New Trends in Software Methodologies, Tools and Techniques
H. Fujita and M. Mejri (Eds.)
IOS Press, 2006

A New Way of Automatic Design of Software (Simulating Human Intentional Activity)

Zenya Koono [a1], Hassan Abolhassani [b] and Hui Chen [c]
[a] *Creation Project, Kanagawa, Japan*
[b] *Computer Engineering Dept., Sharif University of Technology, Teheran, Iran*
[c] *Information Science Center, Kokushikann University, Tokyo, Japan*

Abstract. This paper reports on a new method of automatic design. It is a simulation of "human intentional activity," appearing in management, business, various designs and physical work. It is achieved by repetitive hierarchical decomposition of human concepts using (mainly) natural language expressions. Three typical ways (skill, rule and knowledge based) to create the hierarchical decomposition have been implemented and evaluated quantitatively as a CASE tool. The rule-level operation achieves nearly 100% automatic detailing.

Key Words. Design, Automatic Design, Human Intentional Activity, Hierarchical Detailing, Learning Effect, Zipf's Principle, Skill-Rule-Knowledge.

Introduction

This paper introduces an entirely new way of automatic software design. It is a simulation of a human design that is quite different from conventional studies. There have been being two trends of automatic software design. One is the Software Engineering approach, and another is the Artificial Intelligence approach. In the former, an automatic design is made using input-to-output relationships. This features a simple and highly practical mechanism, such as for data processing, but lacks general applicability. In the latter, a process, controlled by more abstract or so-called "knowledge," performs an automatic design. This may have a more universal power, but the designed results seem to be different from what humans design.

The authors have studied the third method, namely an automatic design in the same manner as a human being designs [2, 5, 6, 8, 14, 16]. In Section 1, a human design is explained; in Section 2 the automatic method to reproduce the aforementioned human design is explained. Section 3 is the evaluation and Section 4 discusses impacts on software engineering

One of authors, Koono, has a wealth of technical experience as an engineer, manager and administrator in the fields of components, hardware, software and systems. From this experience, he became aware that throughout these fields there is a "commonality

[1] Corresponding Author: Creation Project Kanagawa, Japan; E-mail: koono@vesta.ocn.ne.jp.

of human knowledge." From 1991 to 2001 at Saitama University, he conducted the Software Creation Project, for automatic software design. This paper is the result of these studies.

Some specialties of the project are as follows. Software design methodologies began from "step by step detailing" and advanced to various "structured designs." Their key points are a hierarchical decomposition of a human concept. The hierarchical decomposition is not only common to management, business, various designs and physical actions but also any kinds of "human intentional activity." This study uses the hierarchical manner of human concept decomposition.

Natural language (including technical terms) plays a substantial roll in design for enabling the hierarchical decomposition of a human concept. In order to indicate various mutual relationships between concepts rigorously and correctly (as is done in hardware), various figures, charts and diagrams have been used extensively.

A hierarchical decomposition of a parent concept, to the children concepts, is a kind of problem solution. Zipf, in Ergonomics, proved the famous "principle of least effort" [3] in human work at problem solving. This says that, "when a person is given a problem, they try to solve it in the simplest way." If they are unsuccessful, they try a slightly more advanced way, and this continues in stages until the problem is solved [3]. This shows that the human brain consists of a number of engines, and through these it achieves high efficiency. Rasmussen pointed out that among those engines there are the following three typical ways [4]:

Skill-based: speedy almost reflective work, the most frequent.

Rule-based: following patterns and associated rules.

Knowledge-based: activity based on abstract knowledge. It is used infrequently.

Studies have been made taking the three modes into account, and go from skill-based, to rule-based and finally to knowledge-based. As the patent descriptions[6, 14] explain the technical details, this paper puts emphasis on the philosophy and the basis.

1. Principles of Human Design

Figure1 [10, 18] shows the design record of a "clock" program. It is an example of the combined use of graphic and natural language. The design progresses from the left side data flow, to the flowchart, and finally on the rightmost side is the source code.

The rectangular box with a semi-circle at the top and bottom shows the data, and a square box shows the functions. The data flow begins with data, and repeating function and data, ends with data. The topmost section of the data flow on the left edge is the specification, "Clock." By adding data at both sides, the second section is an elementary data flow of the "clock."

It is stretched and intersected by two MAP's (Most Abstracted Point) "Time" and "Hands", following Myers' STS division [11]. It is detailed to form a hierarchically detailed data flow below, which starts from "1 Sec clock" and ends at "Clock surface" in a serial manner. A parent concept, represented by an elementary data flow "Clock" is hierarchically decomposed to three children concepts, represented by three elementary data flows, "Obtain time," "Obtain hands," and "Display." Here, a flowchart starts from a compressed barrel symbol on "Obtain time." It is also shown in the center of Figure 1. The source code corresponding to it is shown on the right side of Figure 1. These are specialties inherent to software, where a control flow is necessitated and computer

language is used for representation.

Next, decompositions are made from "Obtain time," "Obtain hands" and "Display." Among them, that for "Obtain hands" is shown in the next level data flow. Here, both data "Time" and "Hands," are hierarchically decomposed to lower level hierarchical data, resulting in three data flows in a parallel manner. It is a typical pattern of Jackson's Program Design [12].

The next stage shows a hierarchical decomposition of "Obtain minute hand." The first function box "Obtain angle of minute hand" may be achieved by 6 times (minute value), and the code, on the right side list, shows this.

> *Thus design may be defined as follows: It is repetitive concept decompositions. As it is repeated, a concept becomes more solid; more detailed and is finally reduced to minute details ready to be expressed in the means (computer language statement etc.) to implement it [1, 2].*

In this paper, the pair is called a "design rule" and used for design. If a hierarchical decomposition is made with a large decomposition distance, the number of the children concepts (e.g. boxes of the data flow) becomes larger. If the progress is made to be small enough, the children becomes simpler and the average expansion rate of function is a little smaller than 3[23]. (In Figure 1, they are all 3.) As it stems from a human characteristic, it appears all every human conduct.

Such hierarchical decomposition of a concept appears in every human intentional activity. In Military Science "hierarchy of object" is regarded as a fundamental principle in the planning of war [13]. A supreme commander receives the "final objective" (e.g. to occupy X island). This person develops the objective into several means to achieve it, and assigns each to a subordinate (e.g. Navy: approach Y miles to X island. Air Force: strike defenses. Army: land soldiers and bring the island under control). The means for the commander then become the objectives of subordinates: the subordinates repeat the same process. Not only the authors but also many others say that the same applies in management and systems design.

Next is an example from human physical work. "Take a picture" may be decomposed into actions like "direct a camera to the target," "focus on the target" and "press the shutter button."

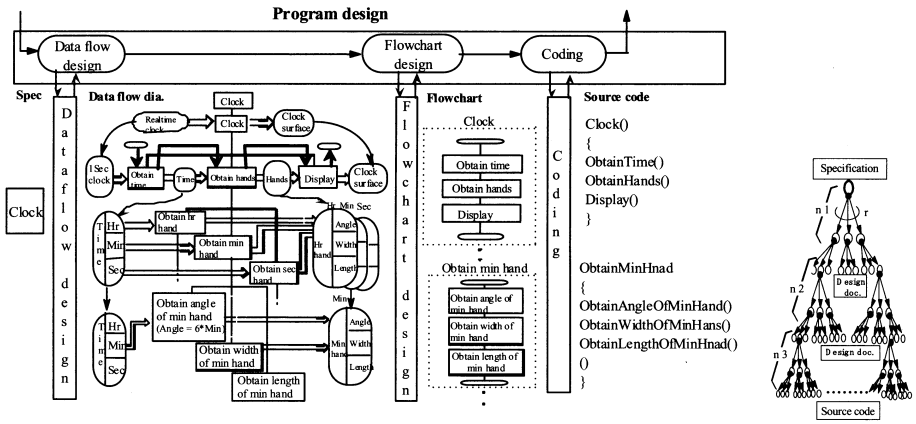

Figure 1. Design records of clock program

In all of them, irrespective of physical activity (e.g. "Take a picture") or mental operation (e.g. designs and planning of war), a hierarchical decomposition performs detailing (sometimes with conversion). It is caused by the fact that a concept from the outside world is projected into one's brain, and is expressed by a symbol there, and the hierarchical decomposition is made using these. In this paper, all these human intentional activities are called "design."

In the rightmost side of Figure 1, a hierarchically fixed-rate expanding network model is shown. A white circle is the concept, while a black dot is a process for a hierarchical decomposition. The design is repeated hierarchical decompositions. From the top, it goes along the network like the single strokes of a brush on paper.

Design by a human being is to detail a parent concept by hierarchical decomposition. In order to make operations simple and without error, various provisions are made:

*Small steps of detailing with accurate statements.

*Visual presentations using graphics.

*Relationship line from a parent function to the children as well as from a parent data to the children.

A step of design, namely a hierarchical decomposition, is a human problem solving process. From Rasmussen's viewpoint, there are "skill-based," "rule-based" and "knowledge-based" solutions. The following definitions are made in the project:

Skill-based: A full set of design rules is used.

Rule-based: Detailed children are created following certain rules.

Knowledge-based: A solution when the preceding two is ineffective.

2. Automatic Design System

2.1 Mechanization of design

In this study, the automatic design system designs just as a human designer does [8], in other words, the system designs by the same graphic operations as shown in Figure 1. In it, a human design is attained by graphic operations of data flow (DFD) and flowchart. A structured chart PAD (Problem Analysis Diagram) is used instead of a flowchart. Figure 2 shows this. It is a hierarchical chart with symbols of "Function," "Decision" and "Repetition." The control starts from the top left edge and goes down, while decomposition progresses toward the right. The "Functions," denoted in three rectangular boxes, activate sequentially. The middle function ("Parent") is hierarchically decomposed to four child functions (Children 1 – 4) on the right side. The wedge symbol, Child 3, is a "Decision." If the value in the symbol is "true," the upper right side function Child 3a is activated; if it is "false," the lower right side the function Child 3b activated. The Child 2 function is hierarchically decomposed to three functions displayed on the right hand side. In the middle left a symbol with a vertical line is a "Repetition." While the

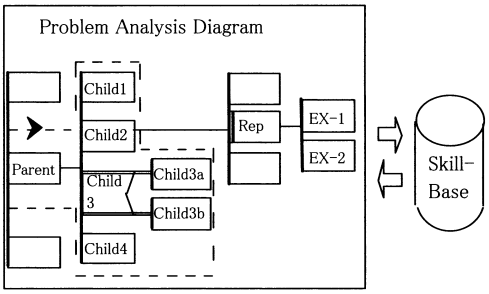

Figure 2. Problem Analysis Diagram

conditions in the symbol are "true", it repeats the two functions EX1 then EX2 extending on to the right hand side.

A design rule may be acquired automatically from a PAD 14, 5]. Let's assume that "Parent" symbol is marked with the "starting point mark", and "acquisition" is commanded. The tree walk program starts from "Parent," rotates around "Parent" symbol and the all the children symbols, and returns to the starting point. During these, it acquires statements from each symbol. The "starting point mark" advances to the next symbol. The information thus acquired is transformed into a design rule.

In order to make an automatic design [14, 5], the designer draws a "Parent" on the CASE tool screen, puts a "starting point mark" on it, and commands to start. In PAD, the tree-walk program acquires "parent information" from "Parent," and it is sent to the engine (e.g. Skill-Base in the right side of the figure.). If there is a corresponding design rule there, it is sent back to PAD. This is pasted on the right side of the "Parent" symbol, and the "starting point mark" moves to the next symbol.

In the case of DFD, the situation is almost the same. Let's take "Clock" in Figure 1 as an example. The design rule is a combination of the elementary data flow of "Clock" and the detailed data flow of the children, which starts from "1Sec clock" and ends in the "Clock surface." The starting point mark is set on "Clock." Let's assume "acquisition" is commanded. The tree walk program turns around the parent elementary data flow. Then it moves to the children data flow via the data relationship arrow line, rotates around it and returns to "Clock" via the data relationship line. The information acquired during the round trip is purified and a DFD design rule and PAD design rule may be obtained. (If the PAD CASE tool works similarly, the acquired design rule may be checked to match it.) They include hierarchical relationship lines connecting function symbols. The starting point mark advances to the next line. In the high level design, DFD may be used alone, and in the end near coding, PAD may be used alone. In between these two extremes, both may be used.

2.2 Inside of a design rule

After the development of the Intelligent CASE tool (ICASE) working in skill-based mode with PAD in 1998, studies toward the next step had been started. The key was thought to be inside of a design rule. The approach was to prepare correct and very good examples, and to extract some of the information from "inside" a design rule. The authors' approach was a repetitive trial and error, seeking another way apart from Artificial Intelligence. The Basic Concept Dictionary (BCD) for elementary words was assumed as a tool.

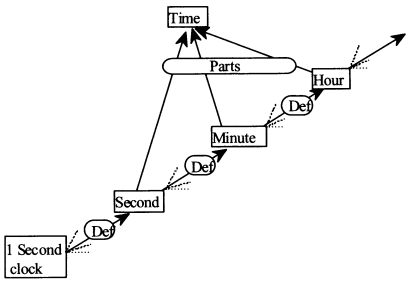

Figure 3. Micro design rules

Figure 3[7, 15] is the mechanism for finding the input side MAP (e.g. "Time" in Figure 1) of STS division. Input (output) side MAP is the most distant concept from the input (output), still keeping an input (output) character. Also, a MAP must be of high abstraction level. The leftmost "1 Sec clock" is the input, and the final output is assumed to be the far point on the right.

In the figure, "Time" on the top is the MAP. A chain is created from the input toward the

output, and nodes in the chain are connected to "Time." Starting from "1 Sec clock", the chain consisted of "Second", "Minute" and "Hour." It was formed by a "micro design rule" for "Building Concept Definition Chain." The next "micro design rule" for "Building Concept Abstraction" finds out "Time" from nodes in the chain. The functions of these micro design rules should be obvious to the readers.

There are other "micro design rules:" "Extending Lower Level Concepts," "Creating Function Name," "Upwarding Correspondance to Upper Level Concept," "Creating Control Structure" and "Selecting I-MAP and O-MAP". Some of them are elemental and others are combinations. Above them, there are macros programs for solving a problem by "micro design rules". Although the inside had been clarified to some extent, the authors' trials extending these methods failed due to the fact that the authors' technology level had not yet advanced far enough to be able to use them.

One alternative to Figure 3 is as follows: Give up the idea of creating MAP's but reuse past experience. When a human designer creates a MAP, connect it to the input or output concerned by an auxiliary relationship line, and present them upon request. Thus specified MAP's may be verified by the logic in Figure 3 with BCD. From these and others, the authors believe that the so-called knowledge problem should be considered in the last phase as Zipf pointed out.

2.3 Abstraction from good examples

Studies for rule-based cases were made on excellent design examples [15, 16]. The material was an elementary "inventory control system," chosen by following considerations of (1.) a small but consistent system, and (2.) a business system, in which the rules are obvious.

Seven programs (P1 - P7) were developed in the same manner as Figure 1, using data flows and companion PAD's with a Japanese language statement [7, 16]. Figure 4 shows a stage of the design. The progress of a design is chosen to be the minimum required. Data flows are shown on the left side, where a lozenge is data and a square box is a function. PAD's are shown on the right side. The original sentences are written in Japanese to indicate operations and data clearly and correctly. After design, repeated reviews and rigorous checks were made. From Figure 4, it will be understood that each detailing is very simple and *data flows are in a very simple form*. These are the results, the progress achieved in small steps.

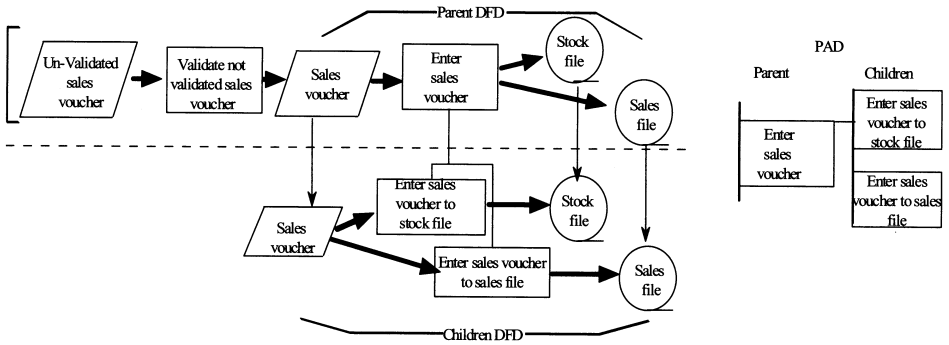

Figure 4. Samples of DFD and PAD

Please read aloud the detailed data flow in Figure 1, from "1 Sec clock" and ending with "Clock surface". Although the sentences are much different from usual natural language sentences, still one can understand what each symbol means clearly. It will also be noticed that they are written in a kind of restricted grammar. This may *be realized through simple logic.*

From these, it was concluded that the best method is a frame type memory system. In it, skeleton DFD and PAD are the *data* of a frame, while the modification logic to complete the statements in each symbol are the *methods* of the frame. This set is needed for each meaning of a verb. They were called the Verb Dictionary. It may be accessed by the verb in the function of the parent elementary data flow. The access causes a read out of several pages (of different meaning) of the verb in the Verb Dictionary.

There has been another problem. A verb, constituting the center of a function, has several meanings, and the detailing must be made based on the meaning of the verb used in the environment. In the ICASE tool developed in 1998, the designer understands the meaning of a verb, and selected the meaning out of the candidates.

The required function has been named the Design Direction Finder.

On closer examination of the design results, it was found that the pair of the verb in the function and the output data in the parent elementary data flow, fixes the meaning of the verb. By using this relationship, it becomes possible to design the Design Direction Finder. From these findings, extracted from good design examples, the architecture of the next Integrated Intelligent CASE (IICASE) has been determined.

2.4 Automatic design system

Figure 5[6, 16] shows the structure of the IICASE tool system [25, 26, 27]. On the left side of Figure 5, there is a DFD CASE tool and a PAD CASE tool working synchronously. The Structure Dictionary Diagram (STD) Case tool is provided to define the hierarchical structure of data (e.g. "time" and "Second", "Minute" and "Hour") graphically. From the hierarchical data structure diagram on the display screen, the tree walk program extracts the hierarchical data definition for programs.

In the center, along with the Integration unit, there are others for control and communication. On the right side, there are various engines that receive parent information and send back corresponding (DFD and PAD) design rules. The Skill-based engine is for the design rules in the Skill-based mode for DFD and PAD. The Creator CASE tool is a platform used for the creation of a design rule, and Creator 00 program is the rule-based engine for creating a design rule. There are various modes for the system operation:

- DFD mode, combined DFD & PAD mode and PAD mode etc.
- Single engine mode, multiple engine modes
- Match check option etc.

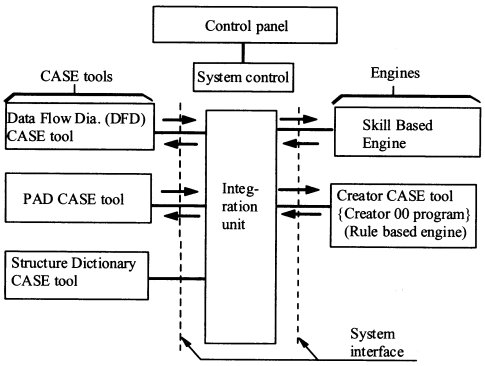

Figure 5. Integrated Intelligent CASE tool

2.5 Principle of IICASE operation

As principles of skill-based operations have been explained earlier, this description is omitted and the rule-based operations are described. A parent concept, the parent elementary data flow, is sent from DFD CASE tool to Creator 00 program. It is transferred to a register, shown at the top of the Design Direction Finder in Figure 6. The verb part is extracted from the function, and it starts access to the Verb Dictionary resulting in several read-out pages of the Verb Dictionary.

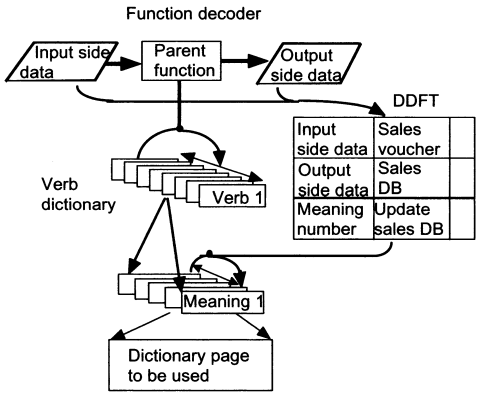

Figure 6. Design direction finder

The Design Direction Finder Table (DDFT) has a table structure shown in the figure. The lines consist of the input side data, the output side data and the meaning of verb information. (Unnecessary data may be omitted by special mark in the data) Each column of DDFT corresponds to a meaning of this verb. Together with Verb dictionary pages, the DDFT for the verb is also read out. For example, the first column data are as follows:

Input data: 'Sales voucher'
Output data: 'Sales DB'
Meaning: 'Update of Sales DB

The initial two lines of each column of DDFT are compared with the pair of the input and output data shown on the top of figure 6. When input and output data matches respective lines of the column, the third line of the column indicates the meaning of the verb page.

Figure 7 shows the structure of the Verb Dictionary page. It consists of

•data area for skeleton (for DFD and PAD) and •method area.

A designer can define any skeleton on Creator CASE tool graphically, and it is converted to a Verb Dictionary page.

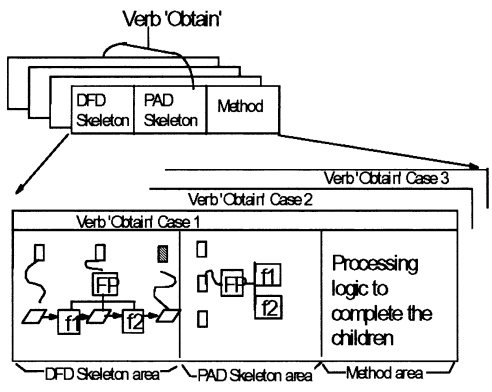

Figure 7. Verb dictionary

*As a small step detailing is made, the number of children symbol is small. (e.g. the average expansion rate of function is a little smaller than 3.)

*As the statement in each box may be expressed in a kind of restricted grammar, necessary variations of the method are a few.

*As the treatments of data are limited, an API is provided to describe the movement of data easily.

*Except for data and function, false parent, function and data relationship lines are provided for making pasting of the children easy.

The method inserts, changes or modifies statements in each box to complete the detailed data flow and PAD. Thus completed detailed children DFD (PAD) are sent back to DFD (PAD) CASE tool, and it is pasted to the bottom of the parent DFD and to the right of the parent PAD. (In most cases, the children concept of PAD may be obtained from the children DFD, and thus the PAD skeleton is not used. On the contrary, if PAD is "decision" or "Repetition" a PAD skeleton is used and DFD operations are not made here.

There is also another type of common rule. It is the Jackson Program Design pattern as shown with "Time" and "Hands" in Figure 1. In this case, both input data and output data are hierarchically decomposed. It is following a 3-step procedure to prepare the children concepts.

1. Input (e.g. time) and output (e.g. hands) data are hierarchically developed by using a Structure Dictionary.
2. From the input and output children data developed, a corresponding pair of input data and the matching output data (e.g. "minute" and "minute hand") are found, and the pair forms each DFD with a blank function.
3. A function phrase, inheriting the verb in the parent function, is inserted in each function box.

3. Evaluation

3.1 Experimental systems

The ICASE tool reproduced human design at skill-based in 1998[5]. It consists of an off-the-shelf PAD CASE tool as a man-machine interface, being modified somewhat, and the intelligent control, knowledge base as the skill base, and related controls. Programs were written using C language, and they operated on Windows 98/NT. For various controls, Finite State Machine control (FSM control [24]) was applied and Midas OS[24] was used for the event driven OS. (FSM control is an advanced version of technology using ITU SDL, a simplified version of which is known as UML 2.0.)

The IICASE tool reproduced human design in both skill-based and rule-based in 2001. In order to develop various CASE tools efficiently, extensive reuse was made. They were

- the standardized graphic sub-system developed using VISIO, and
- the standardized data structure and control structure using FSM control.

The total software size newly designed for CASE tools was around 5 K lines of C code, excluding Midas OS [25, 26, 27]. Creator 00 program, for rule-based engine and the rule base, was around 2.8 K lines of C code [6]. This system works on Windows 98/NT.

3.2 Skill-based operation

The evaluations made in the ICASE [5] are explained. Essentially they are also applicable to others, excluding the learning rate, which is dependent on the environment and the application. At the skill-based operation, the automatic acquisition of this system enables it to accumulate newly experienced design rules as knowledge. It works like an apprentice imitating designs. For the evaluation, so-called maintenance designs were chosen. A simple telephone switching system was used as a model, on

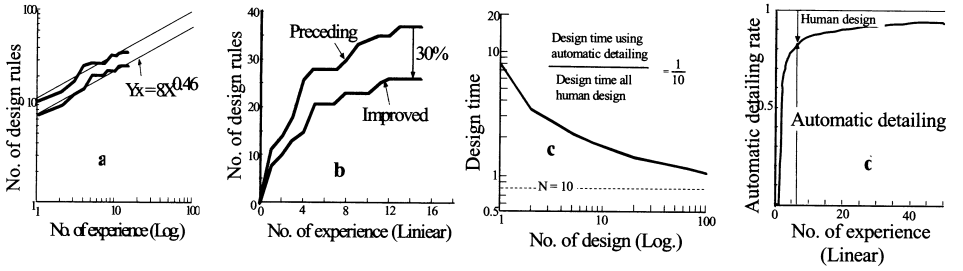

Figure 8. Characteristics of skill-based system

which service features were repeatedly added, and the subscriber's data access programs were also changed. These changes were used for the evaluation.

Figure 8 (a)[5] and (b)[5] show the accumulated number of new designs rules vs. the accumulated number of designs. In Figure 8.a, on the linear scale, the curve grows rapidly at first, and then the gradients gradually decrease. This is called a Learning Effect, and appears when a person is learning a game or a sport. In Figure 8.b in both logarithmic scales, the plots show a linear trend line. This type of a Learning Effect is called a "Logarithmic Learning Effect," and appears in the most characteristics of human conduct. Since this was a very important result, another trial was held to reconfirm the result. The resulting two curves show almost the same gradient. Thus, the results were accepted.

Using the Learning Effect technology in Industrial Engineering [17, 22, 23], quantitative evaluations of this skill-based case were made. Figure 8.c [5] shows the productivity (man-hours for one hierarchical decomposition) as designs were experienced. The curve shows the productivity increases as experiences are accumulated, and the curve shows a logarithmic learning effect. Figure 8.d [5] shows the percentage of automatic detailing as the designs are repeated. Although it grows rapidly at first, the increase over 80% is very slow, and it does not tend to reach 100%. It is a reflection of the shallow nature of the knowledge. It is quite similar to a human apprentice designer.

This skill-based system becomes a very cost effective and a good training tool for learning from past designs. Although the original system needed designers' intervention for the selection of the meaning of a verb, the DFD and PAD combined system can automate the operation. The simplicity of the system and the high efficiency seem to come about when "designs are replaced by graphic operations."

These prove that a learning effect such as increased productivity arises from the accumulation of the "knowledge," and it was found that the logarithmic improvement comes from the logarithmic nature of the accumulation of the "knowledge."

3.3 Rule-based operation

The evaluation of a rule-based engine was made on the aforementioned elementary inventory system [16]. The initial system consisted of seven programs (P1 to P7) and the total size is around 700 lines of C code. During the evaluation, in order to simulate so-called maintenance, additional seven programs (P8 to P14) were designed and used.

The average number of skeletons in the Verb Dictionary page was 4.6 symbols/verb (mostly for DFD). The average number of lines of method/verb was 10.1 lines. The

total software size needed for the rule-base is very small.

Figure 9 [16] shows the accumulated number of Verb Dictionary pages as designs of programs were made. The horizontal axis is the accumulated number of hierarchical detailing from the first to the seventh experience, and the additional seven programs were added. Several sequences of the initial seven programs were tried, and the curves are plotted on the figure. All curves show a similar trend.

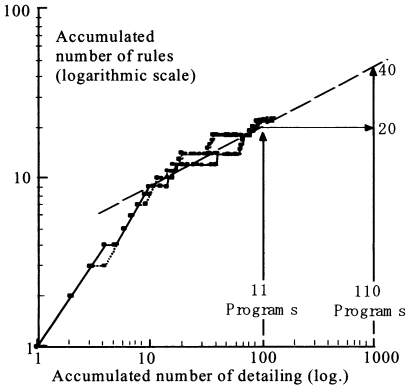

Figure 9. Learning curves of rule-based case

The curve breaks around the 10th hierarchical decompositions. Until this point the gradient is large, and the number of verbs causes the main increase. Beyond this point, the gradient becomes smaller, and the increase of meaning causes the main increase. At the 100th decomposition (approximately 11 programs) the total number of pages of the verb dictionary page was 20.

Let us suppose that this system experiences additions, changes and modifications for 1000 hierarchical decompositions. As this corresponds to around 110 programs, it is an increase of around 6K lines. Extending the second trend line, the shortage of Verb Dictionary pages will be around 20. The designer has to add 20 Verb Dictionary pages during the design. Thus the automatic detailing rate will be (1000 - 20)/1000 = 98%. As the trend line is linear, the rate will approach closer to 100% as more experiences are accumulated. As the preparation of a Verb Dictionary is easy, this must be enough for such simple use. This seems to be the universal nature of a rule-based system.

The key factors for the simplicity and the small software size are as discussed earlier. This concept is worthy of further development.

3.4 Knowledge-based operation

As is shown in 2.2, the authors' challenge of knowledge-based systems was unsuccessful. It may be said that it was a reflection of the lack of the authors' technical experience. Generally, knowledge-based solutions are very powerful, and the appropriate way to use them is difficult. The golden rule may be to consider them when all the other ways were closed. It might be the same as what Zipf pointed out.

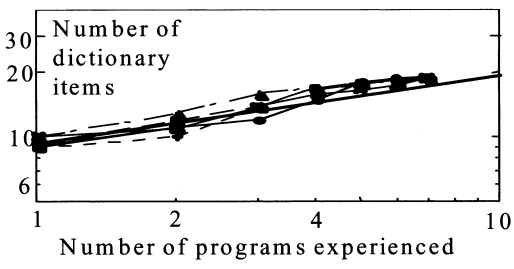

Figure 10. Learning curves of knowledge-based

Up to now, there were two cases. One is to fix MAP based on the past experiences, and another is to extend Jackson's program design case. In Jackson's program design case in Figure 1, as (Second, Minute and Hour) are common both in the input and output side children data. But, if hands were defined "Finest hand" for "Second" and so on, the system can not establish the pairing of the input

and output data. But, if the BCD includes both definitions of "time-based" as "Second" and "product-based" "Finest hand," the pairing becomes possible by "hunting the same word first then use BCD for conversion." This frequently appears in data processing. Thus, BCD is useful in these ways. Figure 10 shows the accumulated number of items of BCD as designs are repeated. Curves are for various sequences of programs. The learning effect also appears here. The logarithmic learning effects of skill-based, rule-based and knowledge-based cases have been shown. This means that in all cases, the logarithmic learning effect appears on the quantitative basis of knowledge. As has been shown in Figure 8, the knowledge increases in a logarithmic way, and thus accumulated knowledge enables the improvement of a human characteristic (e.g. productivity, error rate etc.) to show the logarithmic learning effect. Namely, as a person accumulates knowledge, the amount shows a logarithmic nature. Thus as knowledge is accumulated, it reflects the improvement of human characteristics. As it is accumulated in a logarithmic way, the improvement also shows a logarithmic effect. In short, the improvement is a reflection of the accumulated knowledge. Thus, the empirical law that most human characteristic show a logarithmic learning effect is explained by the quantity of knowledge.

Human designers are not so intelligent as these CASE tools. It is generally accepted in industry that it necessitates around "*ten years*" of accumulating experiences for a person to be an expert in "*one*" field. Therefore, *if the field of the product of a software development team (or a person's work area) changed, from time to time, is their technical expertise can grow up enough? If the team members were change, project by project, can their work process be stabilized enough?* The standard R&D pattern in industry is from "survey, research, experiments/trials, design and evaluation, and finally reaches to a stage of the mass production". *Is the development model in software industry "requirement acquisition, design, and implementation" can be the standard model?*

4. Discussion

This study of automatic design learning from human design was based on the commonality of human work, from business, various designs, and on to direct work in production. As the starting point is quite different from others, the study has resulted in a unique automatic design system. As the technology is still immature, the authors have to overcome various problems before this can make a contribution to the world. They plan to make it evolve into a highly reliability embedded system, and be used for intelligent control such as with humanoid robots.

As this study reveals idealistic human knowledge in human intentional activities, the application may effect related fields. As an example, the application in Software Engineering is explained. The hierarchically expanding network model in Figure 1 shows the work done there. Based on it, it may be theoretically proved that both man-hours and the number of errors are proportional to software size. The process is a linear system, where decomposition and integration are possible [9, 18, 23]. Additionally the authors show the learning effect of designs similar to hardware production work [21, 22].

The network model may also be applicable to the organization structure of a software development team. In the network, the uppermost part corresponds to Systems

designers/system analysts, in the middle there are design teams, and at the end there are programming teams. As a whole, they work as the network does. Namely, the hierarchical organization structure is a reflection of the knowledge possessed.

All these show that the methods of the hardware production process may also be applicable to the software development process[9, 18, 23, 28]. This was a common understanding in the Japanese software industry in 1960-70's, when the unique "Japanese Software Factories[19]" started. Although it is not widely known, software factories have been continuing to progress during these decades. They also developed and use an automatic design system [20].

If many software people understand this, the knowledge in a different way will contribute to the prosperity of the software industry.

5. Conclusion

In conclusion, quite a new way of automatic software design, by simulating human intentional activity, has been discussed.

1. Human works in management; business, design and production works belong to human intentional activity.
2. The ideal model for this is the repetitive hierarchical decomposition of concepts. Thus constructed hierarchical concepts constitute the knowledge, and a pair of the parent and the children is elementary process knowledge.
3. In a hierarchical decomposition, there may be various engines, as Zipf's "Principle of least effort" states.
4. Automatic design systems of software, simulating excellent human designers in three typical modes of operations, have been studied, implemented and evaluated. They are skill-based, rule-based and knowledge-based modes.
5. In the programming area at the end, the system can automate human designs near to 100%, by using skill-based and rule-based operations with some knowledge-based ones if needed.
6. Future works in the field of automatic design:
   - Joint development of these standardized IICASE tools by modifying off-the-shelf CASE tools based on standardized specification, enabling the cooperative operation of tools. Key tools are for Data Flow Diagram, Structured Chart and State Transition Diagram.
   - Field trials and their evaluation with friendly users in the field of high reliability embedded systems and intelligent control systems such as for humanoid robot applications.
   - Evolutional improvements, researches and refinements.
7. Greater effort toward "Human Intelligence" and wisdom toward the future world.

Acknowledgements

The authors wish to express their thanks to Dr. B.H. Far (Associate professor, University of Calgary) and others who contributed to the Software Creation Project. They are also very thankful to Mr. Daniel Horgan for his very careful checks and valuable advise on their English.

References

[1] H. Chen, K. Machida, B.H. Far and Z. Koono: Software Creation: A Systematic Construction Method of Expert Systems Used for Design, *Third World Congress on Expert System,* (1996), 577-584.

[2] H. Chen, B.H. Far and Z. Koono: A Systematic Construction Method of an Expert System Used for Automatic Software Design, *Journal of Japan Society of Artificial Intelligence* 12 (1997), 616-626.

[3] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Hafner Publishing, New York, 1973.

[4] J. Rasmussen: The Role of Hierarchical Knowledge Representation in Decision Making and System Management, *IEEE Trans. on Software Engineering*18 (1985), 523-533.

[5] H. Chen, N. Tsutsumi, H. Takano, and Z. Koono: Software Creation: An Intelligent CASE Tool Featuring Automatic Design for Structured Programming, *Journal of Institute of Electronics, Information and Communication Engineers,* E81-D (1998), 1439-1449.

[6] Z. Koono, H. Abolhassani and H. Chen: Automatic Knowledge Creating Method, Program therefore, Automatic Designing Method and Its System, *US Patent application,* No. 10/478980, Priority 28 May, 2001 (Japan), International Publication 5 Dec., 2002.

[7] H. Abolhassani, H. Chen and Z. Koono: Software Creation: Cliche as Inter-mediate Knowledge in Software Design, *The Journal of Institute of Electronics, Information and Communication Engineers,* E85-D (2002), 221-232.

[8] Z. Koono, B.H. Far, T. Baba, Y. Yamasaki, M. Ohmori and K. Hatae: Software Creation: Towards Automatic Software Design by Simulating Human Designers, *5th Int. Conf. on Software Engineering and Knowledge Engineering* (1993), 327-331.

[9] Z. Koono and H. Chen: Toward Quantitative, Rational and Scientific Software Process, *Proc of Software Process Workshop 2005* (2005), 454-458.

[10] Z. Koono, K. Ashihara and M. Soga: Structural Way of Thinking as Applied to Development, *IEEE/IEICE Global Telecommunications Conf.* (1987), 26. 6. 1-6.

[11] G. J. Myers: *Composite/Structured Design,* Litton Educational, New York, 1978.

[12] M. A. Jackson: *Principles of Program Design*, Academic Press, 1975.

[13] Carl von Klausewitz: *Vom Kriege*, Dummlers Verlag, Bonn, 1832.

[14] Koono, Z. and Chen, H., Automatic Acquisition Method of Knowledge and Its Automatic Design System, *Japanese Patent Laid-open* No. 10-320188, 4 December, 1998.

[15] H. Abolhassani, H. Chen, B.H. Far and Z. Koono: Software Creation: A Study on the Inside of Human Design Knowledge, *The Journal of Institute of Electronics, Information and Communication Engineers,* E83-D (2000), 648-658.

[16] H. Abolhassani, Z. Koono and H. Chen: Software Creation: Automatic Design by Rule-Based and Knowledge-Based Engines, *Report of IPSJ SE 138-15* (2002), 105-112.

[17] Editor G. Salvendi: *Handbook of Industrial Engineering,* John Wiley & Sons, Hoboken, 1982.

[18] Z. Koono, H. Chen and B.H. Far: Expert's Knowledge Structure Explains Software Engineering, *Joint Conference on Knowledge-Based Software Engineering* (1996), 193-197.

[19] M. A. Cusumano: *Japan's Software Factories; A challenge to U.S. Management,* Oxford press, 1991.

[20] T. A. Thayers et al.: Software Reliability Study, *Final Technical Report, RADC-TR-76-238,* Rome Air Development Center, Rome (Oneida County, Wisconsin, USA), 1976.

[21] Z. Koono, K. Igawa and M. Soga: Structural Way of Thinking as Applied to Improvement Process, *IEEE Global Telecommunications Conference,* (1988), 40. 1. 1-6.

[22] Z. Koono, H. Tsuji and M. Soga: Structural Way of Thinking as Applied to Productivity, *EEE International Conf. on Communications* (1990), 204. 2. 1-7.

[23] Z. Koono and H. Chen: Structure of Human Design Knowledge and The quantitative Evaluation (1/2), Technical Report of IEICE (2003), KBSE 2003-57.

[24] Z. Koono, T. Kimura, M. Iwamoto and M. Soga: A Stored Program Controlled Environmental Function Tester Based on FMM/SDL Design, International Switching Symposium 1987, B9.5.1-7.

[25] T. Yamada, M. Tashiro, H. Chen and Z. Koono: Software Creation: An Overview of Integrated Intelligent CASE tool, Technical Report of IEICE (2003), KBSE79 (2001) 21-24.

[26] S. Morimoto, K. Sudou and H. Chen and Z. Koono: Software Creation: Graphical Processing of Design Information in Integrated Intelligent CASE tool, Technical Report of IEICE (2003), KBSE80 (2001) 25-28.

[27] Q. Zhang, M. Yoshida, H. Chen and Z. Koono: Software Creation: Control Structure and the Evaluation of Integrated Intelligent CASE tool, Technical Report of IEICE (2003), KBSE81 (2001) 29-32.

[28] Z. Koono and H. Chen: A Software Process Featuring Quantitative Models, Report of IPSJ SE –147 (12) (2005), 89-96.