The Software Engineering Laboratory: Objectives

Victor R. Basili
Marvin V. Zelkowitz
Department of Computer Science
University of Maryland
College Park, Md. 20742

I. INTRODUCTION

A great deal of time and money has been and will continue to be spent in developing software. Much effort has gone into the generation of various software development methodologies that are meant to improve both the process and the product ([MYER, 75], [BAKE, 74], [WOLV, 72]). Unfortunately, it has not always been clear what the underlying principles involved in the software development process are and what effect the methodologies have; it is not always clear what constitutes a better product. Thus progress in finding techniques that produce better, cheaper software depends on developing new deeper understandings of good software and the software development process through studying the underlying principles involved in software and the development process. At the same time we must continue to produce software.

To gain a better knowledge of what is good in the current methodologies and what is still needed, and to help understand the underlying principles of the software development process, we must analyze current techniques, understand what we are doing right, understand what we are doing wrong, and understand what we can change.

There are several ways of doing this. One way is to analyze the development process and the product at various stages of development. Unfortunately, such analysis is a tedious process. But it must be performed if we are to gain any real insight into the problems of software development and make improvements in the process. We need to study carefully the effect of various changes in the development process or the product to determine whether or not a particular methodology has any real effect, and more importantly, what kind of effect ([THAY, 76], [WALS, 77]).

This requires measures of all kinds, quantifiable and nonquantifiable. Nonquantifiable measures, although subjective, reveal a great deal of information about the product. We can "see" good design and code that meets the problem requirements in a clear, understandable,

effective way and is easy to modify and maintain in unforeseen circumstances. This kind of understanding is clearly needed, and clearly fruitful; it is accomplished by reading and understanding the design and code. Unfortunately, these judgements are not easy to quantify. They require a great deal of time to analyze and measure each product, or class of products.

A secondary approach is to develop a set of measures that attempt to quantify these qualitative characteristics of good software design and development. Although there is currently no mechanical way of evaluating design, the development of quantitative measures that correlate well with subjective judgements of quality can aid in the understanding and evaluation of the product and process. For example, the "goodness" of a product is related to the time it takes to modify it and the aspects of its organizational structure that permit ease of modification and ease of finding and correcting errors where ease is measured in terms of the time required, number of places code needs to be changed, etc. The "goodness" of the development methodology is related to the "goodness" of the product it produces, e.g., the number and difficulty of finding errors in the product it produces.

It is important to understand what characterizes classes of problems and products, what kinds of problems are encountered and errors made in the development of a particular class of products, whether or not a particular methodology helps in exposing or minimizing the number or effect of a class of errors, what the relationship is between methodology and management control, estimating, etc. A better understanding of the factors that affect the development of software and their interrelationships is required in order to gain better insights into the underlying principles. The Software Engineering Laboratory has been established, in August 1976, at NASA Goddard Space Flight Center in cooperation with the University of Maryland to promote such understanding. The goals of the laboratory are to analyze the software development process and the software produced in order to understand the development process, the software product itself, the effect of various "improvements" on the process with respect to the methodology, and to develop quantitative measures that correlate well with intuitive notions of good software.

The next section gives an overview of the research objectives and experiments being performed at the Laboratory. Section III contains the current list of factors that affect the software development process or product and are to be studied or neutralized. The data collection and data management activities are discussed

in Section IV. The last section contains information on the current status and future plans for the Laboratory. Further details of this project can be found in [BASI, 77].

II. ACTIVITIES

It is clear that many kinds of data can be gathered and analyzed to develop quantitative information about the software process and the product to which it leads. The laboratory has limited funding and personnel and for this reason has limited its scope to studying three very specific areas related to reliability, management, and complexity. It is expected that the scope will eventually expand as we learn more about the collection of valid data and what can be done with it. In this section we discuss the research activities and the two classes of experiments to be run.

Because error-free software is as yet an unattainable goal, the reliability study will provide insight into the nature and causes of software errors. We would like to classify errors, expose techniques that reduce the total number or classes of errors, and detect the effect or lifetime of these errors ([SHOO, 75], [THAY, 76], [ENDR,75], [GANN, 75], [AMOR, 73]). We expect to detect the point at which errors enter the process and the relative costs of finding and fixing them.

Management of the software development process is as poorly understood as the technology involved. We believe that a major effort should be expended on this area. The management aspect of the Software Engineering Laboratory involves the analysis of the management process, the classification of projects from a management point of view and the development of reasonable management measures for estimating time, cost, and productivity ([BAUM, 63], [TAUS, 76]). We will study the effect of various factors, such as time, money, size, computer access, techniques, tools, organization, standards, milestones, etc. We would like to understand at what point in the development process, estimates become reasonably accurate, how one can measure good visibility and management control and under what conditions certain methodologies help provide management control.

Lastly, there is a relation between the development methodology and the product it produces. A good methodology should help produce a less complex product than a "bad" one. We are trying to discover whether the complexity of a software system can be measured by the structure of the resulting programs ([SULL, 73], [HELL, 72], [VANE, 70]). Do various techniques create a more systematic structure, one that is easier to read and

maintain, where data and function are localized with a minimal amount of interaction between modules? The relationship between various complexity measures of program structure will be examined throughout the development process and such measures as error rate, development time, the accuracy and speed of modification will be correlated with these complexity measures.

Two kinds of experiments are being conducted: screening experiments and controlled experiments. In the screening experiments, we are collecting data on a large assortment of projects of varying sizes and types. The impact on the development process is manifested by the requirement that the developers fill out a set of data collection forms (see Section IV). The purpose of the screening experiments is to determine how software is developed now. We are organizing a data bank of information to classify projects for future reference and public availability, analyze what methodologies are being used as opposed to what methodologies are supposed to be used, demonstrate how carefully the actual implementation of a methodology can be monitored, discover what parameters can be validly isolated, expose the parameters that appear to be causing major problems, and discover the appropriate milestones and techniques that show success under certain conditions. While the data collected in the screening experiments may not be complete or totally accurate, it will provide input for the more strictly monitored controlled experiments.

The purpose of the controlled experiments is to discover the effect of various factors on the software development process and product in a reasonably controlled environment. A set of duplicate developments will be performed and detailed data collected for all of them. A carefully chosen set of techniques will be taught to and used by one of the development groups, denoted as the "impacted" group. We will then analyze the effect of the introduced factors by comparing the impacted development process and product in a reasonably controlled environment.

The experiment must be designed in such a way as to insure that we are testing the real hypothesis, i.e., to guarantee that we are measuring what we think we are measuring. It is important that all the contributing factors be well understood and the factors that we are not studying be neutralized [CAMP, 66]. Our approach is first to develop a particular experimental design, analyze its ability to neutralize potential interfering factors, (i.e., individual programmer capability) and perform one experiment. Based on this experience, the design will be modified and experiments repeated until we have arrived at a reasonable standard.

One current experimental design is to have two groups, Group 0 and Group 1, each develop a product, A. We will then impact Group 1 with a set of factors by teaching them the use of certain development techniques. Both groups will then develop a second small project B to give Group 1 some experience with the techniques in an operating environment. Then both groups will develop product C, Group 1 using the new approach. This gives us several points of comparison. We can discover any difference in personnel by comparing project A for both groups; the two groups can then be more honestly compared in project C by factoring out differences from project A. The measures developed for the areas of interest will be used to compare the two processes and products.

In a second controlled experiment, several large scale projects (5 to 10 man years each) are to be carefully monitored with some of the personnel given a training course and set methodology to use. Using the notation above, these will be a set of C projects with no A and B. While the projects are not identical, they are highly similar and should yield information about differences in techniques. In Section V, both of these controlled experiments will be described in greater detail.

III. FACTORS

There are a large number of factors that affect the software development process and software product. Initially, we are interested in a list of potential factors to establish the kind of data that needs to be collected. Next, we are interested in the kinds of factors that we can reliably measure. From this measurable set of factors, we would like to isolate those that appear to have a major impact on the development process and product, i.e., those whose use or non-use show large variation in our measures. Finally, when we have a better understanding of the factors affecting the software development process, we want to quantify them in some way by perturbing them to study their effects or neutralizing them to make sure they are not affecting factors that are under study.

Our procedure is to start with as complete a list of factors and categories of factors as possible. We expect continually to build, iterate, and refine this list through the activities of the laboratory. The development of reporting forms and automated tools have helped define the list of factors that we can isolate. The screening experiments will help further isolate those factors which we can measure and those that appear to be contributing strongly to the various measures associated with errors, complexity of program structure,

management difficulties, etc. The controlled experiments will be used to demonstrate the effect of the various factors that have been shown worth isolated study.

A list of factors is given below, categorized by their association to the problem, the people, the process, the product, the resources, and the tools. Some factors may fit in more than one category but are listed only once.

A. People Factors: These include all the individuals involved in the software development process including managers, analysts, designers, programmers, librarians, etc. People related factors that can affect the development process include: number of people, level of expertise of the individual members, organization of the group, previous experience with the problem, previous experience with the methodology, previous experience with working with other members of the group, ability to communicate, morale of the individuals, and capability of each individual.

B. Problem Factors: The problem is the application or task for which a software system is being developed. Problem related factors include: type of problem (mathematical, database manipulation, etc.), relative newness to state of the art requirements, magnitude of the problem, susceptibility to change, new start or modification of an existing system, final product required, e.g., object code, source, documentation, etc., state of the problem definition, e.g., rough requirements vs. formal specification, importance of the problem, and constraints placed on the solution.

C. Process Factors: The process consists of the particular methodologies, techniques, and standards used in each area of the software development. Process factors include: programming languages, process design languages ([VANL, 76]), specification languages, use of librarian ([BAKE, 75]), walk-throughs ([BAKE, 75]), test plan, code reading, top down design, top down development (stubs), iterative enhancement ([BASI, 76]), chief programmer team ([BAKE, 75]), Chapin charts, HIPO charts ([STAY, 76]), data flow diagrams, reporting mechanisms, structured programming ([MILL, 72], [DAHL, 72]), HOS techniques ([HAMI, 76]), and milestones.

D. Product Factors: The product of a software development effort is the software system itself. Product factors include: deliverables, size in lines of code,

words of memory, etc., efficiency tests, real-time requirements, correctness, portability, structure of control, in-line documentation, structure of data, number of modules, size of modules, connectivity of modules, target machine architecture, and overlay sizes.

E. Resource Factors: The resources are the nonhuman elements allocated and expanded to accomplish the software development. Resource factors include: target machine system, development machine system, development software, deadlines, budget, and response and turnaround times.

F. Tool Factors: The tools, although also a resource factor, are listed separately due to the important impact they have on development. Tools are the various supportive automated aids used during the various phases of the development process. Tool factors include ([REIF, 75], [BOEH, 75], [BROW, 73]): requirements analyzers (e.g., PSL/PSA [TEIC, 77], system design analyzers, source code analyzers (e.g., FACES [RAMA, 74]), database systems (e.g., DOMONIC [DOMO, 75]), PDL processors, automatic flowcharters, automated development libraries, implementation languages, analysis facilities, testing tools ([RAMA, 75], [MILL, 75]), and maintenance tools.

IV. Data Collection

Data collection occurs as four components - reporting forms, interviews, automatic collection of data by computer, and use of automated data analysis routines.

A. Forms: There are seven forms that were defined to obtain information on the factors given in Section III. These forms are filled out by various members of the project development team and are used to gather information at various states of the development process. They reveal the resource estimates at inception, the overall layout of the system, the updating of the estimates and the achievement of milestones, the time spent in various activities, the expenditures of resources, and an audit of all changes to the system. Several redundancy checks have been included to validate the accuracy of the information obtained.

Briefly, the seven forms are as follows (See Appendix 2 of [BASI, 77]):

1. The General Project Summary - This form is used to classify the project and will be used in conjunction with the other reporting forms to

measure the estimated versus actual development progress. It is filled out by the project manager at the beginning of the project, at each major milestone, and at the end. The final report should accurately describe the system development life cycle.

2. The Programmer/Analyst Survey - This form is to classify the background of the personnel on each project. It is filled out once at the start of the project by all personnel.

3. The Component Summary - This form is used to keep track of the components of a system. A component is a piece of the system identified by name or common function (e.g., an entry in a tree chart or baseline diagram for the system at any point in time, or a shared section of data such as a COMMON block). With the information on this form combined with the information on the Component Status Report, the structure and status of the system and its development can be monitored. This form is filled out for each component at the time that the component is defined, at the time it is completed, and at any point in time when a major modification is made. It is filled out by the person responsible for that component.

4. The Component Status Report - This form is used to keep track of the development of each component in the system. The form is turned in at the end of each week and for each component lists the number of hours spent on it. This form is filled out by persons working on the project.

5. The Resource Summary - This form keeps track of the project costs on a weekly basis. It is filled out by the project manager every week of the project duration. It should correlate closely with the component status report.

6. Change Report - The change report form is filled out every time the system changes because of change or error in design, code, specifications or requirements. The form identifies the error, its cause and other facets of the project that are affected.

7. Computer Program Run Analysis - This form is used to monitor the computer activities used in the project. An entry is made every time the computer is used by the person initiating the run.

B. Interviews: Interviews are used to validate the accuracy of the forms and to supplement the information contained on them in areas where it is impossible to expect reasonably accurate information in a form format. In the first case spot check interviews are conducted with individuals filling out the forms to check that they have given correct information as interpreted by an independent observer. This would include agreement about such things as the cause of an error or at what point in the development process the error was caused or detected.

In the second case, interviews will be held to gather information in depth on several management decisions, e.g., why a particular personnel organization was chosen, why a particular set of people was picked, etc. These are the kinds of questions that often require discussion rather than a simple answer on a form.

C. Automatic Data Collection: The easiest and most accurate way to gather information is through an automated system. Throughout the history of the project, more and more emphasis will be placed on the automatic collection of data as we become more aware what data we want to collect, i.e., what data is the most valuable and what data we can or need to get, etc. More energy will be expended in the development or procurement of automatic collection tools as the laboratory continues.

The most basic information gathering device is the program development library. The librarian will automatically record data and alleviate the clerical burden from the manager and the programmers. Copies of the current state of affairs of the development library will be periodically archived to preserve the history of the developing product.

A second technique for gathering data automatically is to analyze the product itself, gathering information about its structure using a program analyzer system. A set of modifications to the FACES system is currently underway and will progress as the laboratory gains more experience. These modifications are geared at getting more of the kind of information about the product required for our measures.

D. Database analysis: The above data collected on

the project will be stored in a computerized database. Data analysis routines are being written to collect derived data from the raw data in the database. The data that is being collected is being processed by a PDP11-based system. For ease of implementation, it utilizes the INGRES relational database system [HELD, 75] which runs under the UNIX operating system.

V. Current Status

Beginning in November, 1976, most new software tasks that were assigned by the Systems Development Section of NASA/GSFC were given the added responsibility of filling out the forms, and thus entered our set of screening experiments. At the present time, about a dozen projects are currently involved. These projects are mostly ground support routines to various spacecraft projects. This consists of attitude orbit determinations, telemetry decommutation and other control functions. The software that is produced generally takes from six months to two years to produce, is written by three to six programmers most of whom are working on several such projects simultaneously, and consists of six man-months to ten man-years of effort. Projects are managed by NASA/GSFC employees and the personnel are either NASA personnel or outside contractors.

In June of 1977, the first of the controlled experiments began. Two teams (0 and 1) are assigned tasks to be designed and developed for delivery to the Systems Development Section. The format of these tasks satisfy the experimental design outlined in Section II.

i.e., $A_0 \quad XB_0 \quad C_0$

$A_1 \quad YB_1 \quad C_1$

where A_i, B_i, and C_i, represent tasks to be developed by team i and X and Y are training sessions. These tasks will be developed on the PDP-11/70 at NASA/GSFC. One team will consist of in-house NASA/GSFC personnel while the other will consist of contractor personnel. The tasks will consist of five separate subtasks with two comprising project 'A', one project 'B', and two comprising project 'C'. All subtasks require somewhere on the order of three man-months of effort.

Team 1 will be given a training session (Y) after completing the A projects, consisting of several techniques: PDL, Structured Programming, Walk-throughs, use of Librarians, Code Reading, and will also be given a small project B to take into account the necessary learning

curve before Project C is undertaken. Team 0 will also be given a training session and a B project, but will not be taught the above techniques.

For this first controlled experiment, there is complete control of the development process. The A projects enable us to determine the background of the personnel and the C projects enable us to determine the effects of the training sessions. The small B task enables us to filter out much of the learning curve involved in learning new techniques. Due to cost considerations, the duplicate developments must necessarily be kept small; however, the projects are large enough to require team interaction among the programmers and therefore we believe that they are generalizable to larger projects. In addition, the techniques taught in the Y training session are those most applicable to team situations.

A second, longer range, controlled experiment was begun in March, 1977. In this case, several similar large scale projects are being carefully monitored. These projects can be summarized by the following table:

Project	Man Years	Techniques Used
1	6	NONE
2	4½	Structured code, Librarian, code reading
3	4½	Training session Y of experiment 1
4	6	Not yet defined

In this case we are performing C-like experiments of controlled task 1. Due to budgetary restrictions, it is not possible to duplicate the development of each, however, the tasks are highly similar and should give us results similar to the strictly monitored controlled task 1. While we realize that we have less control over this experiment, this controlled experiment allows us to study larger projects. By varying the methodology, we expect to observe differences in project progress.

The next stop will be to define controlled experiment 3, based upon the preliminary results of experiments 1 and 2. It is expected that controlled experiment 3 will begin in early 1978. In this case, the techniques taught in training sessions X and Y and used in C, may be changed to reflect the new techniques to be measured. It is expected that as this process continues over several iterations, quantitative data on various products and development processes will result.

ACKNOWLEDGMENTS

The development of this laboratory has involved the efforts of many people, including Robert W. Reiter, David L. Weiss, Howard J. Larsen, Charles L. Wolf, Frank McGarry, Richard des Jardins, Walter Truszkowski, Robert Nelson, and Keiji Tasaki.

REFERENCES

[AMOR, 73] Amory, W., J. A. Clapp, A Software Error Classification Methodology, MTR 2648, Vol. VII, The Mitre Corporation, June, 1973.

[BAKE, 75] Baker, F. T., Structured Programming in a Production Programming Environment. International Conference on Reliable Software, Los Angeles, April, 1975, (Sigplan Notices 10, 6, June 1, 1975, pp. 172-185).

[BASI, 75] Basili, V. R., A. J. Turner, Iterative enhancement: a practical technique for software development, IEEE Transactions on Software Engineering, 1, No. 4, December, 1975, pp. 390-396.

[BASI, 77] Basili, Victor R., Zelkowitz, Marvin J., et al., The Software Engineering Laboratory, University of Maryland Computer Science Technical Report, TR-535, May, 1977, 104 pages.

[BAUM, 63] Baumgartner, J. S., Project Management, Richard D. Irwin, Inc., 1963.

[BOEH, 75] Boehm, B. W., R. K. McClean, D. B. Urfrig, Some Experience Aids to the Design of Large Scale Reliable Software, IEEE Transactions on Software Engineering 1, No. 1, March, 1975, pp. 125-133.

[BROW, 73] Brown, J. R., A. J. De Salvia, D. E. Heine, J. G. Purdy, Automated software assurance, Program Test Methods, Prentice Hall, 1973, pp. 181-203.

[CAMP, 66] Campbell, D. T., J. C. Stanley, Experimental and quasi-experimental designs for research, Chicago, Rand McNally Publishing Co., 1966.

[DAHL, 72] Dahl, O., E. Dijkstra, C. A. R. Hoare, Structured Programming, New York, Academic Press, 1972.

[DOMO, 75] Domonic User Guide, Advanced Technology Group, Data Processing Center, Texas A&M University, 1975.

[ENDR, 75] Endres, A. B., An Analysis of Errors and Their Causes in System Programs, IEEE Transactions on Software Engineering 1, No. 2, June, 1975, pp. 140-149.

[GANN, 75] Gannon, J. D., J. J. Horning, Language Design for Programming Reliability, IEEE Transactions on Software Engineering 1, No. 2, June, 1975, pp. 179-191.

[HAMI, 76] Hamilton, M., S. Zeldin, Higher Order Software - A Methodology for Defining Software, IEEE Transactions on Software Engineering 2, No. 1, March, 1976, pp. 9-32.

[HELD, 75] Held, G., M. Stonebraker, E. Wong, INGRES - A relational data base system, National Computer Conference, 1975, pp. 409-416.

[HELL, 72] Hellerman, L., A Measure of Computational Work, IEEE Transactions on Computers 21, No.5 1972, pp. 439-446.

[MILL, 72] Mills, H. D., Mathematical Foundations for Structured Programming, FSC 72-6012, IBM Corporation, Gaithersburg, Maryland 20760, February, 1972.

[MILL, 75] Miller, E. F., Jr., Methodology for Comprehensive Software Testing, Interim Report, Rome Air Development Center, RADC-TR-75-161, June, 1975, AD# A013111.

[MYER, 75] Myers, G., Software Reliability Through Composite Design, New York, Mason Charter, 1975.

[RAMA, 74] Ramamoorthy, C. V., S. F. Ho, FORTRAN automatic code evaluation system (FACES), part I. Memorandum No. ERL-M-466, Electronics Research Laboratory, University of California, Berkeley, August, 1974.

[RAME, 75] Ramamoorthy, C. V., S. B. F. Ho, Testing Large Software with Automated Software Evaluation Systems, IEEE Transactions on Software 1, No. 1, March, 1975, pp. 46-58.

[REIF, 75] Reifer, D. J., "Automated Aids for Reliable Software," An Invited Tutorial at the 1975 International Conference on Reliable Software, 21-23 April 1975.

[SHOO, 75] Shooman, M. L., M. I. Bolsky, "Types, Distribution, and Test and Correction Times for Programming Errors," Proceedings 1975 Conference on Reliable Software, April 21-23, 1975, pp. 347-362.

[STAY, 76] Stay, J. F., HIPO and integrated program design, IBM Systems Journal 15, No. 2, 1976, pp. 143-154.

[SULL, 73] Sullivan, J. E., Measuring the complexity of computer software, Mitre Corp. Report MTR-2648, Vol. V, June, 1973.

[TAUS, 76] Tausworthe, R. C., Standard Development of Computer Software, Part 1 Methods, Jet Propulsion Lab, Calif. Inst. of Technology, Pasadena, Calif., July, 1976.

[TEIC, 77] Teichroew, D., E. Hershy, PSL/PSA: A Computer-aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE Transactions Software Engineering 3, No. 1, January, 1977, pp. 41-48.

[THAY, 76] Thayer, T. et al., Software reliability study, TRW Defense and Space Systems Group, National Technical Information Services AD-A030-798, August, 1976.

[VANE, 70] Van Emden, M. H., The hierarchial decomposition of complexity, Machine Intelligence 5, 1970, pp. 361-380.

[VANL, 76] Van Leer, P., Top-down development using a program design language, IBM Systems Journal 15, No. 2, 1976, pp. 155-170.

[WALS, 77] Walston, C. E., C. P. Felix, A method of programming measurment and estimation, IBM Systems Journal, No. 1, 1977, pp. 54-73.

[WOLV, 72] Wolverton, R. W., The Cost of Developing Large Scale Software, TRW Software Series TRW-SS-73-01, March, 1972.

Research supported in part by grant NSG-5123 from the National Aeronautics and Space Administration to the University of Maryland.