# CONCEPTUAL SUPPORT FOR TEST CASE DESIGN

K.Tatsumi, S.Watanabe, Y.Takeuchi and H.Shimokawa

Quality Assurance Department, Software Division
Computer Software Development Group, FUJITSU LIMITED
140 Miyamoto, Numazu-shi Shizuoka,410-03 Japan

## ABSTRACT

A system has been developed to support the design of test cases in the black box testing field.

Designing test cases for large-scale software requires knowledge, not only of the functions of the tested, but also of the associated software and hardware. In the black box testing, test case designs are based on the external specifications or manuals. However, since not all of the factors influencing program operations (called input conditions) are described in the external specifications, the omission of test cases are liable to occurr. Several methodologies of test data selection have been proposed to date, but the problem of how to extract the input conditions of a program remains to be solved. The test case design support system described in this paper attempts to solve this problem by accumulating and utilizing the knowledge needed to design test cases and by generating test cases automatically based on the concepts of experimental design.

## INTRODUCTION

To date, the following methodologies of test data selection have been devised for black box testing.

- Equivalence partitioning[1)]
- Boundary value analysis[1)]
- Cause-effect graphing[1)]

Of these methodologies, equivalence partitioning and boundary value analysis are mainly used to analyze the values of the input conditions of the program to be tested. Cause-effect graphing is used to combine input conditions. These methodologies, however, do not solve all the problems encountered during black box testing. They pose problems such as these:

- The input conditions must be extracted before equivalence partitioning and boundary value analysis can be used; these methodologies do not provide a means of extracting the input conditions from the external specifications alone.
- Applying cause-effect graphing lacks a method to find out the relationship between causes and effects, so that it can hardly be extended to large-scale software in its present form.

The growing size of software and increasing complexity of cross-references in software have also caused stress on the following considerations:

- Testing large-scale software requires a knowledge of its broad context, including component programs and hardware.
- In tests involving many inspectors, a means is essential to standardize the quality of testing conducted by each inspector, but the ability to import knowledge and skills through training is limited.

This paper introduces the concepts and facilities of the test case design support system developed to deal with these problems.

## TEST CASE DESIGN PROCEDURES

Test case design follows these steps:

Step-1: Segmentation of functions. The functions are segmented to create an outline for the test case design operations to follow. This procedure is called test classification. As shown in Figure 1, first the functions are classified into hierarchical levels to define the test objective at each level. Then test are designed individually on the lowest hierarchical level.

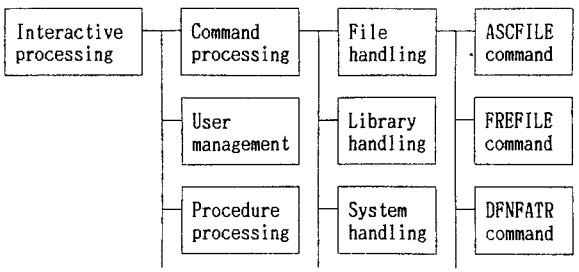


Figure 1. Test Classification

Step-2: Identification of test factors. The input conditions and their values are analyzed based on the external specifications of each segment of the test functions. This procedure is called test factor analysis. In this paper, input conditions are called factors and their possible values are called states. The factors and their states are collectively called test factors. Test factor analysis is conducted by entering the test factors identified from the external specifications into a test factor table as shown in Figure 2.

〔External specifications〕

| Command | Operand |
|---|---|
| ASCFILE | 〔ACNAME ( *access-name*) 〕<br>[DSNAME ({ * / *file-name* })]<br>[DUMMY]<br>. . . |

⇩

〔Test Factor table〕

| Factor / States | | A | B | C | D |
|---|---|---|---|---|---|
| | | Command input location | ACNAME operand | | |
| | | | Entry method | Parentheses | *access-name* length |
| States | 1 | Terminal | Spelled out (ACNAME) | Correctly specified | Minimum (1 character) |
| | 2 | Batch job | Mnemonic (ACN) | No left parenthesis | Intermediate |
| | 3 | | Omitted | No right parenthesis | Maximum (8 characters) |
| | 4 | | | | 9 or more characters |

Figure 2. Test Factor Table

Step-3: Generation of test cases. Test cases are generated by combining the states of the factors in the test factor table. This procedure is called test case generation. It is done by filling in a test case table as shown in Figure 3.

| Test case \ Factor | A | B | C | D |
|---|---|---|---|---|
| | Command input location | ACNAME operand | | |
| | | Entry method | Parentheses | *access-name* length |
| Test case 1 | ① (*1) Terminal | ① Spelled out | ① Correctly specified | ① Minimum |
| Test case 2 | ① Terminal | ② Mnemonic | ① Correctly specified | ③ Maximum |
| Test case 3 | ② Batch | ① Spelled out | ① Correctly specified | ② Intermediate |

*1 : The digits correspond to the number of the states entered in the factor analysis table.

Figure 3. Test Case Table

Step-4: Definition of test results. Expected results of each generated test cases are entered in a test results decision table.

The test case design support system helps the test factor analysis process and the test case generation process which are main processes of test case design procedures described above.

## TEST FACTOR ANALYSIS

Test factor analysis is conducted by extracting input conditions from the external specifications. This section describes the processes which support test factor analysis.

### Test Factor Analysis Process

The test factor analysis process passes through the following phases:

Factor analysis. A factor analysis is made by extracting input factors and environmental factors from the external specifications.

a) Input factors
Most of the functions of a program run on the basis of input explicitly supplied from outside in an instruction format, such as a command, control statement or statement. With commands, the input information is supplied in the command itself or by its operands.

b) Environmental factors
Program operations vary depending on the operating status of associated programs or hardware. These factors relating to associated programs and hardware are called environmental factors.

Analogy of associated factors. In factor analysis, if, for example, the factor 'file name' is extracted, possible associated factors, such as 'file organization' and 'block size', are analogically extracted from the word 'file' and selected or discarded as appropriate. The factor analysis often begins by extracting input factors from external specifications and proceeds to identify environmental factors not explicitly covered in the external specifications; that is, the process works by gaining a clue from one keyword (i.e. factor) to the next keyword.

State analysis. States can be analyzed in the following categories according to their type of values:

a) Factors indicating numeric values
These factors can assume the states of the values resulting from equivalence partitioning or boundary value analysis, as in the format 'maximum value, minimum value, intermediate value, zero...'

b) Factors specifying a selection format
With '*access name* = 〔*device-name* | *model-name* | DA〕', for example, these factors can assume all specifications, in this case '*device-name*, *model-name*, DA, default, invalid specification', as their states.

c) Factors in generic name format
For example, the factor 'direct access storage device (DASD),' can assume all types of DASD as its state. The generic name format can be considered a selection format, but since a choice is already made, selectable options are often not expresses in the external specifications.

Thus, state analyses can be broadly divided into logical analyses as in a) and analyses that enumerate established facts as in b) and c).

Test factor analysis methods. If the test factor analysis processes described above are organized into a map of test factor analysis methods, the result is shown in Figure 4.[2)]

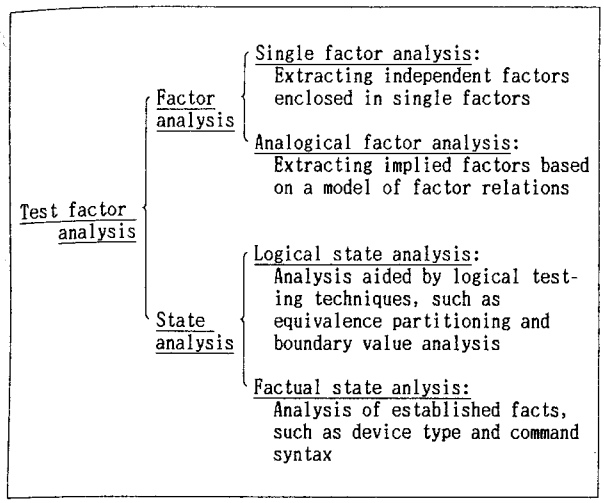


Figure 4. Test Factor Analysis Method

## Test Factor Analysis Support

A scheme permitting automated application of test factor analysis methods and accumulation of the factors and states as their entities is essential to support test factor analysis.

Test knowledge. The knowledge needed to conduct test factor analyses (called test knowledge) includes the test factor analysis methods and the factors and states that serve as their entities. Test knowledge can be divided into simple knowledge and associative knowledge as shown in Figure 5.[2)]

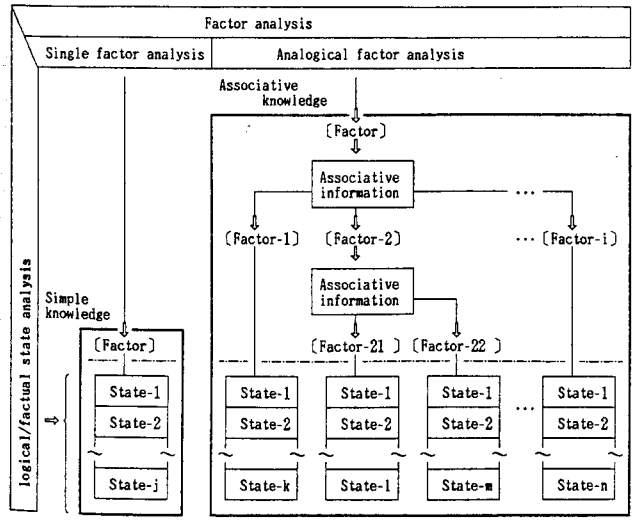


Figure 5. Test Knowledge

Accumulation of test knowledge. Test knowledge is accumulates in databases and utilized in the cource of test factor analysis.

a) Accumulation of simple knowledge
The states obtained by performing logical and factual state analysis are accumulated in a database keyed by factor name. This database is called the test factor database. Terms mentioned in the external specifications must be used as factor names to ensure that either different factors with identical names nor identical factors with different names are stored.

b) Accumulation of associative knowledge
Associative knowledge is a set of simple knowledge which is associated. The set is defined by assigning a supervisory name (called factor group name) to a group of associated factors. This information, too, is accumulated in a test factor database. Figure 6 illustrates stored associative knowledge.

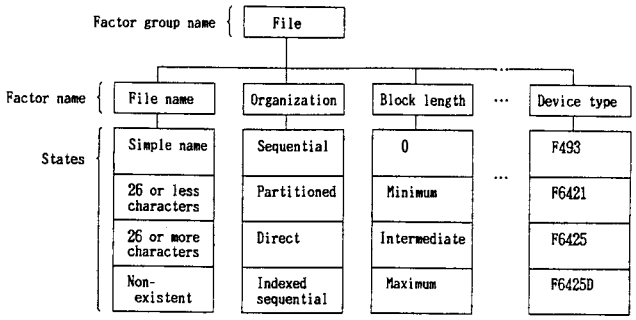


Figure 6. Associative Knowledge Storage

Test knowledge search. This test case design support system creates and edits factor analysis tables on a display screen. With this facility, test knowledge items are searched for during the creation of test factor tables in the following ways:

a) Simple knowledge search
The test factor database is searched using characters entered in the factor column as keys and the states accumulated in the database are displayed. Figure 7 shows this process.

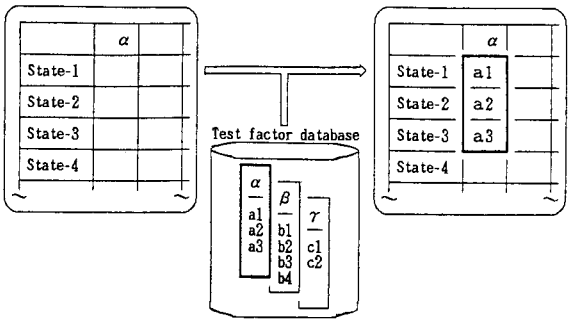


Figure 7. Simple Knowledge Search

b) Associative knowledge search
Associative knowledge is searched for in the same sequence as simple knowledge, and all associated test factors are displayed. Figure 8 shows this process.

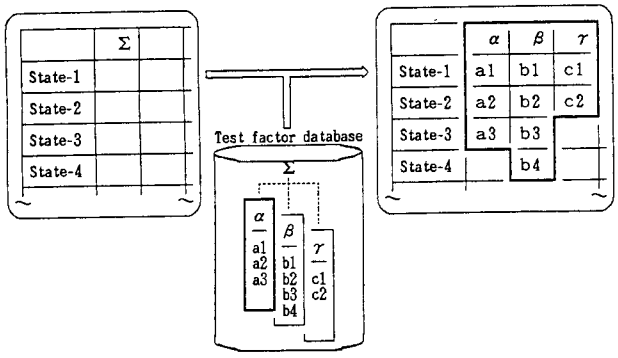


Figure 8. Associative Knowledge Search

## TEST CASE GENERATION

Test cases can be generated by combining the states of appropriate factors. One known method of accomplishing such combination is cause-effect graphing, but in actual application, this requires so much knowledge and labor to find out the relationship between causes and effects that it is not suitable as a tool for testing large-scale software.

To overcome this drawback, this system has chosen to combine the factors with regard only to the input conditions (that is, causes) while taking advantage of cause-effect graphing concepts, such as Boolean graphs and constraints, with the method of their combination being derived from the orthogonal arrays used in experimental design.

### Orthogonal array summary

**Significance of orthogonal arrays in experimental design.** In industry, there is a process of experimentally ascertaining the relationship of the causes influencing the characteristics that represent the quality of a finished product. Among the innumerable possible causes, the causes chosen individually as experimental objectives, or those organized in groups for better experimental efficiency, are called factors. The conditions of the factors that are experimented with are called factor levels, and the number of values used as factor levels is called a level count. Table 1 compares these terms with those used in this paper.

Table 1. Terms used in Tests and Experiments

| Tests | Experiments |
|---|---|
| Factor | Factor |
| State | Factor level |
| Number of states | Number of levels |

In the experiment design stage, factors and levels are selected, the interaction among the factors is considered, and so on. Finally, these procedures are combined in an operation similar to test factor analysis and test case generation as described in this paper. Experimental design is a technique used in this combination step. Orthogonal arrays are useful in experimental design.

**Orthogonal arrays.** Table 2 gives examples of an orthogonal array with two levels. In Table 2(a), any two columns have combinations (0,0), (0,1), (1,0), and (1,1) appearing once in the corresponding two rows. In Table 2(b), any two columns have these combinations appearing twice each. Arrays in which all combinations exist the same number of times between two random factors are called orthogonal arrays.[3] Orthogonal arrays are used to combine factors and levels in experimental design.

Table 2. Orthogonal Array (levels = 2)

(a)

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(b)

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |

### Application of orthogonal arrays to test case generation

**Standard of test case generation.** The following standard of test case generation with orthogonal arrays can be established:

- Standard-1: All combinations of states must always exist the same number of times between any two factors.

In the experimental field, it is essential that all combinations of states always exist the same number of times between any two factors to minimize the effects of measurement errors and other adverse conditions. Since, in software tests, all combinations of states need not exist the same number of times, standard-1 may be relaxed to read as follows:

- Standard-2: Each combination of states must always exist at least once between any two factors.[4]

Table 3 presents an orthogonal array (levels = 2) modified to meet standard-2. This table is called a combination table.[4] In case of three factors, the following four test cases are generated.

- Test case 1 : 1 1 1
- Test case 2 : 2 1 2
- Test case 3 : 1 2 2
- Test case 4 : 2 2 1

Table 3. Combination Table

| Test case \ Factor | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ··· |
|---|---|---|---|---|---|---|---|---|---|---|
| T1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| T2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 | 1 | 2 | |
| T3 | 1 | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 | |
| T4 | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 | 1 | |
| T5 | 1 | 1 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | |
| T6 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 1 | |
| T7 | 1 | 2 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | |
| T8 | 2 | 1 | 1 | 1 | 2 | 1 | 1 | 2 | 2 | |
| T9 | 1 | 2 | 1 | 1 | 2 | 1 | 2 | 2 | 1 | |
| ⋮ | | | | | | | | | | |

Note: The digits (1 or 2) in the table correspond to the number of states.

Application of the combination table to factor analysis tables. The combination table thus completed is applicable only to factors with two levels. This implies that the test factor tables must have been standardized to have two states for each their factors. Generally, however, meeting this requirement is impracticable, for even factors indicating numeric values can have more than two states, including a maximum, minimum, intermediate, greater than maximum, and less than minimum, and also factors seldom have a uniform number of states. It is hence necessary to process the factor analysis tables to meet this requirement.

To this end, a Boolean graph is created as shown in Figure 9, in which states and results are connected as input and output, respectively, using a logical instruction (each node always having two inputs), so that the combination table can be applied at each node.[4] Figure 10 shows the process of creating a test case table from a test factor table.

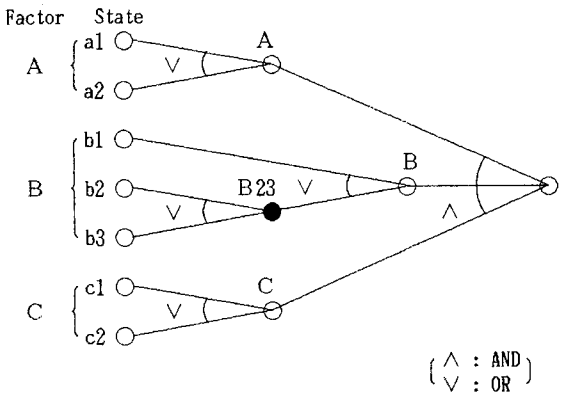


Figure 9. Boolean Graph for Application of the Combination Table

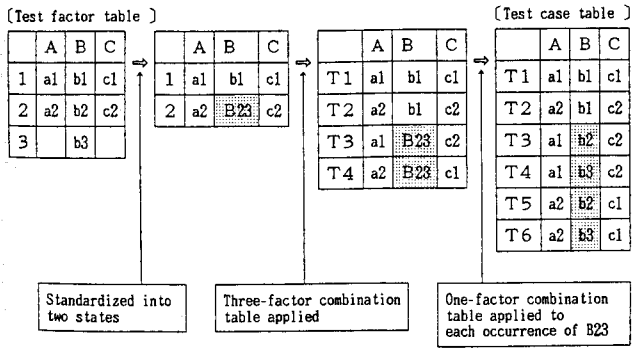


Figure 10. Application of the Combination Table to Test Factor Table

Constraint handling. A method of combinig input conditions has been presented. However, not all of the combinations of the factors and states entered in the test factor table actually exist. For example, a state may have an exclusive relation; that is, when it is selected, another factor or state cannot be selected. Conditions indicating such relations between factors and states are called constraints, as in cause-effect graphing. These conditions deserve special consideration in generating test cases.

Constraints are treated in the following ways:

- Exclusive:
  Test cases including a specified invalid combination of states are regarded as an error.
- Inclusive:
  Test cases not including a specified required combination of states are regarded as an error.
- One and only one:
  Test cases including two or more of the specified states are regarded as an error. (This means that test cases are normal when they include none or one of the specified states.)

## TEST CASE DESIGN SUPPORT SYSTEM

The preceding discussions have stated that the test case design support system assists the test factor analysis process by accumulating and utilizing the test knowledge and the test case generation process by application of the combination table. Furthermore, this system has the following facilities for enhancing its practicability:

- Test specifications (test factor table and test case table) editing functions
  Editing commands of this system enables to copy, move, insert and delete the factors, states and test cases on a display screen.
- Test specifications management functions
  In tests of large-scale software, many test specifications are made. This system manages test specifications by accumulating them according to their test classification codes and by recording their update history.

Figure 11 shows an overall picture of the test case design support system.

## CONCLUSIONS

As the tested software grows in size, it creates problems that cannot be resolved by a mere application of methodology. The test knowledge concepts of the test case design support system presented in this paper are the important key to these problems. But the test knowledge process is still primitive and the following problems are needed to be solved to enhance the effectiveness of this system.

- Automatic accumulation of test knowledge
  The validity of this system is affected by the amount of accumulated test knowledge. Automatic accumulation of test knowledge is necessary.
- Selective usage of test knowledge
  In the current system, test knowledge is selected by test factor names. But this is too broad and sometimes unnecessary entries were selected. The method for accumulating and selecting valid data is necessary.
- Formalizing of input for test case design support system
  In the black box testing, interpretation of information in external specification affects the quality of test case design. Formalizing of external specification and automation of test case design is the final goal.

## ACKNOWLEDGEMENTS

We would like to thank Yoshiaki Abe and the members of his team at Ryobi Systems Incorporated for their co-operation in the development of the test case design support system.


## REFERENCES


〔1〕 G.J. Myers, "The Art of Software Testing", Wiley-Interscience, 1976.

〔2〕 Y.Takeuchi, K.Tatsumi, "Knowledge Based System Oriented Test Factor Analysis System", collection of papers read at the 32nd (first half,1986) national covention (vol. Ⅱ), pp.1085-1086, Information Processing Society, 1986.

〔3〕 K.Ishikawa, et al, "Revised Elementary Experimental Design Text", Federation of Japanese Science and Technology, 1983.

〔4〕 H.Shimokawa, S.Satoh, "Method of Setting Software Test Cases Using the Experimental Design Approach", collection of papers read at the fourth symposium on quality control in software production, pp.1-8, Federation of Japanese Science and Technology, 1984.


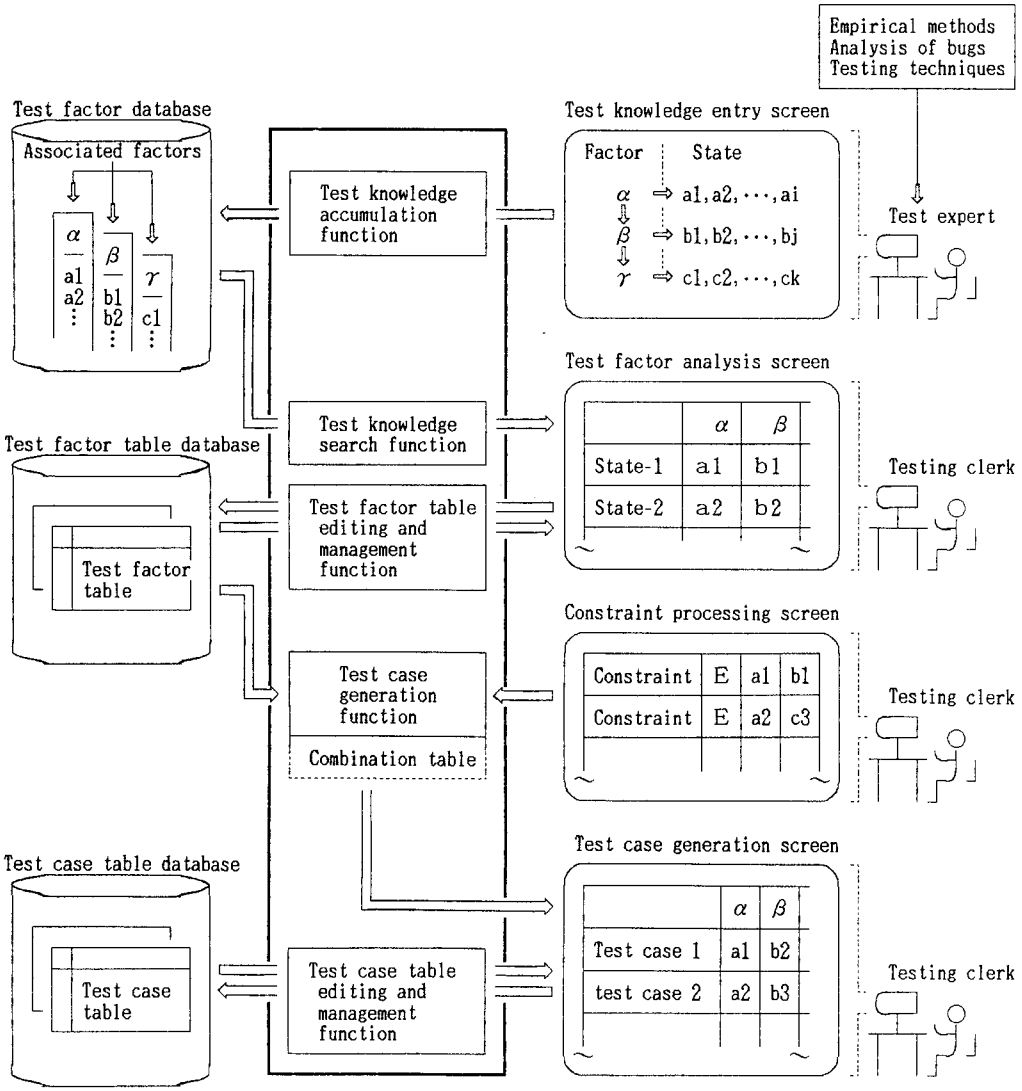


Figure 11. Test Case Design Support System Overview