"Application of design process in mechanics to software design"
4th International Conference on Axiomatic Design
Firenze – June 13-16, 2006

APPLICATION OF DESIGN PROCESS IN MECHANICS TO SOFTWARE DESIGN

Kazuhiko Fujita
k.fujita@jp.fujitsu.com
Fujitsu Limited
140 Miyamoto, Numazu-shi
Shizuoka 410-0396, Japan

Takeshi Yoneyama
yoneyama@t.kanazawa-u.ac.jp
University of Kanazawa
Kakuma-machi,
Kanazawa 920-1192, Japan

Yotaro Hatamura
Hatamura@sozogaku.com
Hatamura Institute for the Advancement of Technology
Tokyo Opera City Tower 52F 3-20-2,
Nishishinjuku, Shinjuku-ku, Tokyo 163-1452, Japan

ABSTRACT

Design methodology has independently developed in each industry for practical production. Each industry carries out design upon common concepts of function, mechanism, structure and constraint separately from other industries. Our studies have revealed that original concepts in one field can map to their counterpart concepts. Our finding suggests that not only design processes established in one field apply to others, but also different industries can share knowledge of design and failure. This paper reports how a design process in mechanics applies to software design by first, laying out the design processes in mechanics and their essential concepts, then showing how each concept in mechanical design maps to its counterpart in software design through the common super-ordinate concept. We conclude the paper with a discussion on the effectiveness and interoperability of design and failure knowledge.

1 INTRODUCTION

Fujita has participated in the development of commercial software products and found that software designers have different thinking processes respectively. It has troubled him that they cannot sufficiently understand each idea and thought by sharing concepts in software design. On the other hand, Hatamura and Yoneyama have clarified basic thinking processes and concepts in mechanical design [Yoneyama, 1993], and [Hatamura, 1999]. They have revealed that designers can be creative if they are conscious of sharing the thinking processes and concepts in design. Fujita has considered counterparts of the mechanical thinking processes and concepts in software. Trough our discussion, it has become clear that there exist not only the thinking processes and concepts in mechanical design but also counterparts in software design.

Design process in mechanics can be divided into two parts, which are the thinking process and the externalizing process. The thinking process is an essential and creative work. The externalizing process is a work to organize the result of the thinking process as drawings and documents. A mechanical designer always analyzes an object to design, decides the design solutions and specifies his thoughts through a set of concepts in mechanics. We assume that the process is the same even in the different industry. If the set of concepts meaningfully corresponds to the other set in the different industry, then the design in the different industry will proceed according to the original thinking process.

We have found a relationship between the set of concepts in mechanical design and that in software design through the common super-ordinate concepts in consideration of some cases of design and failure. In this paper, we will show the basis of the thinking process in mechanics at first, and then abstract a set of concepts in it. Moreover, we will consider and select the counterparts in software design and finally, design software by using the thinking process with the set of concepts in mechanical design replaced by the one in software design.

2 APPLICATION OF DESIGN PROCESS IN MECHANICS TO SOFTWARE DESIGN

2.1 THINKING PROCESS AND A SET OF CONCEPTS IN MECHANICS

Figure 1 shows a basis of thinking process and externalizing process in mechanics [Yoneyama, 1993], [Hatamura, 1999], and [Hatamura, 2006].

(1) Make a plan for designing a machine according to the motivation and the purpose.
(2) Clarify the quantitative performance or the specification for the purpose.
(3) Analyze the functions and the constituents for the machine to possess.
(4) Consider which mechanisms should be adopted to satisfy the necessary functions. Show the whole mechanism by using a mechanism line diagram. A force flow diagram is also very useful to understand how to transfer the force within the machine. The optimum mechanism among the various concepts should be selected to satisfy the various constraints, i.e. not decide it on one idea only.

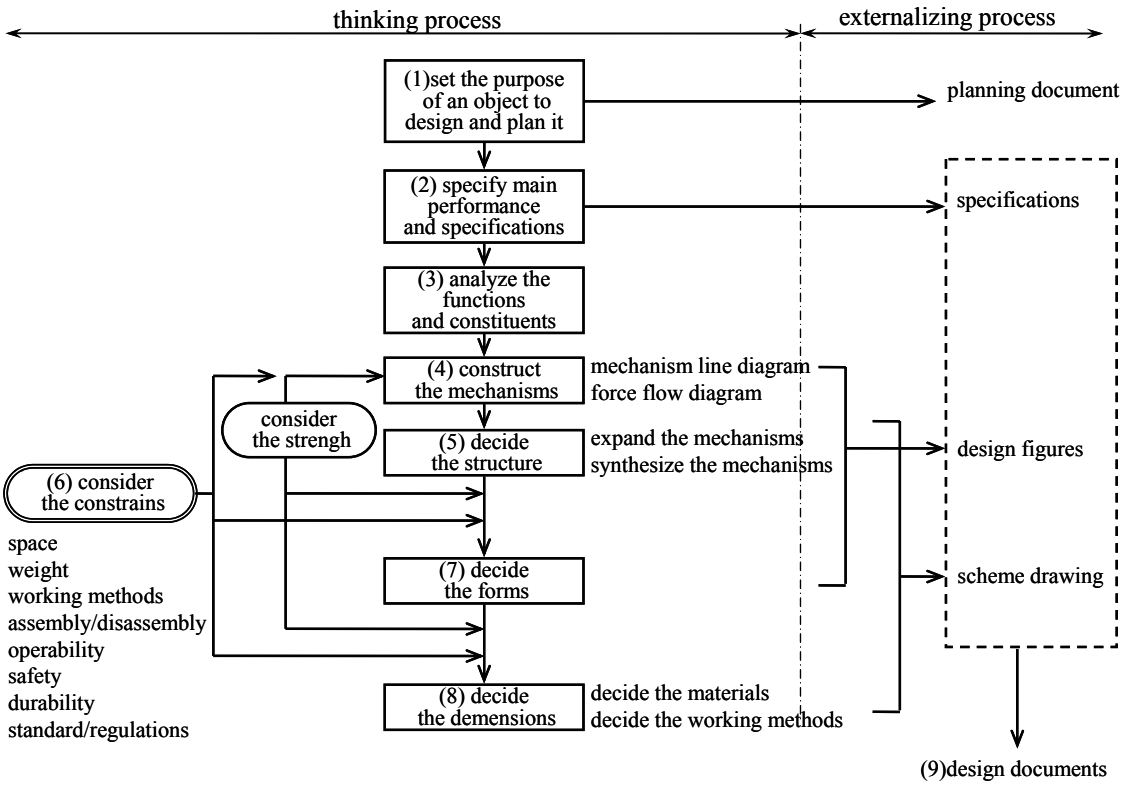

Figure 1. A basis of thinking process and externalizing process in mechanics

(5) Make the structure by expanding the basic mechanism line diagram and combining the various mechanisms. And specify the drawing while zigzagging among the constraints. Then, this makes the design figure grow upon and the ideas synthesized.

(6) Refine the mechanisms and the structures to satisfy the various constraints. Some of constraints are strength, stiffness, space, weight, workability, working method, assembly/disassembly, operability, durability, safeness, and cost.

(7) Decide the forms according to the mechanisms and the structures to satisfy the necessary functions.

(8) Decide the dimensions with the forms. The dimensions of the main parts can be decided by the condition of the main function in the specification. And then decide the dimensions of surroundings in consideration of the constraints about the space.

(9) Decide the materials and the working methods of each part to satisfy the constraints of strength, stiffness, weight and durability. Make scheme drawings with all decisions. And write the design documents to explain the thinking history of the decisions.

We have picked up the concepts such as performance, specification, function, mechanism, structure, constraint, force, form, dimension, material, space, weight, strength, stiffness, working and assembly/ disassembly.

2.2 COUNTERPART CONCEPTS IN SOFTWARE DESIGN

The highly abstract concepts are common between mechanical design and software design, such as performance, specification, function, mechanism, structure and constraint. In this section, we consider the particular concepts in mechanics.

Force is the most important and dominant concept in mechanics. If it is defined as an essential substance flowing within an object to design, then through the super-ordinate concept, data is the counterpart concept in software design because any kind of software can be modeled by input data, processing and output data. Data is always flowing in software.

Form is similarly defined as an external appearance of an object to design. It corresponds to interface in software. Many software designers think interfaces as the external appearances of program modules. Interface seems to be a form of software at the point of fitness of joint, assemblability, easy working and usability.

Dimension is defined as something to decide the characteristics of an object to design. It corresponds to algorithm of a module. In mechanical design, we can make a machine after all of the form and the dimension regarding entire parts have been decided. In software design, we can make software regarding entire

program modules after all of the interface and the algorithm have been decided.

Weight is defined as work per unit to move or operate an object to design. It corresponds to work per unit to execute a program module in software design. This is called time complexity, which is the number of instructions executed by CPU (Central Processing Unit). Software designers always use the phrase that this processing is light weight or heavy weight.

Space is defined as space for an object to occupy. It corresponds to space complexity. In software design, it means memory space for program modules to use, that is, the number of variables or the number of bytes in main memory.

Material is defined as something what an object to design is made from. It corresponds to programming language. It is also fitting at the point of workability. The selection of material is similar to the selection of programming language. This idea is based on strength and stiffness of material.

We should show that there also exist the counterparts of stress and strain derived from force at first. Mechanical designers are always thinking about the relationship between stress and strain in order to consider the deformation and the fracture of an object to design. That can be shown by a stress-strain diagram. We can explain the elastic region, the plastic region and the rupture by using it.

Stress is defined as an internal state of an object to design. It corresponds to a state of a module or a software system. Strain is also defined as the behavior caused in the state. It corresponds to the activity in the state, for example, to display a message to a console if a fault occurs, to calculate something when data is prepared or so on. The change of stress corresponds to a transition of a state. Those can be explained by a state diagram in software design.

Furthermore, under this idea, elastic strain is defined as the activity in a recoverable state, plastic strain is similarly defined as the activity in an unrecoverable state and rupture is defined as the activity in a state of a failure. In software design, it can be thought that the activity in a recoverable state and in an unrecoverable state is classified into the activity in an expected state, which has been decided previously, and the activity in a state of a failure is classified into the activity in an unexpected state, which has not been decided previously because the designer has not considered or omitted the state. The activity in the expected state is divided to two kinds of processing, one is a normal processing and two is an exception handling.

Figure 2 shows a state diagram for the program to sum the sequence of numeric data. Let us suppose that in the sequence of data, the numeric data is not only included but also the different kinds of data such as alphabet, sign or whitespace are included. Now, when the program is started, it goes into the state of waiting for input data. If numeric data is coming in the state, the state of the program transits to the state of normal input. At the time, the sum is calculated as the activity and then the program returns to the state of waiting for input data. If alphabet, sign or white space is coming, the state transits to the state of input error and an error message may be output as the activity, which is an exception handling. If the data is white space, then the program returns to the state of waiting for input data and otherwise the state transits to the state of fatal error. If end of data is coming in the state of waiting for input, the program is ended.

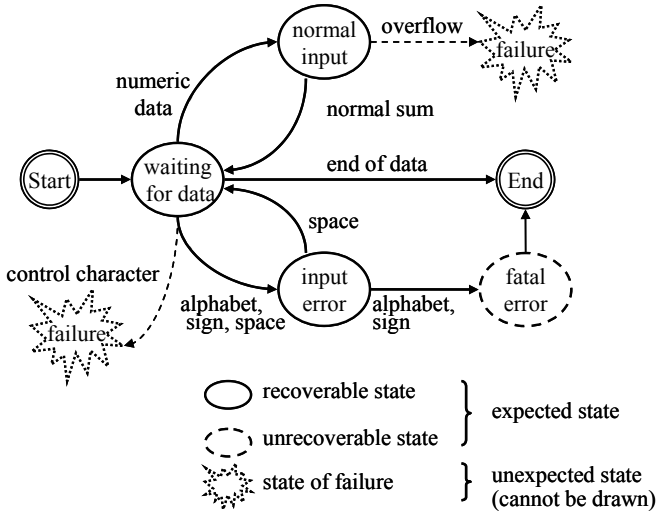

Figure 2. A state diagram of the program to sum the sequence of numeric data

Let us imagine the case that a control character data is coming. The input data function which is the activity in the state of waiting for input data will not work well, and then the program may be in the state of a failure. Similarly let us imagine the case that the calculated result overflows in the state of normal data. The program will be in the state of a failure even in case of numeric data.

In Figure 2, it is shown that there exist three kinds of states; one is a recoverable state, i.e. the state of waiting for input data, the state of normal input and the state of input error, two is an unrecoverable state, i.e. the state of fatal error, three is the state of a failure, which cannot be drawn in the state diagram actually because the designer has omitted the state. In the above considerations, we have determined the previous correspondence.

Strength is defined as a limit state within an object to design. It corresponds to the resiliency of the system, especially when under heavy load or when confronted with invalid input. It can be realized by proper exception handling. Software should have the processing for normal input data, that for invalid input data and that for error result. It also should work even if many exceptions occur due to shortage of system resources such as the memory area, the disk space or so on. The cover ratio of exception handling in all states decides the strength of software, i.e. the resiliency. We have already mentioned exception handling above by using Figure 2.

Stiffness is defined as adaptability of an object to design to environmental changes. It corresponds to system stability in software design. Let us consider the changes of the number of requests in an online system as environmental changes and set the average of the response time as an index of adaptability. The system with higher responsibility cannot drive both the grater throughput and the stable response time. However, if the constraint of the responsibility can be loosened, the system with the grater throughput and the stable response time can be

designed. Namely, even if the changes of the number of requests in the system are grater, the response time is stable, i.e. high adaptability. This is very similar to the concept of stiffness.

Table 1. Correspondence of concepts

Concept in mechanics	Super-ordinate concept	Concept in software
Performance	Common	Performance
Specification	Common	Specification
Function	Common	Function
Mechanism	Common	Mechanism
Structure	Common	Structure
Constraint	Common	Constraint
Force	Essential substance flow within an object	Data
Form	External appearance of an object	Interface
Dimension	Something to decide the characteristics of an object	Algorithm of a module
Weight	Work per unit to move or operate an object	Time complexity
Space	Space for an object to occupy	Space complexity
Material	Something what an object is made from	Language
Stress	Internal state of an object	State
Strain	Behavior caused by internal state of an object	Activity
Strength	Limit state within an object	System resiliency
Stiffness	Adaptability of an object to environmental changes	System stability
Working	The way to make an object	Programming
Assembly/ Disassembly	Composition/ Decomposition of objects	Linkage, Package

Working is defined as the way to make an object to design. It corresponds to programming. There exist many working methods by using various kinds of machines in mechanics such as a metalworking lathe or a milling machine or so on, and also many programming methods by using various techniques in software such as link, sort, queue, stack, lock or so on.

Assembly and disassembly are defined as composition and decomposition of objects. Assembly corresponds to linkage or package of program modules, or integration of software components. The reusability of software modules is always considered, however, the decomposition is not considered usually because they are not joined physically.

The correspondence of concepts in the above consideration is shown by Table1.

2.3 EXAMPLE OF APPLYING MECHANICAL DESIGN PROCESS TO SOFTWARE DESIGN PROCESS

Figure 3 shows the thinking process replaced by the set of concepts in software design according to Table 1. Let us design a software system according to the flowchart. Some values included in this plan are not actually important because the purpose is to verify the thinking process.

The step (1) is to decide the purpose and the plan about designing software. We have planned an online calculator system on which thousands of people can calculate the sum of two numbers simultaneously. Therefore, the system has thousands of terminals. The number of sever machines is just one. Let us suppose the OS (Operating System) is selected properly. We would also like to reduce the cost as low as possible.

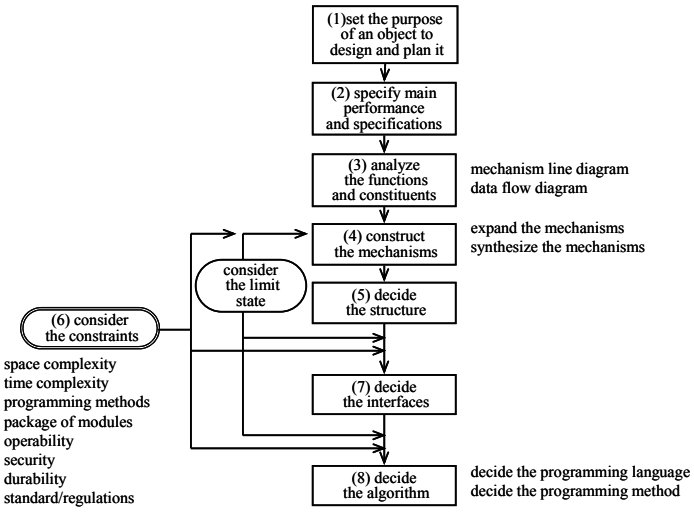

Figure 3. Thinking process replaced by software concepts

The step (2) is to decide the performance and the specification. The system throughput is 100 requests per second. The response time is within 1 second. However, the ability of each client machines and the sever machine is in their own way.

The step (3) is to analyze the functions and the constituents. Regarding a calculator, it should have the way to input data and to display the result. First of all, we must decide the kind of terminal in order to satisfy the requirements. Let us imagine that there are two selections, a limited hardware terminal or a general terminal. In case of selecting a limited hardware, this calculator will be realized using a key pad, a LCD (Liquid Crystal Display) and so on. In case of selecting a general one, this calculator will be realized using a web browser or limited software on PC (Personal Computer), on PDA (Personal Digital Assist), or so on. The limited hardware terminal is usually more expensive than the general one. Therefore, we have selected the general one, and to simplify the design, finally selected limited software. Then we will be able to select a simple client-server model. Figure 4 shows the functions required for the constituents based on client-server model. The figure combining a small square into a big square means a process. The extracted functions are the following: to invoke the processes of the clients and the server, to input data, to send a request message to calculate the sum of two numbers, to communicate with the server process, to receive the request message on the server, to calculate the sum, to send the response message of the result to the client process, to communicate with the client process, to receive the response message on the client and to display the result. In Figure 4, the two figures of processes show the simplest mechanism, and some arrows show the data flow. The thinking process to extract the functions proceeds along the data flow in the system.

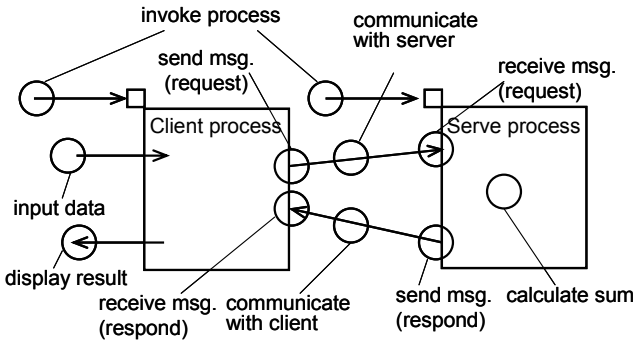

Figure 4. The functions required for the constituents based on client-server model

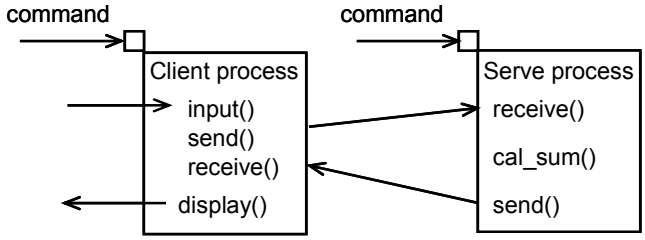

Figure 5. The first –level mechanism line diagram

The step (4) is the construction of mechanisms. The first-level mechanism line diagram is shown in Figure 5. The general OS provides the mechanisms to invoke a process by using a command, to input data and to display data. Those are shown as the command, the method of input() and the method of display(). It also provides the simple mechanism to send and receive a message and to communicate with the processes, i.e. the method of send() and the method of receive(). We must make the mechanism to calculate the sum such as the method of cal_sum().

However, we need to remember that there are thousands of client terminals in the system to decide the mechanism to send and receive request messages. If hundreds of requests arrive at the system, hundreds of server processes will be required because we would like to deal with the requests simultaneously. However, the server system will go slow or be down immediately if the server processes are invoked according to the requests of clients because the server system, which is just one, consumes many areas of memory in order to control hundreds of processes. According to this mechanism, it may be difficult to satisfy both the constraint of the number of the clients and that of the response time even if the server machine has higher performance.

The mechanism of queuing system is known well to solve this problem. We must consider overheads of the queuing system. However, we can decide the number of server processes independently because the server process can queue the request messages from the clients. If the queuing system has the function to reject the exceeding requests, the strength of the system is grater. Therefore, even if the number of the server process is just one, the system will work without the failure. We only tune up the ability of the processing by increasing the number of processes. This mechanism is actually available for the mission critical system, such as a banking system, a trading system and so on. Thus, this mechanism should be selected.

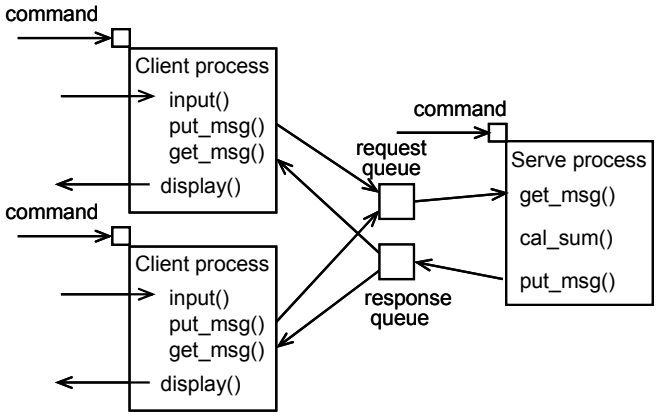

Figure 6. The mechanism line diagram(queuing system)

Next, let us select the same way in order to send and receive response messages. However, if the queues exist client by client, we need to make thousands of queues in the system. When the queues are made on the server, the problem of memory shortage will occur again because the queuing system consumes many areas of memory for the queues.

Then, let us consider the idea that the queues should be made on client side. In this case, the server process must identify the request message of each client, decide the corresponding queue, and respond to the client. The logic and the algorithm are very complicate. The better solution is that the server process has just one queue for the response, and each message has the identification, which each client process sets to the request message and the server process copies from the request message to the response message. Thus, each client can get the message by using the identification.

Figure 6 shows the mechanism line diagram adopting queuing system. In this figure, we will newly suggest the mechanisms of the method of put_msg() and the method of get_msg() instead of the method of send() and the method of receive(). These mechanisms should have the function to put a message with the specified identification to the specified queue, and the function to get a message with the specified identification from the specified queue. Thus, we make the decisions while zigzagging between the functional region and the mechanism and structure region.

The step (5) is to decide the structure. Figure 7 shows the whole mechanism on the client and Figure 8 shows that on the server. We can understand that if all methods are organized at the point of objects and methods, the classes such as terminal, queue, and calculator are necessary for the processes. The whole structure corresponding to each function is shown in Figure 9.

The step (6) is to consider the constraints. In this step, we have to consider the various constraints again, such as time complexity, space complexity, programmability, the construction of packages

of components, operability, security, and durability. Especially, exception handling, i.e. strength of the system should be considered by using some state diagrams like Figure 2.

The step (7) is to decide the interfaces. Each interface will be decided method by method in Figure 9. We must specify the parameters of the methods such as the identification to identify messages in the previous consideration, the message for request and response, the address of queues and so on.

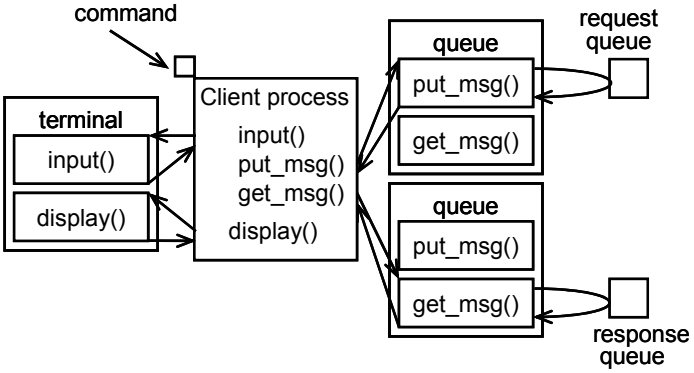

Figure 7. The whole mechanism on the client

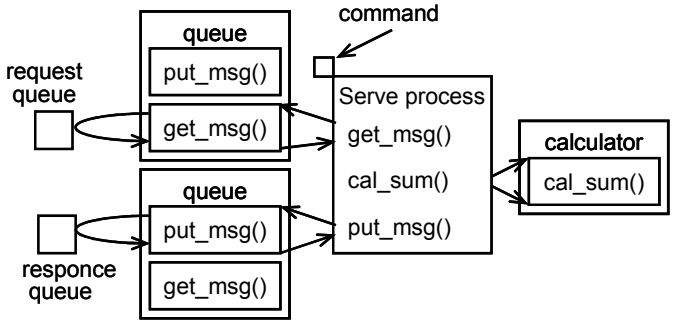

Figure 8. The whole mechanism on the server

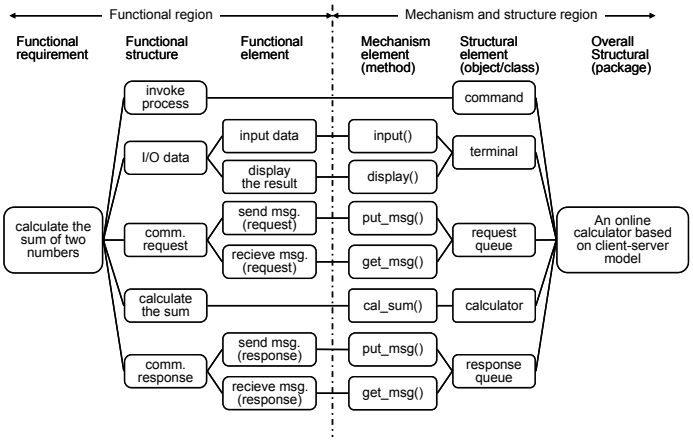

Figure 9. The whole structure corresponding to each function

The step (8) is to decide the algorithm. The algorithm of the main process and each method should be decided by considering each interface, each state of the objects and each state transition of them.

2.4 RESULT AND DISCUSSION

In the above consideration, we have shown that software design can proceed by using the thinking process in mechanics replaced by the set of concepts in software design. The reason why it is possible is that there exist the super-ordinate concepts between the set of concepts in mechanics and that in software. Moreover, at the point of software design view, we found that all main counterpart concepts can be extracted without exception.

Design methodology has the concepts embedded in the thinking process. Therefore, it is difficult to apply design methodology in a field to the other field. We have divided the thinking process with the embedded concepts to two parts, a thinking process independent of the concepts and the concepts. Therefore, it has become possible to apply design methodology in mechanics to that in software (Figure 10).

We are thinking that by using this result, software designers can understand each idea and thought efficiently, and software productivity will be improved. Moreover, failure knowledge will be also translated and used in each field.

We are also thinking that this result is deeply related to how designers recognize and understand an object to design. If the studies in this approach proceed, it is thought that the unified design methodology will have been realized, which has the unified thinking process and in which the concepts in each field can be exchanged.

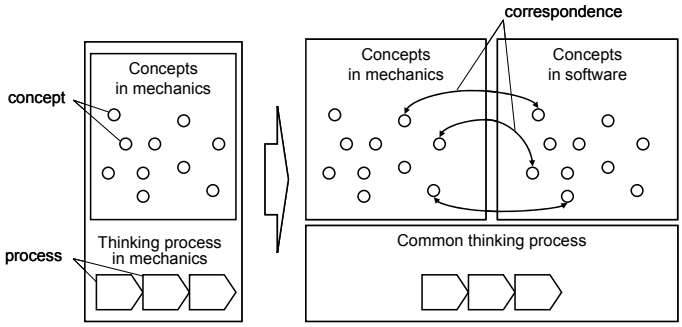

Figure 10. Application of design methodology in mechanics to that in software

3 CONCLUSIONS

In this paper, we have shown that design process in mechanics is applicable to software design by abstracting a set of concepts from the thinking process in mechanics and the counterpart concepts in software design.

Design methodology in each industry includes the particular concepts in the field. Therefore, it is difficult to apply it to the other industries. Our approach and findings help the establishment of the unified design methodology.

Our findings greatly help that software designers can understand each idea and thought in practical design in software development.

4 REFERENCES

[1] Yoneyama, T., 1993, Basic knowledge of machine design (in Japanese), *Nikkan-Kogyo Press*.

[2] Hatamura, Y., 1999, The Practice of Mechine Design, *Clarendon Press Oxford*.

[3] Hatamura, Y., 2006, Decision-Making in Engineering Design: Theory and Practice (Decision Engineering), *Springer-Verlag*.