A two-step approach to the validation of software engineering methodologies*

by GRUIA-CATALIN ROMAN
Washington University
St. Louis, Missouri

ABSTRACT

A two-step approach to the experimental validation of software development methodologies is proposed. The first stage consists of the evaluation of the programming effort required for the reconstruction of some existing medium-sized program which (1) has known development and maintenance costs and (2) is anticipated to undergo numerous developments in the immediate future. The second step is shown to involve a comparative study of (1) the programs' performance, (2) the degree of expertise required to maintain the two programs, (3) the maintenance costs, and (4) the average adaptation times necessary to integrate a new programmer into the maintenance team.

The approach is demonstrated for a method called program control restructuring that was used in the reconstruction of a complex 10,000 line biochemical simulation system.

THE EXPERIMENTAL ENVIRONMENT

Program control restructuring is a method that was designed to be a useful tool for constructing large programs. It provides good quality design and documentation, expedites the implementation process, improves the readability and flexibility of the program, diminishes the frequency of logical errors, and limits the knowledge needed for making program alterations. Furthermore, program control restructuring was conceived as a method that performs well, especially in those adverse circumstances where the program is not an end in itself, but a research tool. In such conditions the program is under continuous development, only partially specified, and exposed to frequent changes in the design specifications. Moreover, programming is often done by transient personnel, hired on a temporary or part-time basis to accomplish specific developments, or by the researchers themselves, who may have some limited computer science training.

No theory is capable of establishing to what degree program control restructuring is able to control such extreme circumstances as those provided by the environment described above. Only practice can offer the final validation. This is the reason why the project SIMBIOR was conceived—the first practical experiment using program control restructuring. SIMBIOR is comprised of two distinct steps. The first one includes the rewriting (based on the program control restructuring approach) of a large and complex biochemical simulation system which for convenience will be referred to as BSS. The second step involves the study of the maintainability, flexibility, and performance of the new program, called BIOSSIM, over a two-year period characterized by intensive developments and numerous changes in personnel.

That SIMBIOR is indeed a fair trial for program control restructuring may be justified by BIOSSIM's size and complexity but, more importantly, by the fact that the environment in which it was constructed and later used is the very same one described in the beginning of this section. The key to a convincing validation of program control restructuring is to be found in the hardships it had to overcome. From the very beginning of the project, it was clear that BIOSSIM had to be constructed with limited financial and no technical support. While satisfying strict efficiency requirements, it also had to meet demanding deadlines. Moreover, the environment was to have no negative effect upon the quality of the product.

Facing these demanding circumstances was a modest and inexperienced programming team. Its members had never before coded more than several hundred lines; neither had they ever used any systematic approach or well-defined methodology. The situation was made worse by the fact that all of them were employed part-time, a circumstance which could cause a team that required considerable interaction to become very ineffective.

SIMBIOR gained financial support based upon the preliminary evaluation of program control restructuring as a software engineering methodology. A tentative schedule was adopted. It was estimated that the construction of BIOSSIM would take eight months followed by two months of testing and would require a three-person team working fifteen to twenty hours per week. There was also a general agreement that the new version would be less efficient and need more core since it was intended to be machine
independent (BSS was not) with the accent on flexibility and clarity. However, the disadvantages were viewed as insignificant in comparison to the gains.

To validate program control restructuring, SIMBIOR had to achieve the following main goals:

1. a. to adapt the method to the use of FORTRAN and to the specifics of the problem under implementation (to show both generality and adaptability);
   b. to meet the deadlines established for completion of BIOSSIM (to demonstrate high productivity and good utilization of available human resources);
2. to prove the program’s maintainability and flexibility during subsequent developments carried out by transient programmers.

The following sections will describe the test program, the methodology, the two validation steps, and the conclusions of the experiment.

THE TEST PROGRAM

A short description of BSS may help the reader gain a basic appreciation of its complexity and value. BSS was designed to be a biochemical research tool. It is used by a number of biologists and biochemists to build, validate, and experiment with various metabolic models. Its main capability is that of predicting the state (a state is defined by the concentration vector of all chemical species involved in the model) in which the system is to be found at various future points in time. (Time is viewed as being continuous.) BSS accepts a formalized description of a biochemical system, simulates its behavior, and displays its state at certain selected moments in time. Dynamic and state alterations to the system are permitted.

The original version of BSS was written by one person within a year’s time and has undergone much development and expansion for more than ten years, reaching a length of approximately 10,000 cards of highly efficient code. The system, initially conceived on a PDP-6 and later run on a PDP-10 computer, was reorganized two times and transferred to an IBM 360. The first attempt required more than one year during which minor alterations were made and debugging took place. The most recent transfer entailed six months of intensive effort made by a team of two persons. The users became increasingly aware of the inadequacies of the system (high development and maintenance costs) as the demand for new facilities and portability intensified.

Because BIOSSIM is intended to be internationally distributed, it is written in FORTRAN. It is assumed that most installations support this language and most users are likely to have some FORTRAN experience. The language choice is also considered as appropriate from the experimental point of view. Since the methodology is language independent, the use of FORTRAN can emphasize the generality of the method even for those that tend to be very hostile toward languages such as FORTRAN. A good approach must be able to function satisfactorily in any environment even when it is known to work better under certain conditions than others.

THE METHODOLOGY

A complete description of program control restructuring and the motivation behind it may be found in Reference 8. For the purpose of this exposition, it suffices to present at this point only a simplified view of program control restructuring. A control restructured program may be viewed as a set of subprograms organized top-down in a tree-like hierarchy. Each subprogram in turn is represented by a set of procedures or routines organized on two levels. The first level is represented by a control block while the second one contains several modules.

The sole function of the control block is that of calling the various modules that belong to the same subprogram and the control blocks of lower level subprograms. The control block is not permitted to alter any data. However, it is allowed to test the value of a special integer variable called the control variable. The modules, on the other hand, are restricted from calling any other procedures. They perform specific transformations over the data to which they have access. Among the modules of each subprogram, there is a distinguished one called the initialization-documentation module. This module is the first ever to be called by the control block and plays the role of setting all pertinent variables to their initial values. Such a program structure is presented in Figure 1.

While other aspects of program control restructuring will be introduced in the next section as they are needed, it must be pointed out now that the key idea behind program

![Figure 1—The structure of a control restructured program](From the collection of the Computer History Museum (www.computerhistory.org))
control restructuring is that of relying heavily on standardization (based upon language and problem dependent criteria) in order to achieve very high productivity, good quality documentation, and inexpensive development and maintenance.

THE RECONSTRUCTION

This section is dedicated to describing the construction of BIOSSIM, the control restructured version of BSS. As previously stated, this was the first stage of the project SIMBIOR and was intended to demonstrate the impact that program control restructuring has upon the productivity and quality of software production in the context of a most imperfect and demanding environment. At the same time, this section provides a concrete example of the way in which program control restructuring may be adapted to a specific problem and given circumstances. The reader interested in employing this approach will find here very useful guidelines.

Prior to starting the actual design of BIOSSIM, a preliminary study of the problem and environment was conducted in an effort to organize the project and select the appropriate standards for program structure. A short and well-structured weekly meeting was viewed as being necessary for progress reports and handing out programming assignments. However, the only precise and correct communication channel was the documentation itself. Each programmer was expected to work independently and require little or no interaction with anyone else. Regulations regarding the usage of auxiliary storage, updating of listings, and precautionary procedures were also established.

The first step in selecting the project standards was data structuring. The process resulted in the adoption of the following types of variables:

- **Control variables**—as previously presented, they are used by the control blocks to determine the sequencing of modules and may be altered by the modules only.
- **General variables**—they are global variables constant for significant portions of the program and grouped in labeled commons based upon their functions.
- **Local variables**—they are represented by those variables local to a given module with the exception of the do-loop indices.
- **Debugging variables**—they are globally defined flags stored in a labeled common and used to control the activity of the debugging modules.
- **Workspace variables**—they are globally defined arrays (in labeled commons) upon which the transformations are applied. The structure of the workspace variables is standardized.

Standards were also imposed upon the modules. They were chosen in an effort to assure not only low connectivity but also reduced maintenance and development costs, small probability of error, and simplified verifications. The most important standardization rules refer to restrictions regarding access to certain classes of variables by certain types of modules, the preservation of the invariant properties of the workspace variables, and the definition of a maximum complexity for each type of module.

Due to certain specific needs of the program, a number of deviations from program control restructuring were accepted as absolute necessities. The introduction of a group of modules called heterogeneous routines is one concession made to assure good portability of the program. They are not actual modules and therefore may be called from within any module. The heterogeneous routines include specific short machine-dependent sections which could be part of the modules, but the designer felt the need to single them out. Transferring the program from one system to another should require only the rewriting of the heterogeneous routines. Later experiments proved this to be true—the transfer cost was reduced by 80 to 90 percent.

The use of FORTRAN determined not only superficial changes in terminology (e.g., control block became control routine) but also the solutions adopted for a number of aspects defining the implementation standards. A first set of language dependent decisions refer to naming and labeling conventions. One-letter prefixing was selected as the simplest way to provide the programmer with immediate access to the semantics of the named element. Thus, each class of modules or variables has a unique letter by which it is identified in an unambiguous manner (e.g., X2OUT must be a control block on the second level). Furthermore, groups of labels have been associated with specific usages or constructs.

The second type of language restrictions involves the usage of constructs within the modules. These conventions have been designed primarily to assure the generation of intelligible code and to simplify program verification. In order to produce block-structured routines, backward and intersecting GOTO paths have been eliminated. As a result, the computational flow is naturally divided into logical blocks.

The need to subdivide BIOSSIM into a number of subprograms was justified by the complexity of the program, the incomplete nature of the program description, and the instability of the specifications. Being a research and development project, the construction of BIOSSIM required that many key decisions be taken or reversed at times when the program implementation was in an advanced state of completion. The division of the system into subprograms diminished the impact of such circumstances and allowed the designer to perform all necessary adjustments at a low cost and in a very short time. The criteria applied in separating the program into subprograms allowed for the design and implementation of the various subprograms taken as a group to be completed in an arbitrary order, which was by no means top-down or bottom-up. However, each subprogram in and of itself was independently designed and implemented in a strictly top-down fashion.
The only documentation preceding SIMBIOR was the user’s guide, which provided the initial specifications for BIOSSIM. The first element of documentation produced during the course of the project was the introduction to the programmer’s guide. It included a detailed and complete description of the methodology and program structure adopted by SIMBIOR. Subsequently, the programmer’s guide accumulated information regarding the algorithms used, references, debugging facilities, etc.

While maintaining the traditional external documentation sources, program control restructuring places a special accent on developing a powerful internal documentation scheme. Each routine is both an active program segment and a documentation source of a specific character determined by the role it plays in the structure. From the documentation point of view, each control block is a ‘‘conversational dictionary.’’ The conversational dictionaries indicate the information which may be accessed by each module and the way it can use the variables made available. The comments preceding each call also include information regarding the purpose of the call, in what circumstances it is made, and the effect it may have upon the control variable.

Similarly, each documentation module becomes a ‘‘data set dictionary.’’ Among them, the documentation module of the top subprogram is distinguished as the ‘‘main data set dictionary’’ while the others are referred to as the ‘‘regional data set dictionaries.’’ The main data set dictionary contains all the global variables used by the program accompanied by a complete description of such things as their purpose, format, and initial values. In the cases where a variable is not documented at the top level or is used in a particular way by a group of modules, references to the places where the pertinent information exists are included. If an initialization takes place outside the documentation routine, an explanation is provided. Those variables that are not documented in the main data set dictionary appear in one of the regional data set dictionaries.

Regarding module documentation, each module contains, immediately after the necessary declarations, a list of all local variables, their rationale, and their initial values. This enumeration is vital for quick comprehension of the code and rapid checking of the type of variables used and is referred to as ‘‘the pocket dictionary.’’

The duality between program structure and documentation scheme promotes the understanding, in a clear and precise way, of the logic of the program and the use of variables. No assumptions have to be made by the reader at any point in time. All information is there; to reach it requires an effort comparable to finding a word in a dictionary.

The quality of this fully standardized documentation was validated by its use in the design and implementation of the subprograms. By making the program self-explanatory to a large degree, the documentation scheme used by program control restructuring eliminated the need for numerous incomprehensible documents which rarely form a unified documentation system anyway. As a result, it was no surprise that developing the documentation parallel to and as part of the program in a natural and totally interdependent manner registered an unusual success.

The construction of each subprogram always followed the same pattern: first, the control block was designed and its correctness informally established; the implementation of the control block followed; the design and coding of the modules came next; and, finally, the testing and debugging of the subprogram took place. The verification of each subprogram, prior to implementing the control block, was found to be extremely useful both in eliminating design errors and also in preparing correct design specifications for the modules. Although the proofs were carried out in an informal manner, their effectiveness exceeded the expectations. All logical errors were completely eliminated during the verification stage. Furthermore, subprogram verification forced the designer to define the interfaces in a very precise and careful manner—one more factor that contributed to reducing the number of coupling errors to zero.

VALIDATION OF THE RECONSTRUCTION STEP

The distribution of the work responsibilities among the members of a programming team plays an important role in achieving full utilization of the human resources. A concept that proved itself very helpful in reaching this goal was the ideal team notion. The ideal team is a team in which, for each indivisible project responsibility, one person is assigned. The structure of the ideal team may easily be determined by analyzing the processes involved in implementing a specific method. In most circumstances one does not expect to have the resources necessary to form an ideal team. However, by knowing its structure the allocation of responsibilities among the available human resources may be done properly so as to assure the highest possible productivity. Each member of a real team may assume the role of one or more members of the ideal team depending upon his experience and qualifications. On the other hand, the duties of each member of an ideal team should never be assigned to more than one person. If this simple rule is not obeyed, confusion, increased need for communication, and lower productivity will result. An example of an ideal team is the chief programmer team concept, which was devised and tested in its pure form by Mills and Baker. The differences between their approach and program control restructuring are reflected in the structure of the respective ideal teams. The ideal team for program control restructuring requires a chief designer, a documentation secretary, several programmers, and technical support personnel.

The CHIEF DESIGNER needs to have an excellent command of the problem to be solved and a strong structurist orientation. His work duties should consist of the following:

- the selection of the design and implementation standards to be adopted:
- the general design of the system and its correctness proof;
- the writing of control blocks;
- the specification of work assignments for the team;
- synchronization of the work;
- the ultimate decision as to choice of programming solution;
- the supervision of any changes to the documentation;
- design of test cases;
- the testing of the whole system.

The DOCUMENTATION SECRETARY has to take care of the maintenance and updating of

- the module library;
- the documentation modules;
- the external documentation files;
- the manuals and archives;
- the backups;
- the distribution of information regarding any changes in design or documentation to all interested parties.

No request is to be serviced by the secretary unless approved by the chief designer.

The PROGRAMMERS have the duty to code, test, and debug the different modules or subprograms whose specifications are given to them by the chief designer. It is important that they respect all conventions and restrictions upon which the system is based. In order to generate unity of style, they need to be able to adopt easily the programming style of the group. A good understanding of the methodology helps in taking advantage of the available documentation. The programmer is not allowed to use any information or data not indicated in the specifications.

A TECHNICAL SUPPORT TEAM with experience in the kind of work in which the project is involved becomes an important factor in accelerating the work and in decreasing the number of typing and minor syntactic errors.

Material conditions did not permit the employment of the three-person team estimated as being necessary to meet the deadlines set by the schedule. The project had to adapt to the situation by reassigning the work duties of the third person to the two available persons. One of the team members assumed the responsibilities of chief designer and documentation secretary and a small part of the programmers' work load. The remaining programming needs and most of the required technical support were responsibilities of the other person. The total amount of time available was only thirty-five to forty-five hours per week. However, the project was developed on a very reliable time-sharing computer, the PDP-10. A noninteractive machine would require that the work be somewhat differently organized due to the possibility of slow turnaround.

Because of the limited human resources, the modules were not proven correct before coupling them together in a subprogram. Therefore, the need for a well-defined systematic testing procedure appeared to be critical for eliminating any errors existing in the modules. The subprogram testing (theoretically unnecessary if the modules were guaranteed to be correct) was structured as a combined subprogram and module evaluation. Although the modules, when compiled for the first time, were usually full of typing errors, the testing proved to be less expensive than one would expect. An appropriate selection of the modules, the use of invariant properties, the very effective and precise documentation, and the separation of the very simple control from the transformations are some of the reasons that reduced testing costs. Another cost decreasing factor was the fact that the design of each subprogram was concerned with minimizing the number of interfaces. Consequently, the drivers and stubs needed to simulate the subprogram's connections to the rest of the program were few and easy to implement. The testing procedure followed the pattern of the subprogram verification. It was aimed at showing that each transformational routine preserved the invariant properties of the variables involved and efficiently achieved its purpose. The analysis moved sequentially from one routine to the next and attempted to be exhaustive. Special attention was given to those conditions that determined the termination of the subprogram.

The fact that a large portion of the time was spent in designing and coding rather than debugging has to be attributed exclusively to simplicity, clarity of design, and the use of correctness proofs. All the debugging was strictly at the level of a programmer's duties. No errors occurred in coupling the subprograms together. All errors were local and, as it was, none of them required much effort to be corrected.

The means by which the subprogram testing and debugging were performed were the DDT facility available on the PDP-10 computer and BIOSSIM's built-in debugging facilities. DDT provides the programmer with breakpoint, display, and change commands. At the same time, BIOSSIM is able to input a set of debugging flags designed to control the activity of the debugging modules. These modules, built for specific purposes, are stored in a file separated from the program itself. They are called from the control blocks and are permitted to trace or display any variable but prohibited from altering any data belonging to the program. The removal of a debugging module cannot have any side effects. However, the debugging modules may be kept in the program (if properly documented) since their activation occurs only upon the programmer's request.

The control routines and the control variables made debugging simple and inexpensive. Tracing the control variable is the best way to analyze the behavior of the program. By knowing the successive values of the control variable, one can easily determine from looking at the control routine what calls have been made and in what order. Furthermore, the low connectivity among BIOSSIM's modules made the detection of error origin very rapid and eliminated the risk of unpredictable side effects when the correction was performed.

At the beginning of the project, there was little doubt that a better program would be produced. There were, however, questions about the time needed to do it and a general
consensus that the program would suffer a considerable loss of efficiency. Nevertheless, BIOSSIM ran faster and required less core. Furthermore, a construction time of four months by two programmers employed half-time exceeded the expectations of the best experts familiar with the program. The reader is reminded that one person worked one year on the initial design of BSS; since then programs have been added to it during a period of more than ten years. It is reasonable to consider that the intensity of work was similar in both cases. Productivity is the factor that generated the significant differences in time and effort. With a modest team like the one used to build BIOSSIM, one may justifiably submit that the remarkable success (at this point only a reduction of time, effort, and cost) is entirely due to the methodology developed.

MAINTENANCE VALIDATION STEP

Upon the completion of BIOSSIM, the project SIMBIOR entered its second validation stage: development and maintainability tests. The results reported in this section cover the first two years since the completion of BIOSSIM.

The programmers employed to alter BIOSSIM were hired sequentially on a temporary basis to provide assistance with the various developments requested by the biological research team. Only one of them was part of BIOSSIM's programming team; the others had no previous experience with program control restructuring or BIOSSIM. They all were exposed to a two-day training session in which the principles of program control restructuring, the standards, and the documentation scheme were explained. Starting with the third day and from then on, they were given programming assignments of significant difficulty, which they handled with surprising success. Information which could be obtained from the documentation was intentionally withheld from them as a way of forcing them to make use of the documentation scheme. The brief adaptation period was especially surprising since it was shorter than the most optimistic expectations. Experience with BSS showed that the accommodation period varied from one to two months and only after four months could one assign the programmer to make alterations of some major significance. Contrast this with the fact that some programmers who worked productively for BIOSSIM were able to stay with the project only three months! Furthermore, while working on BIOSSIM, it was very rare for programmers to make serious errors, and in no case did their alterations produce any unexpected side effects.

Their work has been considered to be very satisfactory and, indirectly, they evaluated program control restructuring as being equal to the claims it makes. Furthermore, the transient programmers were faced with major developments in the short time they were associated with SIMBIOR. Among the new facilities offered to the user were better on-line and post-simulation scope plotting and an extension to the simulation language allowing one also to include FORTRAN-like statements in which chemical names, fluxes, etc., are used instead of variables. This capability extended the power of the language to a degree never before achieved. Consequently, it allowed the research team to use the language to implement a heart model which previously was produced by tedious hand coding. The price for implementing this so-called SIMFOR capability was only two weeks of work. Another major achievement was a machine-independent compaction of the partial derivatives matrix for the case where the initial sparseness is known in advance. This reduced the differential equation solving time by as much as seventy-five percent. A last surprise was the construction in two weeks, instead of two months, of a very much needed program which finds consistent values for heavily underdetermined environmental inputs.

The complexity of these developments fully probed the flexibility of design introduced by program control restructuring.

The experiment was a success that fully justifies the proposed methodology. BIOSSIM was built as a control restructured, machine-independent program for large distribution. It proved to accommodate rapid and unpredictable developments. The methodology was shown to assure low-cost maintenance and expansion as long as any changes made to the program respect the conventions established when BIOSSIM was initially constructed. Its style and structure cooperate in preserving its entity.

CONCLUSION

The proposed experimental validation procedure reflects the recent changes of attitude among those involved in the development of software—the realization that there are two distinct productivity requirements that must be satisfied by any new methodology. The first one is associated with the construction phase of the system, while the second relates to the maintenance and future development costs. Subsequently, however, the validation itself must satisfy an important acceptance criterion—the experimental environment must be a realistic one. Any experimental validation is relative to the environment in which it was carried out.

REFERENCES