Yoshikawa, H. 1983. CAD Framework Guided by General Design Theory. In *CAD Systems Framework—Proceedings of IFIP Working Group 5.2 Working Conference*, eds. K. Bø and E. M. Lillehagen, 241–253. Amsterdam: North-Holland.

Yoshikawa, H. 1981. General Design Theory and a CAD System. In *Man-Machine Communication in CAD/CAM—Proceedings of the IFIP Working Group 5.2–5.3 Working Conference 1980 (Tokyo),* eds. T. Sata and E. A. Warman, 35–53. Amsterdam: North-Holland.

Yoshikawa, H.; Arai, E.; and Goto, T. 1981. Experimental Design Theory (in Japanese). *Journal of the Japan Society of Precision Engineering* 47(7): 830–835.



Hideaki Takeda is a Ph.D. student in the Department of Precision Machinery Engineering, the University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo, Japan. His research interest is in a theory of intelligent computer-aided design, in particular, logic formalization of design and experimental study of design. He will obtain his Ph.D. in precision machinery engineering in March 1991.

Paul Veerkamp is a research associate at the Center for Mathematics and Computer Science in Amsterdam. He is currently a guest researcher at the University of Tokyo. His research focuses on a knowledge representation language for intelligent computer-aided design. He will obtain his Ph.D. in computer science at the Free University of Amsterdam in early 1991.

Tetsuo Tomiyama has been an associate professor in the Department of Precision Machinery Engineering at the University of Tokyo since 1987. From 1985 to 1987, he worked at the Center for Mathematics and Computer Science in Amsterdam, where he served as project leader for the Intelligent Integrated Interactive CAD Systems Project. He received his Ph.D. in precision machinery engineering from the University of Tokyo in 1985. His research interests include intelligent computer-aided design (CAD); applications of knowledge engineering to mechanical engineering, in particular, the use of qualitative physics; and further development of the theory of CAD.

Hiroyuki Yoshikawa has been a professor in the Department of Precision Machinery Engineering at the University of Tokyo since 1978 and the dean of the engineering faculty since 1989. His primary interests are in the theory of design, intelligent computer-aided design, the theory of reliability and maintenance, and maintenance robots. Yoshikawa is a member of the International Federation for Information Processing and the International Institution of Production Engineering Research and vice president of the Japan Society of Precision Engineering.