IEICE TRANS. INF. & SYST., VOL. E78-D, NO. 9 SEPTEMBER 1995

PAPER Special Issue on Knowledge Based Software Engineering

Reconstructing Data Flow Diagrams from Structure Charts Based on the Input and Output Relationship

Shuichiro YAMAMOTO†, *Member*

SUMMARY The traceability of data flow diagrams against structure charts is very important for large software development. Specifying if there is a relationship between a data flow diagram and a structure chart is a time consuming task. Existing CASE tools provide a way to maintain traceability. If we can extract the input-output relationship of a system from a structure chart, the corresponding data flow diagram can be automatically generated from the relationship. For example, Benedusi et al. proposed a reverse engineering methodology to reconstruct a data flow diagram from existing code. The methodology develops a hierarchical data flow diagram from dependency relationships between the program variables. The methodology, however, transforms each module in structure charts into a process in data flow diagrams. The reconstructed diagrams may have different processes with the same name. This paper proposes a transformation algorithm that solves these problems. It analyzes the structure charts and extracts the input and output relationships, then determines how the set of outputs depends on the set of inputs for the data flow diagram process. After that, it produces a data flow diagram based on the include operation between the sets of output items. The major characteristics of the algorithm are that it is simple, because it only uses the basic operations of sets, it generates data flow diagrams with deterministic steps, and it can generate minimal data flow diagrams. This process will reduce the cost of traceability between data flow diagrams and structure charts.

key words: *structured analysis, reverse engineering, dataflow diagram, structured chart, diagram synthesis*

1. Introduction

The traceability of data flow diagrams against structure charts is very important for large software development. Specifying if there is a relationship between a data flow diagram and a structure chart is a time consuming task. Existing CASE tools provide a way to maintain traceability. If we can extract the input-output relationship of a system from a structure chart, the corresponding data flow diagram can be automatically generated from the relationship. For example, Benedusi et al. proposed a reverse engineering methodology to reconstruct a data flow diagram from existing code. The methodology develops a hierarchical data flow diagram from dependency relationships between the program variables. The methodology, however, transforms each module in structure charts into a process in data flow diagrams. The reconstructed diagrams may have different processes with the same name. Minimization of data flow diagrams could be effective to eliminate these detailed design level processes. To minimize data flow diagram, elimination of redundant data flow names as well as redundant processes is needed.

This paper proposes the transformation algorithm that solves these problems. It analyzes the structure charts and extracts the input and output relationships, then determines how the set of outputs depends on the set of inputs for the data flow diagram process. After that, it produces a data flow diagram based on the include operation between the sets of output items. Figure 1 shows the strategy of the algorithm. This strategy will be useful for hiding the detailed design level information from data flow diagrams, since it reduces redundant processes. Moreover, it is convenient since it does not use the direct functional mapping between modules of structure charts and processes of data flow diagrams. This process will reduce the cost of traceability between data flow diagrams and structure charts.

This paper is organized as follows. In Sect. 2, the minimality of data flow diagrams is defined. In Sect. 3, an algorithm that automatically generates a data flow diagram from an input-output relationship is proposed. In Sect. 4, the effectiveness of the proposed approach is discussed. In Sect. 5, the algorithm is extended and applied to reconstruct data flow diagrams from structure charts. In Sect. 6, the algorithm is applied to the structure chart of patients monitoring system [2] and the usefulness of the approach is examined. Finally, in Sect. 7, we close with our conclusion and remaining issues.

2. A Minimal Data Flow Diagram

A data flow diagram is a flexible notation by which system analysts can visually describe system functions [1]. When several analysts cooperatively develop data flow diagrams, they may, individually, have different ways of describing the diagrams. Without standard-

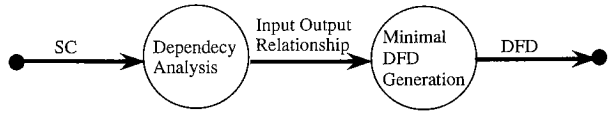


Fig. 1 Strategy of specification recovery.

Manuscript received November 28, 1994.
Manuscript revised April 10, 1995.

† The author is with Software Laboratories, Nippon Telegraph and Telephone, Musashino-shi, 180 Japan.

ized description rules, modifications are likely to occur when the descriptive rules are unified during the integration of individual data flow diagrams. If the notion of the minimal data flow diagram can be defined, the integration of the data flow diagrams will be very easy since individual data flow diagrams are normalized by their minimal forms. Several basic notions of data flow diagrams are necessary to define the minimality of data flow diagrams.

Definition 1: Data flow diagram

A data flow diagram is a graph $G=\langle E_I, E_O, P, F_I, F_O, F_P\rangle$. Here $E_I=\{e_i|$ an input data flow I_i comes from the external node $e_i\}$, $E_O=\{e_j|$ an output data flow O_j goes to the external node $e_j\}$, P is a set of all processes, F_I is a set of data flow between elements of E_I and P, F_O is a set of data flow between elements of P and E_O, and F_P is a set of data flow between processes.

Definition 2: Input set and output set for a process

For each process p of the data flow diagram $G=\langle E_I, E_O, P, F_I, F_O, F_P\rangle$, input set $In(p)$ and output set $Out(p)$ are defined as follows.

$In(p)=\{I_i|$ there is a path of data flow from e_i to p in G, $e_i\in E_I$, $p\in P$, I_i is the input data flow comes from the external node $e_i\}$

$Out(p)=\{O_i|$ there is a path of data flow from p to e_i in G, $e_i\in E_O$. $p\in P$, O_i is the output data flow goes to the external node $e_i\}$

From the definition of input set and output set, the following proposition holds.

Proposition 1: For the processes p and q of a data flow diagram, if there is a path from p to q then

$In(p)\supseteq In(q)$ and $Out(p)\supseteq Out(q)$.

Proof:

$In(q)=\{I_i|$ there is a path of data flow from e_i to q in G, $e_i=E_I$, $q\in P$, I_i is the input data flow comes from the external node $e_i\}$

Since there is a path from p to q in G by the assumption, $In(q)$ can be divided as follows.

$In(q)=\{I_i|$ there is a path of data flow from e_i to p and there is a path from p to q in G, $e_i\in E_I$, $p\in P$, I_i is the input data flow comes from the external node $e_i\}$

$\cup\{I_i|$ there is a path of data flow from e_i to q and the path does not go through p in G, $e_i\in E_I$, $p\in P$, I_i is the input data flow comes from the external node $e_i\}$

$=In(p)\cup\{I_i|$ there is a path of data flow from e_i to q and the path does not go through p in G, $e_i\in E_I$, $p\in P$, I_i is the input data flow comes from the external node $e_i\}\supseteq In(p)$

$Out(p)\supseteq Out(q)$ can be shown in the same way.

Figure 2 shows this input and output set relationship between processes on a data flow path. In case that each data flow has different name, if the set of inputs necessary for y contains every input necessary for x, $In(p)\subset In(q)$, there should be a data flow path from p to q where p and q are the processes from which data flow x and y go to external sources. In this case, the set of outputs from p includes the set of outputs from q, $Out(q)\subset Out(p)$.

Definition 3: Input output relationship of data flow diagram

For a data flow diagram $G=\langle E_I, E_O, P, F_I, F_O, F_P\rangle$, input output relationship $D(G)$ is defined as follows.

$D(G)=\{(I_i, O_j)|$ there is a path of data flow from e_i to e_j in G, $e_i\in E_I$, $e_j\in E_O$, I_i is the input data flow comes from the external node e_i, O_j is the output data flow goes to the external node $e_j\}$

Definition 4: Maximum set and minimal set

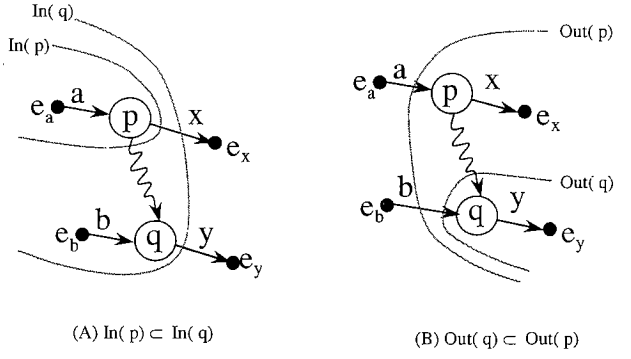
For any sets X and Y, X is the maximum set that is included in Y if there is no set Z such that $X\subset Z\subset Y$.

For any sets X and Y, X is the minimal set that includes Y if there is no set Z such that $Y\subset Z\subset X$.

Definition 5: Minimal data flow diagram

The data flow diagram $G=\langle E_I, E_O, P, F_I, F_O, F_P\rangle$ is minimal if the following conditions hold.

(C1) For any p_i, $p_j\in P(i\neq j)$, $In(p_i)\neq In(p_j)$ and $Out(p_i)\neq Out(p_j)$.



(A) In(p) ⊂ In(q)

(B) Out(q) ⊂ Out(p)

Fig. 2 Basic relationships between input sets and output sets.

(C2) For any $f \in F_P$, if f comes from $p \in P$ and goes to $q \in P$, $In\ (p)$ is the maximum set that is included in $In(q)$ and $Out(p)$ is the minimal set that includes $Out\ (q)$.
(C3) Input and output data flow names should be uniquely defined.

Condition 1 reduces redundant processes. Condition 2 reduces redundant flows among processes. Condition 3 eliminates the repetition of names among data flows.

3. An Algorithm for Generating a Data Flow Diagram

For each input and output, the set of outputs is extracted from the relationship between the input and output in the following way. For each input a, if a is necessary for output x, then x is an element of the output set for input a. For each output x, y is an element of the output set for x, if y depends on every input necessary for x.

In case that each data flow has different name, if there is a data flow path from P to Q, an output derived from Q is also derived from P. Since each output set corresponds to a process of the data flow diagram, we can generate data flows between processes from the include relationship between output sets. This is the basic notion of the algorithm.

Algorithm 1

Step 1: Determine the relationship, D, between an input set I and an output set O, where

$D=\{(a, x)|$ an input $a \in I$ is necessary for an output $x \in O\}$

Step 2: Determine an output set R_a for each input $a \in I$, where

$R_a=\{x|(a, x)$ is in $D\}$

Step 3: Determine an output set R_x for each output $x \in O$, where

$R_x=\{y|$ for each input a, (a, y) is in D if (a, x) is in $D\}$

Step 4: Construct processes for each element R_k of set R based on the following condition, where $R=\{R_a|a \in I\} \cup \{R_x|x \in O\}$.
(1) For $R_k \in R$, if there is no $j (j \neq k)$ such that $R_j \in R$ and $R_k = R_j$ then a process is generated for R_k.
(2) For $R_k \in R$, if there is $j (j \neq k)$ such that $R_j \in R$ and $R_k = R_j$ then a process is generated for the set $\{R_j|R_j=R_k, j \neq k, R_j \in R, R_k \in R\}$.
Step 5: Generate a data flow from process P to process Q if the following conditions hold:
(1) Process P and Q are constructed from R_1 and R_2 in R.
(2) R_1 is a minimal set that includes R_2. In other words, R_2 is a maximum set that is included in R_1.
Step 6: Generate an input data flow a to process P from an external source if P is constructed from R_a in R.
Step 7: Generate an output data flow x from process P to an external source if P is constructed from R_x in R.

Example 1

Step 1: For example, consider the following relationship D between an input set $\{a, b, c\}$ and an output set $\{x, y, z\}$.

$D=\{(a, x), (b, x), (b, y), (c, y), (c, z)\}$

Step 2: Determine the output sets R_a, R_b, and R_c for the input symbols.

$R_a=\{x\}$

$R_b=\{x, y\}$

$R_c=\{y, z\}$

Step 3: Determine the output sets R_x, R_y, and R_z for the output symbols.

$R_x=\{x\}$

$R_y=\{y\}$

$R_z=\{y, z\}$

Step 4: Construct processes for each element of a set $R=\{R_a, R_b, R_c\} \cup \{R_x, R_y, R_z\}$.

Construct $P1$ for $R_a=R_x=\{x\}$

Construct $P2$ for $R_b=\{x, y\}$

Construct $P3$ for $R_c=R_z=\{y, z\}$

Construct $P4$ for $R_y=\{y\}$

Step 5: Generate data flows between the processes.

Generate a data flow from $P2$ to $P1$ because $\{x, y\}$ includes $\{x\}$.

Generate a data flow from $P2$ to $P4$ because $\{x, y\}$ includes $\{y\}$.

Generate a data flow from $P3$ to $P4$ because $\{y, z\}$ includes $\{y\}$.

Step 6: Generate input data flows to the processes from external sources.

The input data flow names of $P1$, $P2$, and $P3$ are a, b, and c, respectively.

Step 7: Generate output data flows from the processes to external sources.

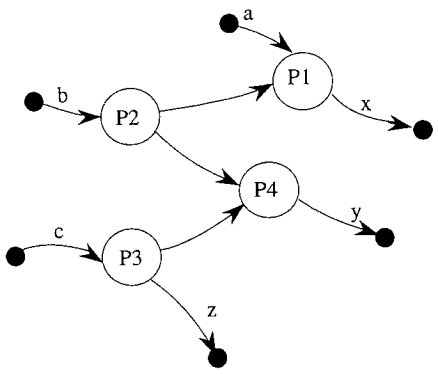


Fig. 3 Example of a minimal dataflow diagram.

The output data flow names of $P1$, $P3$, and $P4$ are x, y, and z, respectively.

Figure 3 shows the generated data flow diagram from relationship D between the input set $\{a, b, c\}$ and the output set $\{x, y, z\}$.

4. Effectiveness

The proposed algorithm is effective because it has the following properties. First, it can generate a data flow diagram whose relationship between inputs and outputs is equivalent to the given ones (Correctness). Second, it uniquely generates a minimal data flow diagram (Minimality). Third, every data flow diagram can be transformed to the minimal form by the algorithm (Completeness).

Proposition 2: Correctness
The algorithm generates a data flow diagram whose relationship between inputs and outputs is equivalent to the given ones.

Proof: To show the correctness of the algorithm, it is sufficient to prove the following proposition 3.

Proposition 3:
If the algorithm generates a data flow diagram $G=\langle E_I, E_O, P, F_I, F_O, F_P\rangle$ for the given inputs $I=\{I_i\}$, outputs $O=\{O_j\}$ and input-output relationship $D=\{(I_i, O_j)\}$, then there is a path from the external node e_i to the external node e_j in G if and only if (I_i, O_j) is in D.

Proof: If there is a path w from e_i to e_j in G, we have two cases.
Case 1: There is a process p, and path w is divided by two data flows, (e_i, p) and (p, e_j).

For process p, there is an output set RI_i and the input data flow (e_i, p) is generated from RI_i in Step 6 of the algorithm. Also for process p, there is an output set RO_j and the output data flow (p, e_j) is generated from RO_j in step 7 of the algorithm. RI_i is equivalent to RO_j because input data flow I_i and output data flow O_j share the same process p. RO_j contains O_j by definition. Therefore, we can state that RI_i contains O_j because $RI_i=RO_j$. Consequently, (I_i, O_j) is in D.
Case 2: Path w is divided into path v and two data flows (e_i, p) and (q, e_j), where v is a path from p to q.

Suppose that RI_i and RO_j are the output sets for processes p and q, respectively. Then RI_i includes RO_j by the condition of Step 5. RO_j contains O_j. Therefore, RI_i contains O_j. By definition, if RI_i contains x, then (I_i, x) is in D. Now we have (I_i, O_j) in D, by replacing x as O_j.
The opposite part of the proposition is clearly derived from the construction of the algorithm.

Proposition 4: Minimality
The algorithm generates the minimal data flow diagram G.

Proof: To show the minimality of the data flow diagram generated by the algorithm, it is sufficient to prove the generated diagram satisfies Condition 1 through Condition 3 in Sect. 2.

If G does not satisfy Condition 1, we have 2 cases.
Case 1: G has different processes P and Q, such that the set of outfputs depending on input of P, $Out(P)$, is equivalent to the set of outputs depending on input of Q, $Out(Q)$. This conflicts with the fact that the algorithm generates only one process for each output set. Therefore, G satisfies Condition 1.
Case 2: G has different processes P and Q, such that the set of inputs necessary for output of P, $In(P)$, is equivalent to the set of inputs necessary for output of Q, $In(Q)$. Suppose P and Q are generated for output sets R_x and R_y, respectively. If X is in R_x, then X is also in R_y. Otherwise, $In(P)$ is not equivalent to $In(Q)$. This indicates that R_x is included by R_y. In the same way, R_y is included by R_x. Therefore, $R_x=R_y$. This conflicts with the fact that the algorithm generates only one process for each output set. Therefore, G satisfies Condition 1.

If G does not satisfy Condition 2, we have 2 cases.
Case 1: G has different processes P and Q, such that the set of outputs depending on inputs of P, $Out(P)$, is the minimal set that includes the set of outputs depending on inputs of Q, $Out(Q)$, and there is no data flow from P to Q. This contradicts the fact that the algorithm generates a data flow P to Q, because output sets of P and Q satisfy the conditions of Step 5. Therefore, G satisfies Condition 2.
Case 2: G has different processes P and Q, such that the set of inputs necessary for outputs of P, $In(P)$, is the maximum set that is included in the set of inputs necessary for outputs of Q, $In(Q)$, and there is no data flow from P to Q. Suppose P and Q are constructed from output sets R_x and R_y, respectively. For any element a of $In(P)$, if (a, x) is in D then (a, y) is in D, because $In(P)$ is included in $In(Q)$. In this case, y is in R_x, by the definition in Step 3. Therefore, R_x includes R_y. By Step 5, a data flow P to Q should be

generated. This is a contradiction. Consequently, G satisfies Condition 2.

If G has different data flows with the same name, this conflicts with either Step 6 or Step 7 of the algorithm. Therefore, the algorithm generates a data flow diagram which satisfies Condition 3.

Proposition 5: Completeness

For any data flow diagram G, G can be transformed to the minimal data flow diagram G' with the algorithm.

Proof: For any data flow diagram G, we can always develop an input-output relationship D according to the definition 3 by the data flow path analysis of G. Then, we have the minimal data flow diagram G' for the input-output relationship D by the algorithm. It is clear that the G and G' have the same input-output relationship D. This shows the correctness of the transformation.

5. A Method for Reconstructing Data Flow Diagrams from Structure Charts

The traceability of data flow diagrams against structure charts is very important for large software development. Specifying if there is a relationship between a data flow diagram and a structure chart is a time consuming task. Existing CASE tools provide a way to maintain traceability. If we can extract the input-output relationship of a system from a structure chart, the corresponding data flow diagram can be automatically generated from the relationship. For example, Benedusi et al. [3] proposed a reverse engineering methodology to reconstruct a data flow diagram from existing code. Byrne [4] reported an experiment of the methodology. The methodology develops a hierarchical data flow diagram from dependency relationships between the program variables. The reconstructed diagrams, however, may have different processes with the same name. Also, the methodology generates the detailed level processes in data flow diagrams from each module in structure charts.

Recently, O'Hare [5] proposed an reverse engineering tool named RE-Analyzer. It can generate a hierarchy of data flow diagrams from source code by analyzing the control partitions. The problem of the generated diagrams is the sparseness. They are liable to have many isolated processes which have no data flow with others. This is because they are extracted from the control blocks of source code.

These problems can be addressed by applying Algorithm 1 in Sect. 3, because it can remove redundant processes in a data flow diagram and make the diagram minimal. The following Algorithm 2, a revised version of Algorithm 1, gives the data flow diagram generation procedure from a structure chart. This process will reduce the cost of traceability between data flow diagrams and structure charts.

Algorithm 2

Step 1: Determine the relationship between input and output parameters of modules D for a structure chart S, where

$D=\{(a, x) |$ for an arbitrary module m in S, a is an input to m and x is an output from m, and there is a transformation that x is generated from $a\}$

For the module m that has only inputs, no elements of D are derived from the parameters of m, because the input-output relationship is not extracted from them. For the same reason, for the module that has only outputs, no elements of D are derived. For the data that path through a module, the input-output relationship is not extracted from it, because the data is not transformed in the module.

Step 2: If D contains (a, b) and (b, x) but does not contain (a, x), then (a, x) is added to D. This procedure continues until no new element is added to D.

Step 3: Determine an output set R_a for each input $a \in I$, where

$R_a=\{x|(a, x)$ is in $D\}$

Step 4: Generate a process for each R_k on the following condition, where R_k is an element of the set $R=\{R_a| a \in I\}$.

(1) For $R_k \in R$, if there is no $j (j \neq k)$ such that $R_j \in R$ and $R_k=R_j$ then a process is generated for R_k.

(2) For $R_k \in R$, if there is $j (j \neq k)$ such that $R_j \in R$ and $R_k=R_j$ then a process is generated for the set $\{R_j| R_j=R_k, j \neq k, R_j \in R, R_k \in R\}$.

Step 5: Generate input a from external sources to the process P generated for R_a, if there is no output set R_k that includes R_a, where R_k is an element of the set $R=\{R_a|a \in I\}$.

Step 6: If R_a includes R_b and there is no R_c such that R_a includes R_c and R_c includes R_b, then generate an internal data flow b from process P to Q, where P and Q are generated for R_a and R_b, respectively.

Step 7: If there is no output set R_k that is included in R_a, and R_a contains x, then generate output x to external sink from process P, where P is generated for R_a and R_k is an element of the set $R=\{R_a|a \in I\}$.

Step 8: The processes are named by the following conditions.

(1) If process P and module m have the same inputs and outputs, then the name of P is set to the name of m.

(2) If process P has only one input a, P is named by the name of the module that reads input a.

(3) If process P has only one output x, P is named by

the name of the module that writes output x.

Example 2

Step 1: The following relationship D is extracted from the structume chart in Fig. 4.

$$D=\{(a, b), (b, c), (b, d), (d, x), (d, y), (c, z)\}$$

For example, module P has an input parameter a and an output parameter b. Therefore (a, b) is in D.
Step 2: For input a, (a, c) is added to D because (a, b) and (b, c) are in D. In the same way, (a, d), (a, x), (a, y), and (a, z) are added to D.

For input b, (b, x) is added to D because (b, d) and (d, x) are in D. Also, (b, y) and (b, z) are added to D. For inputs c and d, no new input and output pair is added to D, because x, y and z do not appear in the left part of any of the pairs in D.
Consequently, we have

$$D=\{(a, b), (a, c), (a, d), (a, x), (a, y), (a, z), (b, c), (b, d), (b, x), (b, y), (b, z), (c, z), (d, x), (d, y)\}$$

Step 3: Determine R_a, R_b, R_c and R_d from D.

$$R_a=\{b, c, d, x, y, z\}$$

$$R_b=\{c, d, x, y, z\}$$

$$R_c=\{z\}$$

$$R_d=\{x, y\}$$

Step 4: Generate processes for R_a, R_b, R_c and R_d.
Step 5: For the process of R_a, an external input a is generated because there is no output set R_k that includes R_a.
Step 6: The following data flows are generated.

Data flow b from the process of R_a to that of R_b.

Data flow c from the process of R_b to that of R_c.

Data flow d from the process of R_b to that of R_d.

Step 7: An external output z, which goes from the process for R_c, is generated because there is no output set R that is included in R_C. For the same reason, external outputs x and y are generated from the process for R_d.
Step 8: The names of the processes for R_a, R_b, R_c and R_d are set to P, Q, R and S, respectively. For example, the process for R_a has input a and output b. Module P also has input a and output b. Therefore, P is the name of the process for R_a.
Figure 5 gives the generated data flow diagram from the structure chart in Fig. 4.

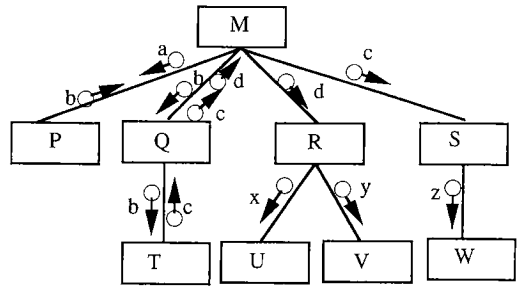


Fig. 4 Example of a structure chart.

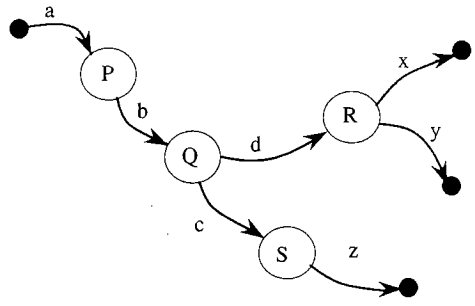


Fig. 5 Example of a dataflow diagram generated from the above structure chart.

6. Experimental Evaluation

The algorithm is applied to the example in [2] to examine its practical applicability and the limitations. The example is the patients monitoring system. Figure 6 shows the structure chart of the system. Figure 7 shows the recovered data flow diagram by the algorithm. In the experiment, subjects were asked to develop data flow diagrams from the structure chart shown in Fig. 6. Subjects were classified into two groups: novice and experienced subjects. Experienced subjects were familiar with the SA/SD methodology. On the other hand, novice subjects had no experience with the SA/SD methodology, although they knew the primitive knowledge of the methodology. The steps for developing data flow diagrams are as follows:
1. Understand the given structure chart
2. Draw a data flow diagram that is consistent with the structure chart
3. Verify the data flow diagram
4. Modify the data flow diagram if necessary

Each subject worked separately to describe their own data flow diagrams. The data flow diagram shown in Fig. 7 was then compared with data flow diagrams developed by subjects. The results of the comparison are shown in Fig. 8 and Fig. 9.

Figure 8 shows that the data flows of the data flow diagrams developed by the experienced subjects are all generated by the algorithm. Novice subjects described the control data, EOF, for reading file, because they confused data flow diagrams with flow charts. This

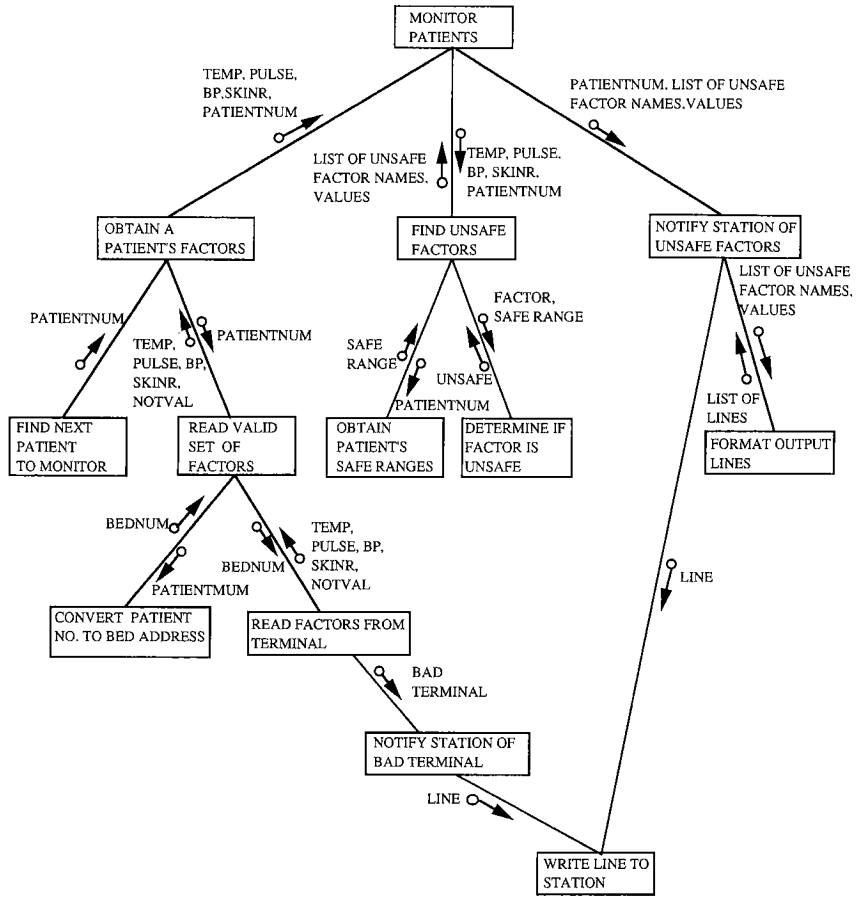


Fig. 6 Structure chart of monitor patients.

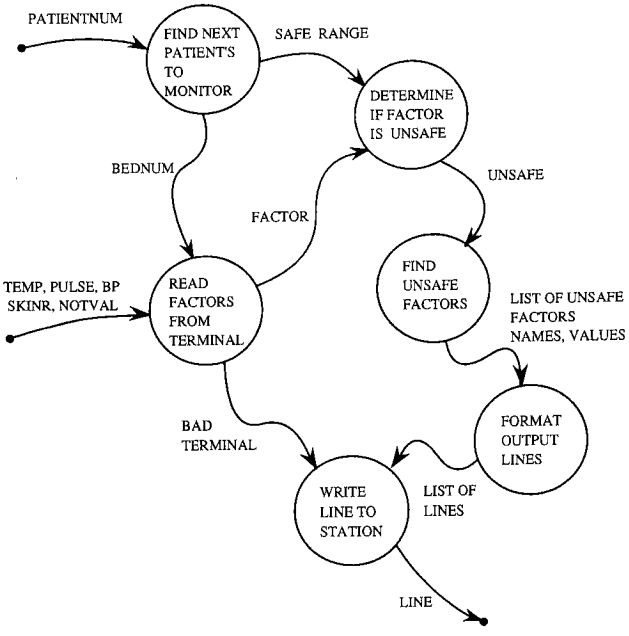


Fig. 7 Data flow diagram of monitor patients.

kind of data that were not extracted by experienced subjects and the algorithm. Novice subjects are apt to describe illegal data flow diagrams. For example, processes that have no input data flow and processes that have no output data flow. The algorithm could prevent these mistakes. This verifies the usefulness of the algorithm from the viewpoint of data flow.

The algorithm does have limitations, however. It generated the formatting data, i.e., LINE and LIST OF LINES and the control data, i.e., NOTVAL and UNSAFE, but experienced subjects did not extract them. This data flows should be removed manually, since the automatic reduction of them is difficult without the semantic dictionary for deciding the necessity of these terms in data flow diagrams.

Figure 9 shows that the processes of the data flow diagrams generated by the algorithm are all developed by the novice subjects. Novice subjects described the control process that were not extracted by experienced subjects and the algorithm. The number of processes of data flow diagram generated by the algorithm is almost half as much as those of modules of structure chart. This verifies the usefulness of the algorithm for reducing implementation level modules.

The algorithm, however, generated the process for

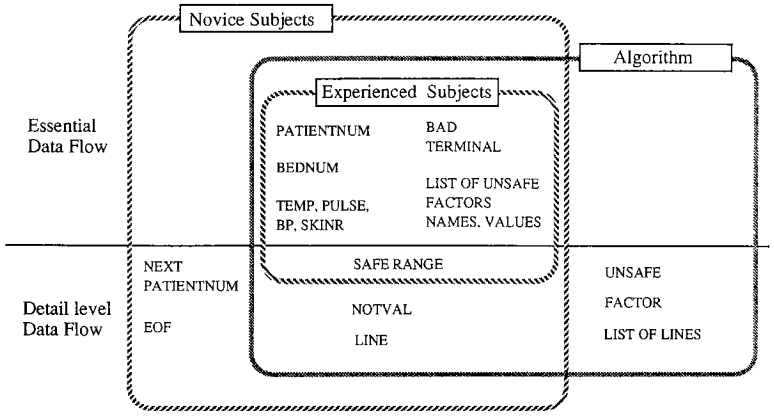


Fig. 8 Data flows developed by subjects and the proposed algorithm.

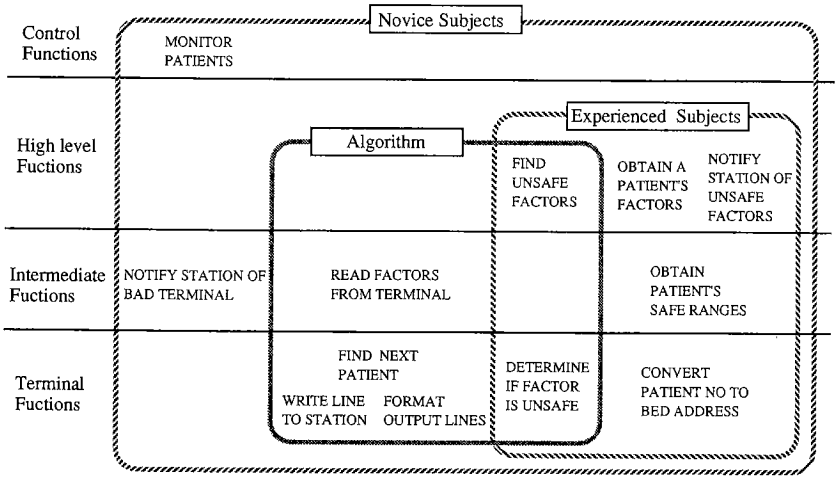


Fig. 9 Processes developed by subjects and the proposed algorithm.

reading and formatting data, but experienced subjects did not extract them. This is because the algorithm could not eliminate detailed level data as noted before. If we remove these detailed level data from the input-output relationship for the structure chart, it will reduce these processes that correspond to terminal modules in structure charts.

For understanding and drawing data flow diagrams, efforts of novice subjects (approximately 52.5 minutes) and experienced subjects (approximately 47.5 minutes) were almost the same. For verifying and modifying data flow diagrams, novice subjects spent more effort (approximately 22.5 minutes) than experienced subjects (approximately 10.0 minutes). The proposed algorithm can reduce the effort for understanding and drawing data flow diagrams for novice as well as experienced analysts. As a result, it can cut 70% through 80% of effort for recovering data flow diagrams from structure charts.

7. Conclusion

We have proposed an algorithm that automatically generates a data flow diagram from the input-output relationship. For large software systems, normative representations of specifications are important means for communication among team members. The proposed algorithm will ease the standardization of data flow diagram representation by producing a minimal data flow diagram. In practical software development projects, few analysts have enough experience with Structured Analysis methodology. Inexperienced analysts tend to develop redundant and erroneous diagrams. This algorithm will resolve this problem, since it automatically generates the minimal and valid data flow diagram from the input-output relationship. Also, we presented an application of the algorithm to the reconstruction of data flow diagrams from structure charts.

Further study of consistency management support

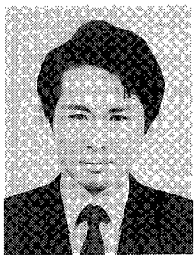
is necessary, including an impact analysis of data flow diagrams and input-output relationship modifications. The integrated support of requirements analysis with other diagrams, such as an entity relationship diagram, is also expected, because multiple kinds of views are needed for requirements analysis.

Acknowledgments

The author would like to thank Ryoichi Hosoya and Hironobu Nagano, who made valuable suggestions, and Ryuuichi Yasuhara for his helpful discussions in the course of examining ideas. He also thanks the subjects, who patiently developed structure charts.

References

[1] M. Page-Jones, "The Practical Guide to Structured System Design," Prentice-Hall/Yourdon Press, 1988.
[2] W. Stevens, G. Myers, and L. Constantine, "Structured design," IBM Systems Journal, vol. 13, no. 2, pp. 115-139, 1974.
[3] B. Benedusi, A. Cimitile, and U. De Carlini, "A reverse engineering methodology to reconstruct hierarchical data flow diagrams for software maintenance," Conf. on Software Maintenance, pp. 180-189, 1989.
[4] E. Byrne, "Software Reverse Engineering: A Case Study," SOFTWARE-PRACTICE AND EXPERIENCE, vol. 21, no. 12, pp. 1349-1364, 1991.
[5] A. B. O'Hare and E. W. Troan, "RE-Analyzer: From source code to structured analysis," IBM Systems Journal, vol. 33, no. 1, pp. 110-130, 1994.



Shuichiro Yamamoto is a senior research engineer, supervisor, of NTT Software Laboratories. Previously, he was engaged in the development of CASE tools. Currently, he directs development of design methodology for client server information systems. His research interests include distributed information systems, end user computing, reengineering, and requirements engineering. Mr. Yamamoto received a B.S. in information engineering from Nagoya Institute of Technology in 1977, and an M.S. in information engineering from Nagoya University in 1979. He is a member of IEEE, IPSJ and JSAI.