TR-071

Design Concept for a Software Development Consultation System

by
M. Sugimoto H. Kato and H. Yoshida
(Fujitsu Limited)

August, 1984

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Design Concept for a Software Development Consultation System

by

M.Sugimoto, H.Kato and H.Yoshida

FUJITSU LIMITED

1015, Kamikodanaka Nakahara-ku

Kawasaki 211, Japan

Abstract One of the main problems in developing and maintaining software is that the level of description required from designers and programmers is too detailed. The Japanese Fifth Generation Computer Systems project plans to implement a software development consultation system as a fundamental solution to this problem. We are considering two approaches: introducing a formal specification language, similar to a natural language; or extending a logic programming language to make it more compatible with the type of thinking processes that software designers and programmers naturally engage in the course of their work. The functions planned for this consultation system include semi-automatic program synthesis, and retrieval and explanation of existing programs, based on the formal specification language.

§1 Introduction

The number of software users is increasing rapidly, in part because the rapid growth in sales of affordable personal computers. This is due to recent advances in hardware, such as microprocessors and 256K RAM chips. Furthermore, there is no doubt that new products will utilize advanced hardware and software technology.

One of the main problems facing this rapidly growing industry is that the task of developing and maintaining software requires particularly detailed accuracy. This work requires the expertise of a considerable number of high-trained individuals.

The Fifth Generation Computer System project has decided to attempt a fundamental solution to this problem. We plan to develop software design tools, a new programming language and other facilities to better control the design process by supporting and enhancing the natural thinking of human beings.

Here, we discuss considerations for a software development consultation system[1)2)].

§2 Software development and maintenance

Designing and implementing software are prime examples of human problem-solving tasks. A working environment that reflects the capabilities of an expert software development team leader is required.

Here, we discuss a model of the human thinking process. As shown in Fig.1, there are roughly two types of human memory: short-term memory (STM) and long-term memory (LTM). It is said that STM can hold 7±2 chunks of information. STM capacity is

small. Also, the duration for which a person can retain such information is, at most, several minutes [3].

The capacity of LTM is large. It takes, however, 100 milliseconds to access the contents of LTM and it takes 5 seconds to store a content into LTM. Therefore, external memory (EM) is needed during the software design, implementation and maintenance phases. In the case of software development, EM corresponds to various type of documentation.

Designers write design documents, which are then used by the program development team. Under current software technology, documentation is the method used to maintain all such specifications.

However, as such documentation increases in volume, it becomes more difficult to incorporate it into the natural thinking process. A human being can grasp only a limited amount of information at any given moment. In this paper, we refer to this limit on visual focus as "foveal". In this sense, no matter how many documents we write, they cannot be used effectively without an efficient method of description and a logical strategy for utilizing the descriptions generated by such a method. As always, clear documentation is a primary requirements.

Today, we write documents using a word processor or a Japanese processing system in a general purpose computer. Therefore, documents stored in computer memory devices are becoming an integral part of the human thought process (Fig.2). In this case also only a single displayed screen can be comprehended by the user at any given time; he can focus only on that portion of the overall data that is directly related to the

problem he is actively engaged in solving.

In addition to conventional documents, program execution results, prompt messages, and error messages are regarded as documents in the broad sense of the term.

In a system in which man and computer interacts, such as a software development consultation system, documents and the representation must always be easy to understand.

Historically, in the process of developing high-level programming languages, steps/man-month, one of the standard units used to evaluate software productivity, have been almost constant. Software productivity has increased in proportion to the descriptive power of the languages employed. Parallels can be drown to the development of human cognitive capabilities.

Unless we develop simple and clear methods of representing and using the information stored in computers, these devices will remain little more than enormously complex black boxes[4].

§3 Documents and specifications for software development

The primary document for software development is called a specification. There are many kind of specifications according to the level of description required, and there are several names in use for the same level. Here, we consider a requirement specification, a functional specification, a detailed specification and a program listing.

In a requirement specification, an outline of the system that we are going to design, implement and maintain is described. This specification becomes the interface between the writer who produce it and the system designers. There are some systems in

use for detecting contradictions in the contents of a description and for converting descriptions to diagrammatic representations, ISDOS and SREM are two such systems.

A functional specification describes the functions in accordance with the requirement specification. This specification is used as the basis for the detailed design. In the process of describing a functional specification, we confirm the function and detect any contradictions.

At present, a functional specification is used for communication among human designers. Therefore, the description format must be easy for *people* to read.

The detailed design specification is either written in natural language (Japanese, in this case) or displayed in diagrammatic form (HIPO, FESDD, YAC, HCP, PAD, flow chart, etc.)[5].

This specification is also used for communication purposes.

§4 Detailed design specification and program

The detailed design specification has been in use for about 30 years. Generally, natural language representation has been used. There has been little change in the level of representation, because specifications have not, until now been processed automatically.

Much interest has recently been focused on logic programming. With the increased availability of languages having this capability, the time has come to reconsider the relationship between detailed design specifications and programs.

Tools to help people understand programs

To understand the contents of a program, tools are needed to supply required information or to convert the program into an equivalent form that is easier to understand.

In Fig.3, the process represented by Conversion-1 converts a kernel language program written by a user into an equivalent form that is easier to understand.

Conversion-2 extracts necessary auxiliary information from a program, or converts it into natural language text for documentation purposes.

The process of program understanding demonstrates the advantage of automatically transforming programs to facilitate quick comprehension.

Transforming executable programs directly from detailed specifications is also a primary goal[9].

To define a detailed specification language for this purpose, the following need further clarification:

(1) The relationship between natural language descriptions and corresponding programs in a logic programming language.

(2) The scope of application of first-order predicate logic.

(3) New techniques being developed through increased experience in logic programming.

The relationship between natural language specification and predicate logic is being investigated from the point of view of first-order predicate logic and its extensions[6].

The above concepts are vital to the development of the type of documentation required for effective software development and maintenance. A software development consultation system is needed to support the design process by making use of such

documentation.

So that the designer may develop software as he wishes, an interactively responsive system is required.

To implement this system, the following items must be considered:

(1) The structure of the knowledge base required for interaction between system and designer.

(2) The hierarchical arrangement of levels of detail in system responses to user queries.

(3) The further definition of an optimal interactive design environment.

Many problems arise in attempting to implement a software development consultation system.

We first started researching the relationship between software descriptions written in natural language and the corresponding programs. We are also collecting data for the investigation of semi-automatic or fully-automatic program synthesis. For this purpose, we are implementing a tool for generating functional explanations in natural language from programs written in a logic programming language. This tool for generating explanatory text can be used not only to collect methodological data, but also during the actual software development for documentation.

§5 Software development consultation system

A software development consultation system must be capable of achieving the following objectives:

(1) The implementation and maintenance of software, in

accordance with the design document

(2) The automatic or semi-automatic synthesis of programs based on the design document

(3) Program verification by interactive simulation.

In the software design process, designers are working on the following:

(1) Design of objects (in the sense of object-oriented programming)

The selection of objects to be created or to be processed.

(2) Description of rules to describe relations between objects

The description of object relations, restriction rules, abbreviation conditions, and so forth.

(3) Explanatory comments

Supplying descriptions of individual process steps.

In implementing a software design consultation system, the following points are also important:

(1) Description of module structure

Module structure must define object units and their locality[7].

(2) Stepwise refinement

Functions based on human cognitive processes are required to support stepwise refinement[8].

(3) Design constructs

Several design constructs are required. A construct is a framework in which designers can describe higher-level functions they wish to implement. Up to now, there has been constructs for data-flow-oriented design, and structured programming for conventional programming languages.

Emphasis must now be placed on constructs for logic programming and the set processing.

Requirement specifications, functional design specifications and detailed design specifications, will be in a form approaching that of natural language. We will be able to maintain the softness of correspondence relations between general descriptions and detailed descriptions through functional concepts.

We concluded from studying possible internal structures for this consultation system, that the difference between natural language specifications and logic language programs is so great that we must fill the gap. We are going to employ a formal language intermediate between natural language and logic programming language. Such a formal language should have following features:

- Object-oriented design
- Both functional and logic programming
- Design constructs
- Various object access paths (data access paths)

We will also adopt the following extended features into our logic programming language:

- Object-orientation
- A combination functional and logic programming
- Module structure

Our approach is shown in Fig.4.

Summary of functions

(1) Semi-automatic software synthesis module

This module semi-automatically generates a kernel language

program from a software specification in either Japanese or English. Because of ambiguities in the meaning of natural language specifications or lack of stored program primitives there will be cases in which a program cannot be synthesized. In this event, designers can interactively inspect and modify the program [10)11)12)].

(2) Function retrieval module

This module searches a module which has a related function as specified in natural language. The function of this module is not necessarily to search a module having exactly the same function, but aims at searching a module set having a similar function. For this purpose, we implement function retrieval using the function concept. As envisioned, this function will be able to be called either from the semi-automatic software synthesis module or directly by the designers. It will also be executed for retrieving objects (data structures).

(3) Function explanation module

The function explanation module explains the program logic of the kernel language program. Several methods were considered for making programs and function explanation text correspond. We selected a method that enhances the man-machine interface and that can also be used for object explanation and function retrieval.

The configuration of software development consultation system module is shown in Fig.5. Users develop software while storing module specifications into a module specification library. Stepwise refinement is made possible by this module

specification library.

The semi-automatic synthesis module generates a kernel language program based on the module specifications.

After a module is synthesized, it is stored in the program library.

(1) Function description support system

The function description support system has an editing function which supports specification in natural language. It also supports stepwise refinement.

(2) Specification analysis module

This module analyzes a specification text written in natural language and converts it to an internal representation.

(3) Program synthesis module

The program synthesis module searches the module specification library using, as a key, a specification analyzed by the analysis module; it then synthesizes a program.

The function retrieval module consists of two parts.

(1) Retrieval module using the "functional concept"

This module searches a module specification library according to the functional concept of the function to be implemented.

(2) Retrieval module using the "object concept"

This module searches a module library for data structures.

The function explanation module explains the logic of the kernel language program in natural language.

(1) Function explanation parameter registration module

This module registers the formats for sentences which then become the frameworks for function explanation text, parameters, parameter types, etc.

(2) Type/mode inference module

The type/mode inference module determines the type of a predicate's parameter or input/output mode by analyzing the kernel language program.

(3) Program reconstruction module

The program reconstruction module rearranges predicates by analyzing the kernel language program for an "easier-to-understand format".

(4) Explanatory text generation module

The explanatory text generation module explains each predicate.

§6 Conclusion

Through consideration of the problems of current software development and maintenance, we discussed design concepts for implementing a software development consultation system. We have described a software development consultation system in which the main modules are a software semi-automatic synthesis module, a function retrieval module and a function explanation module.

We are now proceeding to the detailed design and implementation of a prototype system.

Acknowledgements

This research is being done as one of themes of the Fifth Generation Computer Systems project. The authors would like to

thank the members of ICOT for providing the occasion for this research.

References

1) Sherlis,W.L., Scott,D.S.: "First Steps Towards Inferential Programming", Proc. Information Processing 83, IFIP (1983)

2) Novak,G.S., Ohsuga,S., et.al.: Panel Session PB2: "Knowledge-Based Systems", Proc. 6th International Conference on Software Engineering, IEEE Computer Society Press (1983)

3) Miller,R.: "Human Problem Solving and its Relationship to Computer Aided Engineering Systems", Boeing Commercial Airplane Company (1982)

4) Kanda,Y., Sugimoto,M., Sawai,S.: "Programming in Japanese for Endusers", J. IPSJ, Vol.21, No.3 (1980)[in Japanese]

5) Miyano,T., Sugimoto,M.: "An Approaches to New Generation Software Development System", the Airport Information Symposium '84, ICAA (1984)

6) Ohno,Y., Agusa,K.:"Automated Techniques for Software Development", J. IECE JAPAN, Vol.67, No.1 (1984)[in Japanese]

7) Kobayashi,J., Mizoguchi,R., Toyada,J., Kakusho,O., Isomoto,S.:"An expert system for designing logical structure of database", preprint 33-2, WGAI of IPSJ (1984) [in Japanese]

8) Furukawa,K., Nakajima,R., Yonezawa,A: "Modularization in Logic Programming", ICOT TR-022 (1983)

9) Sakai,S., Ochimizu,K.: "A Text Editor Incorporated with a Function for Creation, Processing and Documentation of a Program Hierarchical Structure", Trans. IPSJ, Vol.23, No.5 (1982)[in Japanese]

10) Yasukawa,H.: "LFG in Prolog - Toward a formal system for

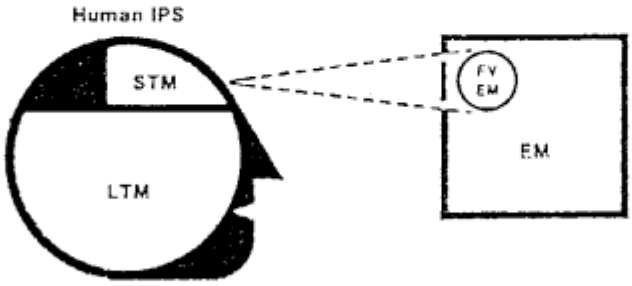
representing grammatical relations", ICOT TR-019 (1983)

11) Abbott,R.J.: "Program Design by Informal English Descriptions", CACM, Vol.26, No.11 (1983)

12) Enomoto,H., Yonezaki,N., Saeki,M.: "Specification language NSL in software development system TELL and its applications", preprint 34-14, WGSE of IPSJ (1984)[in Japanese]

Foveal EM

• Only a part of entire EM



STM – Short Term Memory

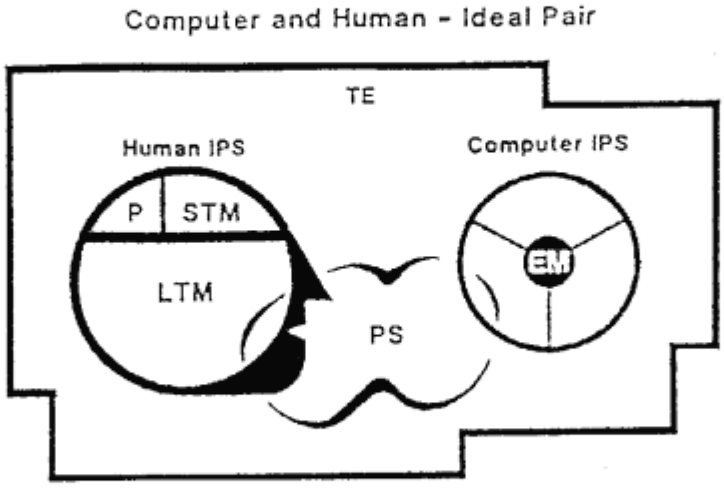
LTM – Long Term Memory

IPS – Information Processing System

EM – External Memory

FV – Foveal View (area of visual focus)

Fig.1 Model of man's thought process[3)]



Clear documentation is very important

in the problem-solving interaction between computer and human.

TE - Task Environment

PS - Problem Space

P - Process

Fig.2 Model of cooperative problem solving[3)]

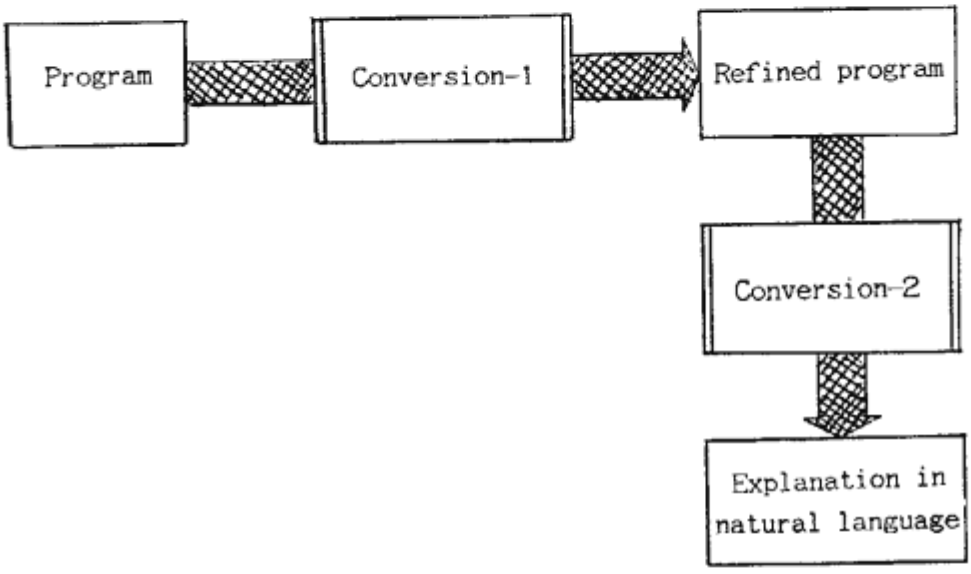


Fig.3 Program understanding

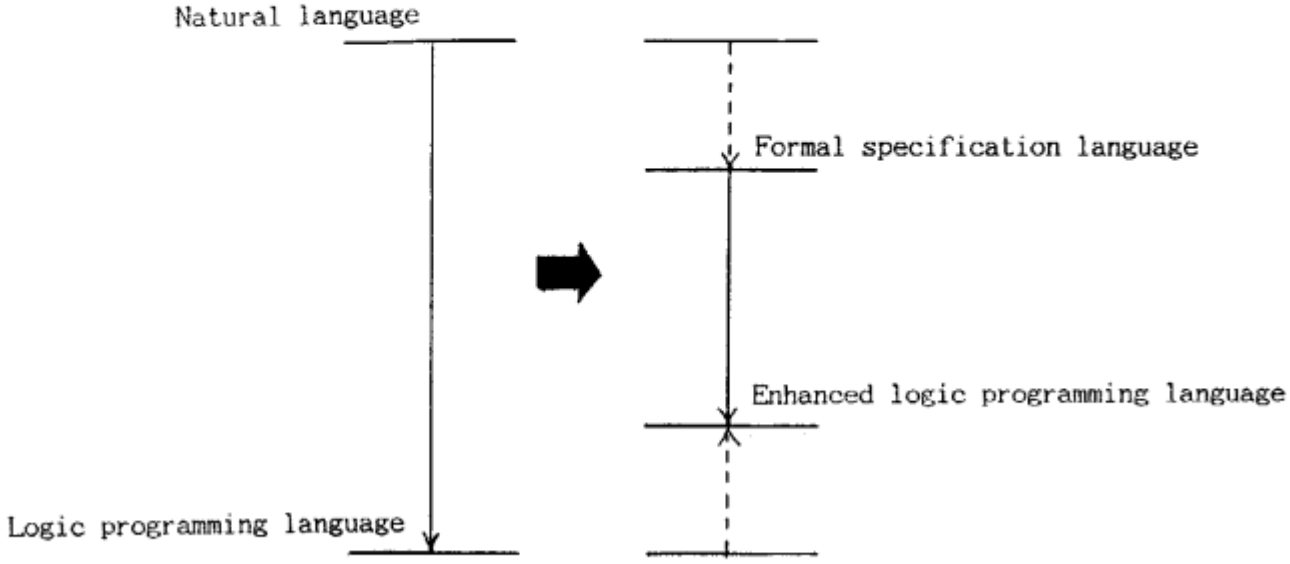


Fig.4 An approach to a software development consultation system

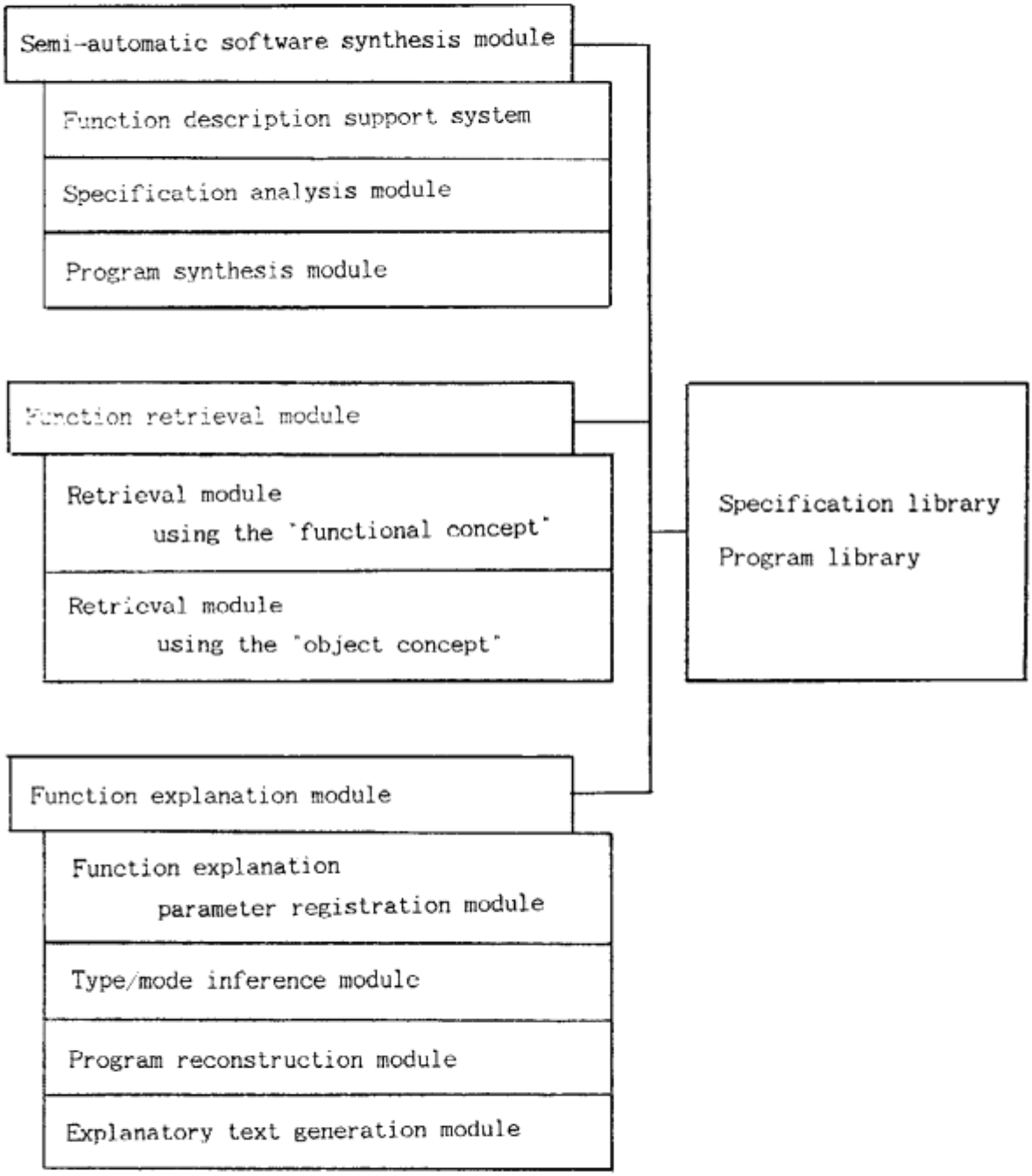


Fig.5 Configuration of the software development consultation system