UDC 681 . 3

Software Maintenance Support Tool Set

Masahiro NOGUCHI*(1) Hiroaki KIMURA*(1)

Abstract:

The software maintenance support tool set developed by authors provides software engineers with the information that helps program maintainability enhancement and repair activities, rendering software maintenance more efficient, which is one of the most important topics to the organizations responsible for information systems in enterprises. This tool set includes program analysis engines based on the most recent compiler technology with our own original algorithms, providing integrated maintenance aid for quality built-in in development, and program comprehension and impact analysis during program repair. Further, the architecture of the tool set is designed such that it can be tailored for multiple procedural languages with relatively small amount of efforts.

1. Introduction

With the "Year 2000 problem" as a trigger, the enhancement of software maintenance efficiency has become a big concern in recent years. To meet this need the authors developed "quality enhancement by static testing" (hereafter called QS) as a tool set to support software maintenance. This paper first describes the basic concept of computer support for software maintenance, and then the main functionalities of QS and its system architecture.

2. Computer Support for Software Maintenance

2.1 Software maintenance

Development cost tends to be the main point of focus regarding software cost. It is known, however, that the lifetime of business applications is about 10 years on average[1], and about 40 to 75 % of their life cycle costs fall into maintenance[2]. Therefore, with the increase of software assets, the budget for maintenance suppresses the development budget in many cases, and the ratio between maintenance and development budgets are said to be 1:1[3] in many of the large enterprises and organizations. Based on this recognition, improvement of software maintenance performance needs to be seriously considered to curtail maintenance cost.

2.2 Computer support for maintenance

Maintenance efficiency can be enhanced by providing maintenance staff with the means to help them understand existing programs and the means to repair programs as correct as possible. Computer support for software maintenance is classified broadly into support for enhancing maintenance quality itself in the development phase and support for repairing programs, which are described in the following sections.

2.2.1 Supporting maintenance quality enhancement

The first step for improving maintenance performance is to enhance the maintainability of the targeted software; that is, making software easier to understand and modify. For these purposes, software needs to be designed in such a way that resulting complexity of programs will be minimized. One reason is that complicated programs force users to think and remember many things at a time in order to understand the program. Another reason is that due to a complex and entangled data flow, repairing just one part influences many other parts, some of which frequently emerges at unexpected locations. As metrics indicating complexity of program, cohesion and coupling[4] showing qualitative complexity, and cyclomatic number[5] indicating quantitative complexity are proposed. Many examples in the literature point out the relationship between these metrics and maintainability, and the relation between these metrics and program decay[3,6-9]. Therefore, mechanically measuring and gathering these values and feeding them back to software development processes allows the improvement of maintainability.

Next, from the viewpoint of enhancing maintenance quality, it is also important to develop programs according to programming rules possessed by software developing organizations. In order to assure maintainability, these rules have usually been created after thorough discussions. Programs developed according to description

*(1) Electronics & Information Systems Div.

rules standardized by an organization help people accustomed to these rules comprehend easily. Conventionally, compliance to the rules has been improved by personnel's actually reading and inspecting the programs. However, many rule violations can be automatically detected in principle. In this point, computer support activities are expected.

As frequent repair deteriorates program quality, it also meaninglessly increases the complexity of programs. One of the main reasons is that source code fragments written by one person can not be fully understood by others, thus the influence of deleting the code on other parts might not be correctly tracked down. In such a case the code to be deleted is often left untouched, and some logic bypassing the code is made to be built-in, resulting in leaving unnecessary code in the program. Support for detecting such redundant code is also important.

2.2.2 Support for program repair work

Major maintenance activity includes elimination of defects inserted in the development stage, partial program improvement, or program modification to suit an operating environment. Its typical process is: the current function of target software is comprehended, the part to actually be modified is extracted, the modification is designed, and finally the program itself is analyzed and actual repair is performed, based on enormous volume involving at least a few thousand pages of design documents. Since a maintenance staff usually differs from a program developer, understanding existing programs involves a heavy work-load. It is often said that more than 50 % of maintenance activities are the process of understanding the program[10].

The first step to understand a program is to refer to its design documents, but the document often differ from what the program represents. Furthermore, no such document exists in some cases, requiring direct reading of the program to understand its contents. Experiments have shown that information on program structures and variables are most often referred to[11]. Therefore, showing this information to maintenance staff in an easy-to-understand style is important for computer support.

Program modification work needs extraction of parts to be modified and their secondary influenced parts from program originally written by a third person; however, manually performing this process is extremely hard. Computers are expected to precisely analyze the dependence among program elements and to show the impact of modification.

3. Software Maintenance Support Tool Set QS

3.1 Overview of QS

QS aims at computer-aided software maintenance based on static analysis, program analysis without executing programs. This tool set provides lifecycle software maintenance support, for quality built-in in development phase to program comprehension and impact analysis in maintenance phase. This tool set provides GUI (refer to 3.3.1) to show users results of analysis in an easy-to-understand way, because the results involves large volume of information which is often mutually related. QS is now available in two versions which realize essentially the same functions, one for C language and the other for FORTRAN. **Fig. 1 to 3** are screen examples, showing parts of analysis results by applying QS for C language to freeware*1 called f2c*2.

3.2 Overview of each tool

3.2.1 Support tool for maintenance quality built-in

Metrics measurement tool

This tool automatically measures the complexity metrics such as cyclomatic number, cohesion, coupling and the size metrics such as lines of code (with or without comments) and the number of tokens. The results will be stored in a file in a certain form, enabling comparison to the quality reference value or statistical analysis results.

Programming rule checker

This tool checks whether or not targeted programs comply with the C programming rules prescribed by Systems R&D Center in Electronics & Information Systems Div., Nippon Steel Corporation. The programming rules include identifier naming rules, declaration rules, type integrity rules, constant number usage rules, operator usage rules, control structure construct rules, data flow rules*3, and commenting rules. The tool checks about 40 of above rules whose compliance can be automatically checked. Data flow shows anomalous execution paths, as well as variables causing data flow anomaly.

This tool can be launched from an editor, making it easy to relate a warning message with a program line where noncompliance is detected. Moreover, help message can be displayed on a WWW browser. **Fig. 1** shows an example.

Redundancy checker

As mentioned before, redundant code is a result of program delay from frequent program modification. The tool is able to detect the execution statements and the input variables, both of which have no effects on output of the procedure to which they belong. Though such code has no effect on execution results, it not only hinders programs comprehensibility and worsens maintainability, but also does it accumulate every time repairs are performed. Extraction of such redundant code by this tool, and its elimination prevent programs from such decay.

3.2.2 Program repair support tool group

Structure analyzer

This tool shows program structures in a simple way as program comprehension aid. It graphically displays call graphs showing interprocedual caller-callee relationship, and flow graphs showing intraprocedural control structures. The tool also displays source code in a source code browser. Since these browsers work cooperatively, relationships among each graph element or source code can be easily obtained by mouse operation. For example, by clicking a procedure node in a call graph, a flow graph and source code of corresponding procedure are displayed. Then, clicking the node in the flow graph causes the color of the corresponding source code fragment to change. Similarly, specifying any part of source code, the corresponding node in the flow graph changes its color.

Since complicated program causes the corresponding graphs to be also complicated, making it hard to comprehend, this tool allows users to change graph layout using a mouse, or to cut out only related segments (for example, select all the procedures called by a procedure). **Fig. 2** shows an example of the structure analyzer screen.

Maintenance document generator

This tool automatically generates hypertext documents of various information obtained through program analysis. The document

*1 Free software: Software delivered free of charge.

*2 ©1990 AT & T Bell Laboratories and Bellcore.

*3 Data flow: Relation between data definition and reference. For example, referencing data without defining the value of them is called "Data flow anomaly".[12]

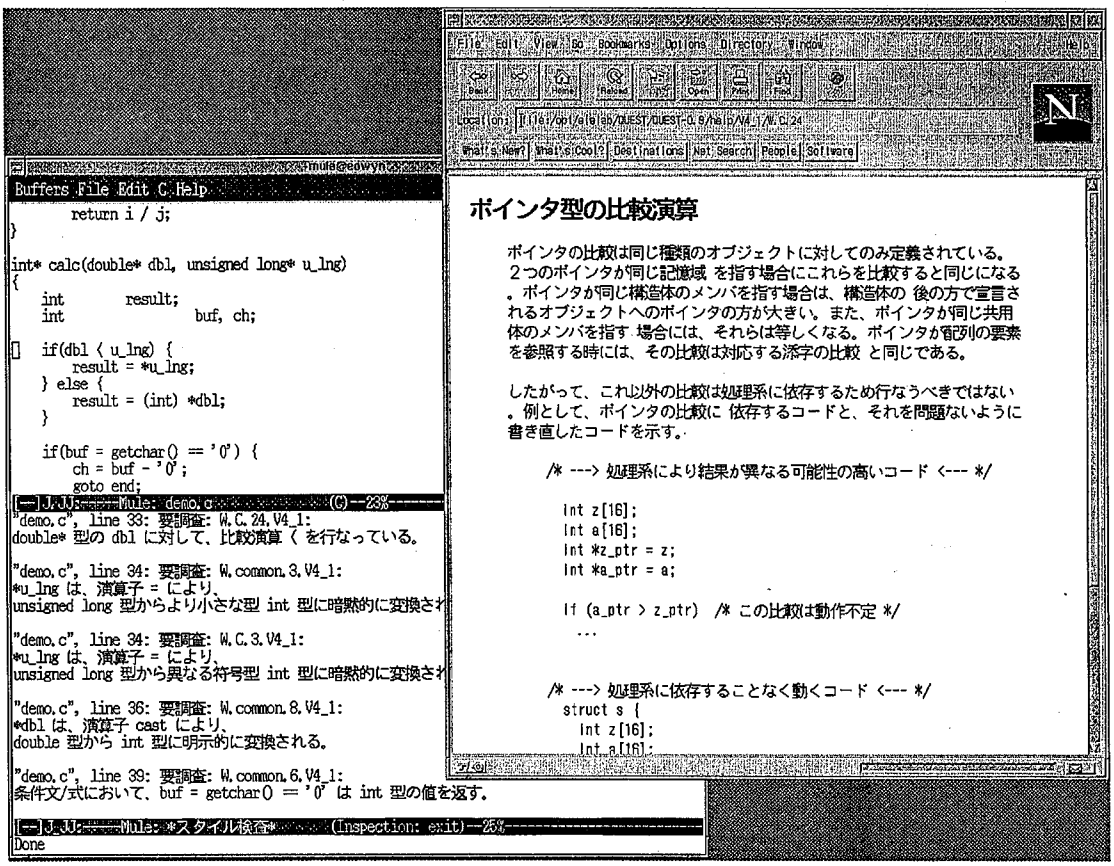


Fig. 1 Example of programming rule checker screen

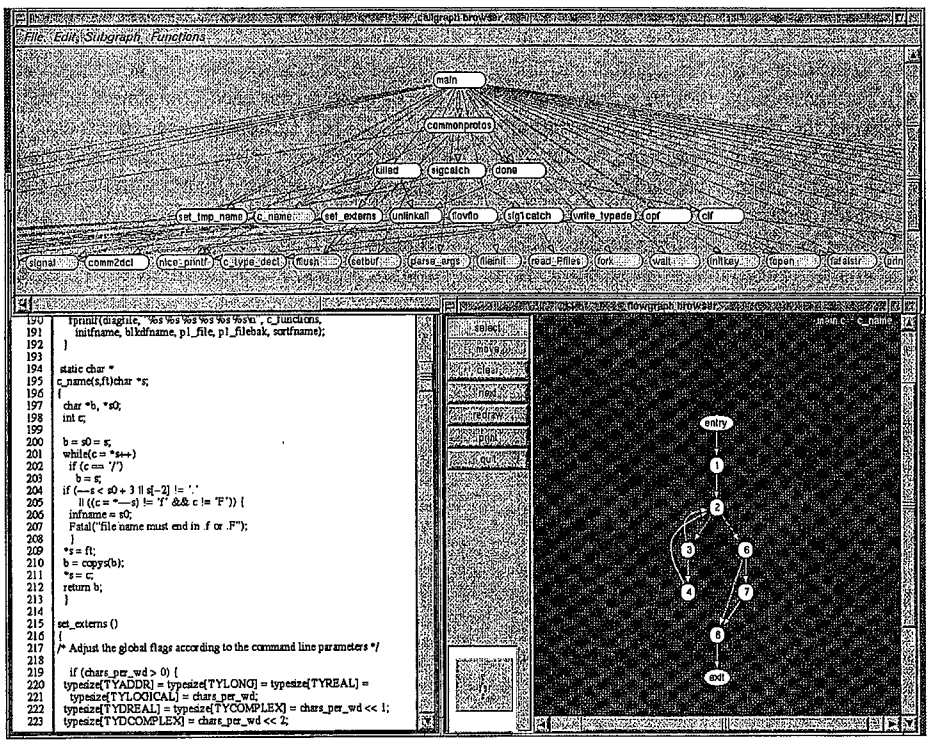
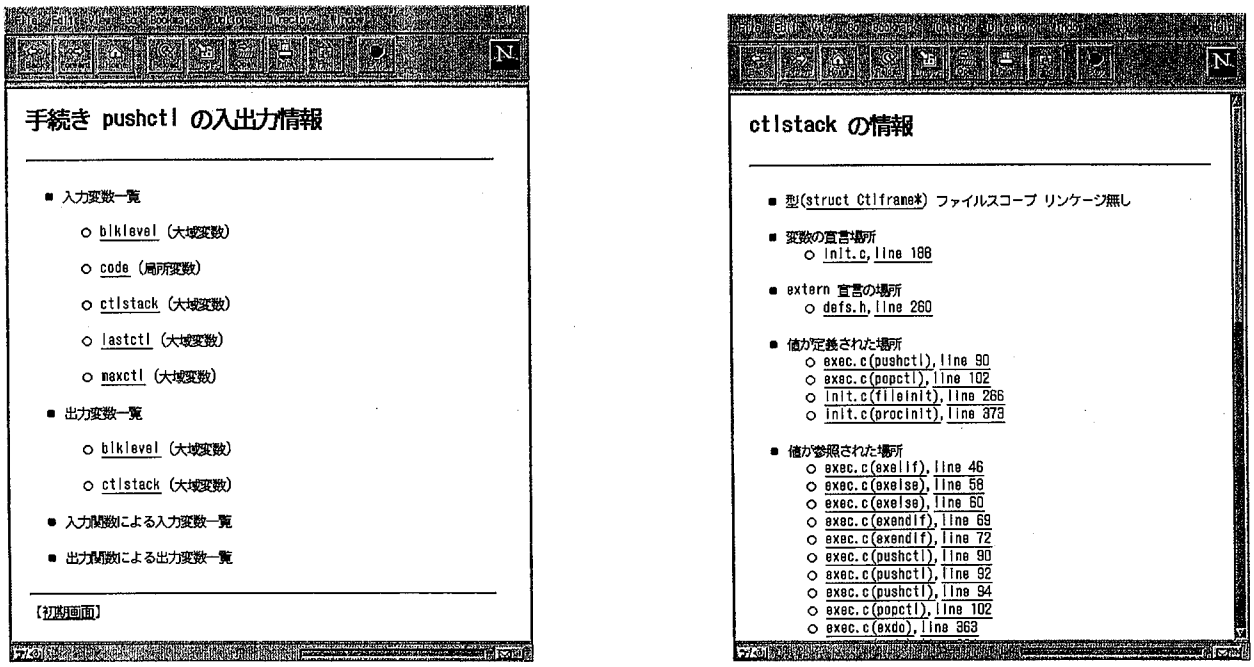


Fig. 2 Example of structure analyzer screen



(a) Example of procedure input/output page

(b) Example of variable information page

Fig. 3 Example of created maintenance document

generated using this tool can be seen on a typical WWW browser, enabling navigation of information on files, procedures, variables and types with mouse operations.

Fig. 3 (a) shows an example of input/output information page of a procedure named "pushctl". Links to the information on the input/output variables (argument and global variable) of the corresponding procedure are indicated by underlining. By clicking "ctlstack", the browser jumps to the page related to a global variable named "ctlstack" (Fig. 3 (b)). Furthermore, by clicking a type named "struct Ctframe"*4, it proceeds to the page linking to the location of the source code at which this type is defined, and to the information on other variables of the same type. Similarly, links anchored as "variable declaration location", "variable definition location", and "variable reference location" list links to the information pages on the corresponding procedures or program texts. Such link navigation provides users with information on how variables are used in the program, giving useful knowledge in repairing the program.

Impact analyzer

When a variable in a program is specified, this tool finds code lines whose execution result is directly or indirectly affected by the value of this variable. It also finds a program fragment which creates the value of the specified variable at the specified line; executing this fragment is guaranteed to produce the same value for the specified variable as when the whole program is executed. This functionality allows sure separation of the program fragments that are impacted by modification and those that are not. It also enables to check if program is correctly repaired by finding program fragments creating output variable values.

3.3 System architecture

QS was developed in C++ language, and runs under SunOS™*5 version 4.1.3 or, greater or 5.5 or greater. To develop QS, an object-oriented application framework was first constructed, then each tool was developed based on the framework. This gives advantages in not only maintainability, but reusability. In QS for C language, for example, about 90%, on tool average, of its source code is reused from the framework, showing how large this effect is. The following sections describe each component of the framework.

3.3.1 GUI (graphical user interface)

InterViews[13], a free software, is adopted as the GUI core library. Using this software, diagram drawing libraries have been developed. A diagram is a grammar-based graph structure consisting of figures and strings, and it is used to visualize complex information. In this library, users can freely define figure types, rules of logical structure such as joints or inclusions among these figure types, and layout rules, and display figures in diagrams. This library is used to draw call graphs and flow graphs.

3.3.2 Communications

Interprocess communication is used for cooperative operations among browsers as described in the structure analyzer section. This cooperation mechanism has the following characteristics.

- Processes have no master/slave relationships and deliver messages one-way.
- No connection is established between sender and receiver processes when communicating.
- Receiver process may not have activated when a message is sent. In this case the receiver process is expected to automatically be activated, initialized, and then receive the corresponding

*4 Logo marks of Netscape, Netscape Navigator, Netscape Communicator are trademarks of Netscape Communications Corporation in the US.

*5 SUNOS™ is a trademark of Sun Microsystems Inc.

message.

- Change of message sender/receiver relationships between processes takes place frequently when developing tools.

To realize this mechanism, LINDA[14], a shared memory type communication model, has been adopted, and then a library implemented with this model has been developed. A class which totally manages the relationship between message types and sender/receiver processes has been added to the library, realizing the automatic activation in the receiver process, and making it easy to change the message sender/receiver relationship.

3.3.3 Analysis function

The core of QS consists of the data structures such as call graphs, flow graphs, program-dependence graphs[15], and the program analysis modules including the data flow analysis[16] and program slicing[17] with alias analysis[18] using state-of-the-art compiler technologies. Furthermore, the core employs our own algorithms aiming at enhancing analysis accuracy and increasing processing speed.

Several procedural languages*6 are currently used in Nippon Steel Corporation. Considering possible future needs for tools like QS, it is desirable to easily modify QS to be applicable to such languages. However, since grammars are different in these languages, so are parse trees*7 generated by corresponding parsers*8. Accordingly, analysis modules, which utilize parse tree information, are required to be developed for the each language.

In QS, parse trees access is abstracted and is provided as a single interface as shown in **Fig. 4**. This mechanism is called "Accessor" and only through it all analysis modules can utilize the syntax analysis information. As a result, the differences in grammars among languages are absorbed in Accessor. This scheme allows QS to handle more than one language without changing the analysis modules. Investigation of several tools*9 for C language utilizing Accessor shows that about 70% of their source code is in common with corresponding FORTRAN tools, and most of the remaining, 30%, is the implementation part of the parser and Accessor.

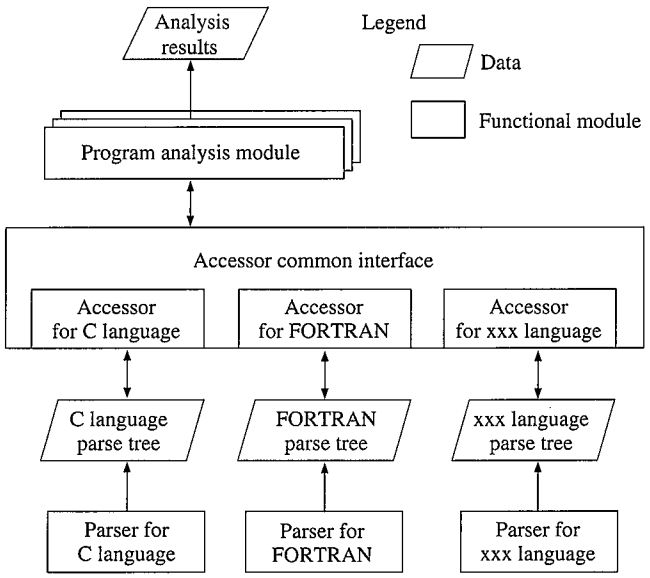


Fig. 4 Architecture easily handling multi-languages

4. Conclusions

This paper provides an overview of QS, a tool set to support maintenance activities. Besides QS enhancing program maintenance quality, it effectively supports maintenance activities by offering appropriate information to maintenance staff. The QS has already been delivered to several divisions of Nippon Steel Corporation, and is being utilized. QS for C language is installed in NSCASE[19]*10, the standard development environment in the process control field of the steelmaking business.

Future plans call for a redesign of the mechanism and development of an algorithm to provide more accurate data flow information, usability improvement, and extend it to handle other languages.

References

1) Tamai,T., Torimitsu, Y.: Proceedings of International Conference on Software Maintenance. 1992-11, p.63-69
2) Vessy, I., Weber, R.: Communications of the ACM.26 (2), 128 (1983)
3) Jones, C.: Assessment and Control of Software Risks. Yourdon Press, 1994
4) Yourdon, E., Constantine, L.L.: Structured Design. Yourdon Press, 1979
5) McCabe, T.: IEEE Transactions on Software Engineering. SE-2(4), 308 (1976)
6) Card, D.N. et al.: Proceedings of 8th International Conference on Software Engineering.1985, p.372-377
7) Korson, T.D.,Vaishnavi, V.K.: Empirical Studies of Programmers.1986-6, p.168-186
8) Troy, D.A.,Zweben, S.H.: The Journal of Systems and Software. 2, 113 (1981)
9) Selby, R.W.,Basili, V.R.: IEEE Transactions on Software Engineering. 17(2), 141 (1991)
10) TAKESHITA,T. : Program Maintenance, Reengineering and Reuse. Kyoritsu Shuppan, 1992
11) Mayrhauser, A. et al.: IEEE Transactions on Software Engineering. 22 (6), 424 (1996)
12) Fosdick, L.D., Osterweil, L.J.: ACM Computing Surveys. 8(3), 305 (1976)
13) Linton, M.A.: InterViews Reference Manual. Stanford University, 1992
14) Gelernter, D.: ACM Transactions on Programming Languages and Systems. 7 (1), 255 (1985)
15) Ferrante, J. et al.: ACM Transactions on Programming Languages and Systems. 9 (3), 319 (1987)
16) Aho, A.V. et al.: Compilers: Principles, Techniques, and Tools. Addison-Wesley, 1986
17) Weiser, M.: IEEE Transactions on Software Engineering. SE-10(4), 352 (1984)
18) Landi, W., Ryder, B.G.: Conference Record of the 18th Annual ACM Symposium on Principles of Programming Languages. 1991, p.93-103
19) Nishida, T.: MID/IEA Joint Meeting by The Institute of Electrical Engineers of Japan, Feb. 1997

*6 Procedural language: A programming language which describes programs using procedures as basic units (for example, functions for C language, and subroutines for FORTRAN.) COBOL, PL/I, FORTRAN, and C language are typical examples.

*7 Parse tree: Analyzing a program written in a programming language by comparing it to its grammar is called a syntax analysis. The data created in this analysis usually has a tree structure, which is called parse tree. This tree also includes symbol tables in this paper.

*8 Parser: In this paper, it denotes a component executing both lexical syntax analysis.

*9 Metrics measurement tool needs to closely refer to the language grammar in its implementation. Therefore, it does not use the Accessor, but directly accesses the parse tree. Such tools are not included in this investigation.

*10 NSCASE is a trademark of Nippon Steel Corporation.