SYS/3 - a Software Development Tool

J. Toczki, F. Kocsis, T. Gyimóthy, G. Dányi, G. Kókai*

Abstract

In this paper we summarize the current state of the development of SYS/3 (SYStematic SYStem Development SYStem), the main features of the system and further research. SYS/3 assists in developing software systems according to a pre-stated standard and assures proper quality of software during its entire life-cycle. A software is considered a set of system configuration items called documents. SYS/3 facilitates writing and checking appropriate documents. Special built-in and external functions generate test cases and Pascal source frames. The major parts of SYS/3 are generated with the SYSGEN system generator system. It makes SYS/3 fairly flexible and easily modifiable. The input specification yields a high level description of the system.

1 Introduction

One of the crucial problems for software developers is the quality of software. There are two methods for providing high quality of a software product. The first is software validation. It means that one proves the correctness of the system using formal mathematical methods. The second is software quality assurance, i.e. one achieves high quality software using both formal and inforaml methods. The result of developing software through standard and consistent methods is a set of deliverables containing all substantial material pertinent to the system. This set of deliverables ensures that the system is correct, i.e. it meets the specification and is efficient, and can be applied as required.

SYS/3 is based on inherently informal software quality assurance, although it contains more or less formal elements as well (for example the T-GEN system, see 5.1). We decided to choose the second approach for two reasons. The first is efficiency. In the case of relatively large real-size projects validation by formal tools is a formidable task with a number of both theoretical and practical problems. The second motivation is that - in practice - we cannot provide a totally formal specification of a system. Usually, the user has mathematically informal ideas and requirements. A more formal description is outlined at a later stage during requirements analysis. On the other hand, a formal validation in itself is not clear to and convincing for an everyday user even if that happens to be totally correct.

The quality of a software product depends on the whole development process. If we intend to assure proper quality we must pay attention right from the specification to the very las steps. The product is viewed as a set of deliverables called documents. SYS/3 works with the following documents: System Specification, Software Plan, Requirements Analysis, Test Specification, Preliminary Design Document, Detailed Design, [1]

*Research Group on the Theory of Automata at the Hungarian Academy of Sciences, Szeged, Aradi vértanúk tere 1, H-6720, Hungary

Program Source. The set of these materials is called system configuration. We say that the system is correct if all the documents are prepared appropriately. It implies that the program itself works as specified but it comprises only a part of its correctness.

The standards and methods applied in SYS/3 are based on [1], partly on [2] and [6]. Some ideas have been adapted from [7].

SYS/3 was started in autumn 1988. The first version was completed in the beginning of 1990. This version is summarized in [8], a complete description is given in [9]. Now we are working out the plan of a new extended version. In sections 2 through 6, we describe features of the first version and briefly mention possible extensions. Section 2 contains the overall description of the system. In further sections, we describe the major functions of SYS/3: project management, document handling and special functions. A description of the SYSGEN generator system is given in section 6. Some basic parts of SYS/3 have been generated using SYSGEN. An outline of future plans is presented in section 7.

2 Overall System Description

We consider software development a team work. The members of the team are users of the system. Users work on several workstations (machines). One of the users is the manager of the project. His/her workstation is the main workstation. Other workstations are ordered in a tree hierarchy. A main library belongs to the project on the main workstation, and each user has a library on the workstations which he or she may use. The manager or other authorized users may define tasks (subtasks) for other users.

The system developed is a set of deliverables, called documents. The documents have both original and backup instances. The system has a built-in configuration management subsystem. It handles documents and document instances. One of the most important functions is version handling. Each version is identified by a version label.

The objects mentioned above, i.e. the project, users, workstations, tasks, documents, document instances, version labels are all recorded in the object table. Documents are physically stored in ASCII files. Besides the object table and document files, the system has a diary. All actions are recorded in theis diary. Records stored in the diary may be listed in a structured format.

From a functional point of view, SYS/3 has three subsystems: project management, control and development. The management subsystem contains the functions used for organizing and managing the project. Information can be accessed from the system diary using a control subsystem. A more detailed description of these subsystems is given in section 3.

The development subsystem contains all functions used for preparing and checking documents, i.e. the functions of configuration management, document handling programs and special functions. The system contains a syntax-directed editor, a parser and a formator program for each type of documents. The document subsystem is described in section 4, section 5 contains an overview of special functions. Figure 1 illustrates the functional structure of the system.

Figure 1.

From a technical viewpoint, the functions of SYS/3 are available through a menu system. The structure of the menu is close to functional structure. On of the central elements of the system is the object table It contains records of the objects. The object table is maintained by the project management and the configuration management routines. Dociments, system files and the diary are handled by a built-in file handler. The editors, document parsers, formators and the special functions are connected to the system through an interface. The user may connect other external elements.

Several routines of SYS/3 are generated from high level descriptions. One of them is the user interface, which was written with UIGEN general user interface generator developed by our research group. UIGEN is capable of generating simple screen handler routines with formed input and ouptut fields as well as help systems, structured file listers, menu windows and usual text (not syntax-directed) editors. In the input description the user only has to define the structure of his/her fields, windows and screens. The routines in the generated user interface units may be used as simple procedures.

SYSGEN, the other generator used for developing the system is more closely related to SYS/3. It has two components that generate structured menu handlers with attributes and logical conditions and document handler routines as syntax-dependent parts of syntax-directed editors, parsers and formators. SYSGEN system is described in section 6. Figure 2. showes physical structure of the system.

Figure 2.

3 Project Management and Control

The project manager or other chief programmers may plan and organize the whole work using the project management subsystem. It can be viewed as a frame of project overview.

The management subsystem handles the following objects: the users, the workstations and the tasks. The users are usually programmers and other members of the team. Each user has his own libraries on the workstations. Users have authority codes. It determines the functions, documents and workstations the user may use. As the aim is to overview the whole process of development, it is important to control the authorities.

SYS/3 is mainly used on single machines, but it also supports development on configurations of several machines. Workstations can be installed in SYS/3. In general, one workstation is one machine, but more logical workstations can be installed on a machine when it is used for processing various subtasks. These are called internal workstations.

Workstations form a tree hierarchy, the root of the tree is the project manager's machine. A workstation may communicate only with its neighbors in the tree. With this structure we assure the consistency of system configuration, although the workstations operate independently. In the first version of SYS/3 communication is maintained through floppy disks. In the future it will be necessary to develop a new network version. There the major tables will be stored locally, only a common coordination table will be located on the server (main) workstation. It allows for relatively independent work.

The most substantial part in the subsystem is subtask handling. A subtask is a logically independent part of the whole work. Authorized users may define subtasks and check the results. These check points are completely informal. The fact that the chief programmer declares a subtask to thave been completed is a merely legal matter. The protocol, of course, contains a record of the tests and other routines used during this process.

In future versions, task scheduling will have to be facilitated with more automatic scheduling, cost estimation and special routines for printing reports. Relations between tasks, documents and system components may be handled in a more structured way.

Another important part of management is controlling the work of the project. SYS/3 has a built-in system diary. Each operation is recorded in this diary. Authorized users may list the diary records and the elements of the object table using functions of the control subsystem. In the first version current status, history of objects and diary records may be listed in a flexible form.

4 Handling Documents

Software development in SYS/3 is considered as preparing documents. The system handles the following documents: System Specification, Software Plan, Reguirements Analysis, Preliminary Design, Detailed Design, Test Specification and Program Source. The documents are stored in files.

SYS/3 has a syntax-directed editor for each document type. This editor ensures that the document is syntactically correct. If the user applies his own editor, formal correctness may be checked with document parsers. Therefore static semantics of documents can be evaluated during parsing even if created with the built- in editor.

Structured and flexible forming routines prepare the documents for printing. In the first version, we attached external printing programs to the system. commands and directives of some printing systems can be used inside the documents.

Syntactic structures of documents are given with $LL(1)$ type contex-free grammars. Static semantic structures are described with one-pass attribute grammars. Test specification, detailed design and Pascal sources are totally formal documents written in the metalanguage of T-GEN, in PDL (Program Design Language) and Turbo Pascal, respectively. In the other documents, informal elements predominate. In further versions, other formal documents will be installed.

4.1 Configuration Management

A large software system usually has a highly complicated structure. Some parts of the system are used independently, but they parts may have elements in common. During the maintainance phase, several releases and versions are formed. Handling backup copies is also significant. When one uses a configuration of workstations, a document may have many instances located on different workstations.

The large amount of document instances and the complicated structure outlined above makes configuration a key issue. The central part of the configuration management subsystem is the table of documents and document instances. This table is a part of the object table It is claimed that all materials connected with the system have to be recorded in this table. This implies that even external programs (as compilers) and DOS commands should be called through SYS/3. With these rigorous rules SYS/3 differes from other configuration management systems (e. g. [10]). Although rigorousness causes technical problems, it is very important from the point of view of safety and quality assurance to ensure as high a degree as possible. From this aspect, SYS/3 may be considered a complete and integrated software developing environment. Some elements of the environment are built into the system, others are externals. The system gives a frame in which work proceeds.

With this introduction, we are ready to describe the structure of configuration management. System configuration is a set of documents. A document has instances. An original instance can have several backups. They together form a version of the document. Each document and each version has its name. Instances can be located on several workstations, and can be copied and sent to other neighboring workstations.

Complete system releases are called configuration version. Their elements are signed with version labels. The structure of system configuration is illustrated in Figure 3. Arrows indicate the direction of searching. System functions appear with their names and double arrows point to the direction of data flow.

Figure 3.

4.2 Syntax-Directed Editors

SYS/3 has built-in syntax-directed editors for each type of documents. The editors are fairly practical and together with other tools for document handling they form an efficient and comfortable environment. The editors are generated with SYSGEN. In the first version, they are defined with $LL(1)$ type contex-free grammars. As described in section 6, SYSGEN is partly based on PROF-LP compiler generator system [4]. PROF-LP is capable of processing LR-type grammars as well. We are now working on a new version of SYSGEN based on extended LR contex-free grammars. As the structure of the editor is closely related to the grammar, LR editors will become more efficient and simpler than LL editors. A syntax- directed editor includes a common editor with usual functions. In this version we apply WordStar-like commands. This part of the editor is generated with the UIGEN user interface generator, therefore the definition can be modified or redefined very easily. Other features of the editor are based on the grammar defining the document. Each document has been defined with a CF grammar; some of the documents have fully informal parts that have been handled by introducing TEXT type nonterminals. If X is TEXT, each string over a set of symbols is derivable from X which does not contain symbols from follow(X).

A nonterminal symbol can be either high level or low level. In the case of high level nonterminals, the editor builds the production tree incrementally. Low level nonterminals (for example parts of an expression) are not represented in the tree. Leaves derived from the last high level nonterminal are stored in a list. Hence low level nonterminals are also referred to as LIST nonterminals. TEXT nonterminals are always high level nonterminals. The production tree is sometimes not complete, i.e. nonterminals may occure in the the input string. There is always an active symbol in the tree. It determines an active area on the editor screen that is the substring derived from the active symbol. The major syntax-dependent functions are listed in Table 1.

Command		Function
E	Edit	Edit the active area and parse the edited string.
P	Prod	Choose a production if a leaf nonterminal is active.
A	Active	Activate the symbol at the cursor.
N	aNcestor	Activate the ancestor of the active symbol.
O	Open	Open a previously closed subtree.
C	Close	Close a subtree on the screen but do not delete it.
D	Delete	Delete the subtree of the active symbol.
M	Move	Move the active area to another leaf nonterminal.
P	coPy	Copy the active area to another leaf nonterminal.

Table 1.

There are some other commands to make the work of the user more comfortable. Most of these commands are the combinations or iterations of the basic commands.

Mention should be made of the context-dependent HELP system connected to the editor. It is generated with SHOWGEN (an extension of UIGEN) and it contains useful information on the active nonterminal and the productions that can be selected.

Using the built-in editors had advantages for beginners and for advanced users as well. Beginners can leam about the structure of documents easily using Prod commands and only the rest of the text has to be typed. Texts - first of all formal ones - can be written in a structured way according to top-down strategy. Using the Close and Open commands one can view both the high level structure and the detailed text of the document.

4.3 Document Parsers

SYS/3 contains a syntactic parser for each document generated by the same grammar as the editor. The parser can be used when the user has written some documents with an external editor. The parsers have a more useful application as well. The parser generator of SYSGEN is an extension of the one-pass subsystem of PROF-LP. PROF-LP processes attribute grammars. Static semantics of documents including semantic conditions and code generator statements are defined with attribute grammars. In this way semantic evaluators for checking the consistency of documents and document translators used in special functions are generated from the descriptions.

PROF-LP processes $LL(1)$ grammars with one-pass attribute grammars and LALR(1) grammars with ordered attribute grammars. We extended the original system with two features. One is that the input string may also contain nonterminal symbols and the strings derived from a TEXT nonterminal X may contain any symbol except for the members of follow(X). In these cases attributes are given their default values. Default value was originally used in PROF-LP when a syntactic error occured. This extension was made to provide compatibility with syntax-directed editors.

The second extension assists in establishing connections among documents on the one hand, and between a document and the menu, on the other. A document or a document instance recorded in the object table is considered as an object with the same attributes as the start symbol of the codument grammar. The attributes of the menu points and objects may be referred to in the menu description and in the document grammars as global attributes. These global attributes are evaluated incrementally when a menu point is entered or a parsing process is completed.

In the first version we ignore side effects arising from dynamic data structures. The procedures implementing the semantic functions must be prepared with the side effects in mind. In further versions some semantic actions may be done incrementally inside the editors. A crucial problem arising here is that of efficiency. In the case of semantically complicated documents it is time consuming to execute all the semantic routines when the input string is changed.

5 Special Funcitons

There are two special functions in the first version of SYS/3. One is the T-Gen test case generator system adapted to SYS/3, the other is the PDL source frame generator routine based on a program design language.

5.1 The Test Case Generator

According to authentic evaluations, during an average software development the cost of testing is about 40-50of the complete development effort. In the case of high reliability software this rate may exceed the 80

The T-GEN system gives an assistance for the functional testing of programs (units) in C, Turbo Pascal and dBase. The standpoints of the testing can be specified in a high-level description. From this description the system generates executable test cases.

T-GEN contains a language for the specification of the process of functional testing. The system is based on the method presented in [3]. In the T-GEN system the original method has been augmented with numerous new features. We discuss these extensions later in this section.

The process of functional testing is based on program specification. That specification - usually in natural language - describes the relationship of the input and output parameters, the environment conditions etc. In functional testing the program can be considered as a black-box. Consequently, in general cases the programs cannot be tested with all possible properties of the input parameters, hence the tester's first task is to define the critical properties of parameters. These critical properties - called categories - are investigated in the testing process. (This testing method is termed as the category-partition method).

Unfortunately, even with the use of categories the number of possible test cases is so great that further restrictions are needed. The categories can be divided intoclasses called choices presuming that the behavior of elements of one choice is identical from the point of view of the test process.

If categories and choices for a program have been defined, then T-GEN is able to generate all the possible test frames. A test frame contains exactly one choice from each category.

In general, there are many superfluous and menaingless frames among the generated test frames. These frames can be eliminated by associating selector expressions withthe choices. A choice can be made in a test frame if the selector expression associated with the choice is true. The selector expressions contain property names. A property name is also associated with a choice and can be considered a logical variable. The value of this variable is true if the given test frame contained that choice.

Similarly to input parameters, the results of a program can also be divided into categories andchoices. Of course, a program usually produces a number of results and thetester must define the 'interesting' test frames. The result categories can also be defined by selector expressions.

Running test cases in applications usually necessitates time-consuming installation of environment parameters. The test frames using the same environment can be divided into test scripts by way of selector expressions.

In the test specification, declarations and executable statements can be written for the implementation of test cases.

The structure of a test specification is described in the following.

```
TEST test-name;
          {global declarations}
PARAMETERS ARE
  CATEGORY category_name:
         {declarations}
         {executable statements}
  choice_name: (selector expression)
         (property names)
         ERROR, SINGLE
         {declarations}
         {executable statements}
  ......
TEST IS
    {executable statements}
SCRIPTS ARE
      script-name: (selector expression)
           {declarations}
           {executable statements}
RESULTS ARE
    CATEGORY category-name:
      choice-name: (selector expression)
          {declarations}
          {executable statements}
```

Of course, several categories, scripts and choices can be defined in a test specification. Only one test frame is generated for the choices denoted by ERROR or SINGLE.

The system presented in [3] investigates only the generation of test frames. The new features implemented in T-GEN (test scripts, result categories, test cases) extend the application possibilities of thecategory- partition method. A more detailed description of the T- GEN system can be found in [5].

5.2 Program Source Frame Generator

During detailed design, the user outlines the structure of program units and procedures more and more exactly. A more detailed version of the design document contains almost all informations stated in a previous version. A structured program source is realy "the most detailed design" of the program.

If we use a formal, but language independent design language, it is possible to execute static semantic evaluators and preliminary source generators. On the other hand, it is convenient for the user to insert some informal parts intothe design document. A first version is usually almost totally informal, but later versions are expected to become more and more formal.

In SYS/3 we use PDL (Program Design Language) which is partly based on the program design language presented in [1]. We extended the original language with structured data types, with some new statement structures and we made technical changes to eliminate inconsistent and inexact parts from the definition. PDL has both formal and informal feauters, it is up to the user to decide the level of formalism. It makes it possible to use a top-down design strategy.

Formal elements of PDL define the most important data types and program structures. Inner parts of a type or a block can be defined exactly or can be described with a text similar to a comment in a program source. Table 2 contains the data types, Table 3 shows the command structures of PDL.

data type	description
SCALAR	simple scalar types (integer, real, logical, etc...)
ARRAY	vector with one index
MATRIX	matrix with two or more indices
LIST	a dynamic list of elements of the same type
CHAIN	double chained list
STACK	stack type list - with a top element
TREE	binary tree
ROW	row type list - with a first and a last element
SET	set of scalar elements
RECORD	structure of fields of various type
FILE	external data set

Table 2.

structure	description
IF	conditional statement with two branches
CASE	conditional statement with more branches
REPETITIONS:	repetition structures
DO	cycle with control expression at the begining of the cycle
FOR	cycle with control variable
REPEAT	cycle with control expression at the end of the cycle
ESCAPE	abnormal ent of a procedure
EXIT	exit from the system
RETURN	normal end of a procedure

Table 3.

From the PDL description of a program, unit or procedure SYS/3 generates a preliminary Turbo Pascal source. Further versions will contain static and dynamic semantic analysers as well.

6 The SYSGEN System Generator

As outlined in the introduction of the paper, SYS/3 is based on a standard software developing method. This method includes the activities to be carried out and the documents to be prepared. The standard depends on the type of software to be developed and even on the local rules of the developing team. To provide flexibility of the system we developed high level metalanguages to desribe the menu structure, the document objects and attribute grammars defining syntactic and semantic structures of documents. SYSGEN generates the menu handling procedures, the object table structures and the programs handling the documents: editors, parsers and formators. The input description consists of two parts. The first part defines the menu structure and the document objects. The menu tree contains points and commands. A command in essence is a special leaf point which may have parameters. Attributes belong to menu points and to commands. A command is defined with a regular expression. Some terminals called tokens can have attributes in these expressions. Dynamic attributes are evaluated incrementally when a point is entered, static attributes are evaluated only once when the point is entered for the first time.

Figure 4.

Figure 5.

Example 1 illustrates possibilities of the menu description. Document descriptions are attribute grammar definitions written in the metalanguage of PROF-LP [4] with the extensions desribed in section 4.2. Document object desriptions are similar to the attribute evaluation and action parts of menu point descriptions shown in Example 1. All user interface units in SYS/3 are generated with the UIGEN user interface generator system. In the UI description, fields and windows can be defined. A field is either an input or an output field. The system handles several types of windows: static for constant texts, menu for seelcting menu points, lister for listing files, help for structured context-dependent help systems, edit for editors. Using SHOW descriptions, complete HELP systems can be described. The MENU generator generates menu windows for a whole menu tree.

```
SYSTEM DESCRIPTION EXAMPLE                    -system name
                                              -global declarations
ATTRIBUTES                                    -attribute declarations
   NAM:STRING STATIC;                         -static attr. evaluated once
   NAM1,NAM2,TMP:STRING TEMP;                 -temporary attr. lost
                                              when exit the menu point
TOKENTYPES                                    -token type definitions
IDENTIFIER:                                   -token type name
   LETTER(LETTER/DIGIT)[1]                    -form of the token (regular exp.)
   HAS NAM                                    -attributes of the token
   DO
     NAM ← SEARCHSTRING(VAL,SMBT);            -attr. evaluations
   END;

MENU DESCRIPTION
NODE ROOT                -root point of the menu
   HAS NAM1,NAM2,TMP                          -attributes of the point

                                              -local declaration
   TOKENTYPES
     STRING:                                  -FORTRAN-like string
       DIGIT 'H' (ANY)
       HAS VAL;

   TOKENS
     NAME:IDENTIFIER;                         -token declarations
     NOTE:STRING;

   NODE POINT1                                -other menu points
   . . .                                      -a point is a block with local declarations
   END

   COMMANDs                                   -command declerations
     COMM1:                                   -command name
```

```
      "COMMAND1"NAME([","NOTE]/ "," "UBUL")
      HAS NAM1                               -command form and attributes
      DO
        NAM1:=NAME.NAM;                      -attribute evaluations
      END;
  TREE                                       -menu subtree description

    POINT1                                   -selectable menu points
    COMMANDS COMM1                           -applicable commands
      DO                                     -attribute evaluations
        NAM1:=X+COMM2.NAM1;
        NAM2:=COMM1.NAM1;
        COMM2.INHER:=0 + x[2];
        POINT1.INHER:=COMM1.NAM1;
    END

    ACTIONS                                  -actions on entering the pont
      CALL INITIALPROC(TMP);                 -procedure call
    END
  END                                        -end of block
END OF SYSTEM DESCRIPTION.
```

7 Experiences and Future Plans

Finally we summarize experiences with the system and present plans for the future. Some plans have already been mentioned in previous sections, now we shall turn to some other more comples possibilities not connected to aspecific part of the system. In contrast with most software engineering systems, our purpose has been to develop a frame system assisting the whole software life- cycle, not just to facilitate one or more steps of the process. The software developed is considered as a complex structure including all documents related to it.

We succeeded in selecting the most important steps of software development and in stating a standard operating procedure including document structures. The other basic conclusion is that the quality assurance approach is extremely useful and efficient in practice.

The first version of system pointed out a trend for further development. Even in this stage, the most important advantage of SYS/3 is the flexibility of the standard. It is a basic direction of development to integrate other flexible features. It contains a more complete semantic description of documents, a high level description of the whole object table and a formal description of some routines called from the menu. With the help of SYSGEN, the user can generate his own SYS/3 system. Of course, a competent knowledge of software engineering is a prerequisite.

The special routines that cannot be generated may form a library in the system. We do not go into details about all possible special routines. It is clear that the PDL routine and the T-GEN system are only a small part of this library. Now let us mention but a couple of possibilities: special project management routines as scheduling and estimating algorithms, graphic design tools, static and dynamic analyzers for design and program sources written in various languages.

It has not been mentioned in previous sections that some parts of the system (the user interface generator, the syntax directed editor generator, the PDL routine or the configuration management subsystem) can be considered as independent software systems with wide fields of application. Actually, T-GEN is an independent system connected solely to SYS/3.

Let us call attention to an effect of software engineering systems like SYS/3. The "heart" of our system is an integrated standard built into it. By using SYSGEN one can state his own standards. Using SYS/3 provides a possibility to manage and control the work of a development team according to a standard. It means a new style of programming where a set of development rules is realised in a computer system. The developers have to adhere to the rules due to continuous control. However, it is not a burden as everyone is free to use his own set of rules. Another feature of SYS/3 is that a software system is considered as the set of all materials. We know that, in general, a complete documentation is not given to the user, but in the case of high safety and quality requirements, it is necessary to know about the overall system configuration.

And now a few words about actual experiences. We found that the system is usable- and convenient in practice. However, there are some technical problems. As the whole system is very large - about 130.000 program lines of Pascal code - organizng memory management and speed of the system is a critical point. We had to face a large amount of technical problems because usual programming tools are not efficient for developing very large systems. Now we have deeper insight into the various ways these problems may be handled.

References

[1] Software Engineering. General Electric Company, Corporate Information Systems, Bridgeport, Connecticut, USA, 1986.

[2] William E. Perry: A Standard for Testing Application Software. Auerbach Publishers, New York, 1988.

[3] Ostrand, Thomas J. and Balcer, Marc J.: The Category-Partition Method for Specifying and Generating Functional Tests. *Communications of ACM,* Vol. 31 (1988), Num.6, 676-686.

[4] PROF-LP User's Manual, Research Group on the Theory of Automata, Szeged, 1986.

[5] T-GEN Reference Manual, Research Group on the Theory of Automata, Szeged, 1990.

[6] System Engineering Development. Preliminary Standard Operating Procedure and Guidelines by Z-MOD Ltd., Pholadelphia, U.S.A. 1988.

[7] I. Sommerwille and R. Thomson: An Approach to Software Evaluation, *The Computer Journal,* Vol. 32. No. 5, 1989, pp. 386-398.

[8] Toczki, J., Gyimthy, T., Kocsis, F.: SYS-DL A System Development Language for Supporting Software Quality Assurance. *Proceedings of the First Finnis-Hungarian Workshop on Programming Languages and Software Tools.* Szeged, 1989.

[9] SYS/3 System Documentation (Beginner's Guide, User's Guide, Reference Guide, Tutorial), Research Group on the Theory of Automata, Szeged, 1989-1990.

[10] PLYTRON Version Control System. User's Manual, POLYTRON Corporation, Beaverton, Oregon, USA, 1987-1988.