THE SOFTWARE ENGINEERING LABORATORY: EXPERIMENTAL DESIGN AND PROCEDURES

Progress Report 1977*

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

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

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 facto-s 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 our current list of factors that affect the software development process or product and are to be studied or neutralized. Section IV discusses one of the types of experiments being run and the last section discusses the data collection and management activities.

II. OVERVIEW OF THE LABORATORY 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, the classes of experiments to be run, and a brief overview of the data activities.

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.

Projects are being studied at three levels of control and detail: screening experiments, semi-control experiments, and control experiments. The screening experiments involve the collection of data on the development of several ground support systems used to control spacecraft operations. Their purpose is to determine how software is being developed now, supply data for a data bank of information to classify projects for future reference and public availability, expose the differences between the theoretical application of a methodology and its practical implementation, 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.

The semi-controlled experiments involve the collection of data on several similar attitude determination systems using prespecified development methodologies. Their purpose is to extend the screening experiments by permitting tighter control and to permit the monitoring of different development methodologies, comparing the various factors affected by the different methodologies.

The controlled experiments involve the development of duplicate programs analysis and data base systems with fairly rigid control of many of the factors affecting software development. The purpose here is to design usable experiments for various development factor evaluation that can yield statistically valid results.

The data activities of the Laboratory include data collection, data processing, data evaluation, and courses on methodology. Data collection activities include forms filled out at various points in the development, covering various aspects of the product; interviews used to validate the forms and collect information not easily filled out on a form; and automatic collection of data via the existing system and program development library when available. The data processing activity involves entering the data into a computerized data base and screening for accuracy. The data evaluation activity involves the analysis of the data collected with emphasis on management, reliability, and complexity. This analysis deals with simple raw statistics, in the forms of counts, sums and averages, derived statistics in the form of correlations and multivariate analysis of the relationships between various factors, and the development of measures based on theoretical models of program behavior and complexity. The measures will be computed from combinations of the raw statistics and derived statistics.

The simple raw statistics include project attributes such as type of processing, instruction size, code size, methodologies and techniques used, resources, total man hours, calendar time, types of computer access, automated aids and tools, usable items from previous projects, milestones, etc. Also included are counts of total number of program changes, total errors due to various factors, such as misinterpretation of the requirements or specification, total time spent in each phase of development, errors found using various techniques, total computer usage, etc.

Derived statistics are used to find relationships among various factors. These would include the relationship between the number of errors and program size, between errors plotted against time in development, between various estimates (e.g., cost, time) which have been reevaluated at various points in the development, and actuals at the end of the project, noting points at which the estimates become realistic, and between errors distributed according to the amount of time they remained in the system, etc.

At the other end of the spectrum, measures are being developed that are associated with different models of program behavior and complexity which are being tested against the statistics calculated above. For example, various measures derived from models of complexity will be compared with development time, errors and subclassification of errors, subjective views of the complexity of various modules, etc., to determine if these measures and models can be validated in a real environment. Measure of complexity include the number and significance of control paths and data bindings based on various criteria for hierarchical decomposition of the program into elementary sub-schemes using different formulas to combine their individual evaluations ([SULL, 73], [LING, 77]).

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 nonuse 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. <u>Problem Factors</u>: 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. <u>Process Factors</u>: 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]), structured flowcharts, hierarchical input/output charts ([STAY, 76]), data flow diagrams, reporting mechanisms, structured programming ([MILL, 72], [DAHL, 72]), techniques ([HAMI, 76]), and milestones.

D. <u>Product Factors</u>: 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. THE CONTROLLED EXPERIMENT

This section deals predominantly with the purpose, design and problems involved in organizing a controlled experiment. The purpose of the controlled experiment is to isolate the effect of specific factors, in our case mostly software development techniques, on the development process. The idea is to design as airtight as possible a valid, usable experimental paradigm for methodology evaluation. Clearly, regardless of the statistical significance of the results of one experiment, one needs to run several such experiments to accumulate substantial confidence in the interpolations made from those results.

The current design involves the use of two programming teams, both assigned the same project tasks. Both groups, the experimental (1) and control (2) will be treated in identical fashion as much as possible, with the sole systematic exception being the experimental treatment. The experimental group will be assigned a task A which will be used as an indicator of their "normal" behavior. They will then be trained in a specific methodology. The methodology will be reinforced in a session analyzing the development task A but in the new methodology. They will then be given a second task B in which to practice the methodology. They will then be assigned task C with the specification that they use the newly learned and practiced techniques. In parallel, a control group, group 2, will also be given task A, followed by session analyzing the development of A using the methodology performed on A, followed by task B, followed by task C. The difference is that group 2 has no training session.

This design gives us several points of comparison. We can discover any differences in the capability of personnel by comparing data from project A for both groups. The two groups can then be more honestly compared on project C by statistically controlling for any differences observed in project A. The design was developed to minimize several problems which often jeopardize the validity of experimental designs ([CAMP, 66]).

The experimental group and control group will each consist of three programmers, working approximately half time on each of the tasks assigned. This is the normal operating environment at Goddard; most programmers work on more than one task at a time. Task A is about a three or four man month effort, with a deadline of about two calendar months. It is necessary to have a minimum of three man months to make the project amenable to a three man effort. At the same time, since the project is to be duplicated and the design is still experimental, it is difficult to argue for too large a project because of cost.

After task A has been completed, the experimental group will undergo training in a particular set of structured design and programming techniques. The training will consist predominantly of a one week course which will cover the topics shown in the following outline:

Day 1:	Introduction, functional model of structured programming, process design language, reading structured code, laboratory exercises in the given techniques.
Day 2:	Management and estimating problems, proving correctness of structured design, writing structured programs, laboratory exercises.
Day 3:	Documentation, stepwise refinement, stepwise reorganization, laboratory exercises.
Day 4:	Organizing for structured programming, modularization, top down development, stubs, program development libraries, walk throughs, laboratory exercises.
Day 5:	Project control, iterative enhancement, case study analysis, laboratory exercises.

This course is offered on a continuing basis and the experimental group will be members of a larger class of approximately 25 people. This approach was chosen to help minimize any special group interaction effect on the experimental group.

In order to help insure that the group understands the new methodology, there will be a one or two day special training session with the group in which the design of the code analyzer from task A will be reanalyzed within the framework of the newly learned methodology. To compensate for this special group activity and design analysis instruction the control group will be exposed to a similar experience. Even though they will not be given a course, they will have a similar one or two day session in which they will review their design of the code analyzer with respect to their particular design methodology.

Clearly, it takes time and practice to achieve a reasonable amount of skill in the use of a new set of techniques. It is difficult for us to estimate how long it will take for the programmers in the experimental group to acquire a sufficient amount of skill to show an improvement, if there is going to be one, over the previously acquired technique skills, before we can attempt to test the use of this new skill. To compensate at least partly for this uncertainty, task B will be assigned. Its main purpose is to give the experimental group experience with the newly learned methodology. Task B consists of a one to two man month effort. Both groups will perform task B. Task C will be used to perform the comparison between the two groups. It also consists of one or two three man month systems, most likely ground support systems or subsystems.

A major advantage of this design is that it permits us to analyze the differences between the two groups with regard to task A, working in their own self-defined environment. Measures on task C can then be statistically adjusted by performance on task A.

During the entire duration of the control experiments, regular meetings will be held with each of the groups to answer any of their questions on how the reporting forms should be filled out and to gather extra information that is either not clearly specified on the form or was not obtainable from the forms at all. These interviews will also be used to help validate the information that is on the filled out forms. To help eliminate any biasing here, all interview questions will be written down and the same questions will be asked of both groups.

There are several problems that arise in trying to implement the design in a "real world" environment. A couple of these will be discussed. One problem that arises in the development of real large scale projects is that the specifications are not always well defined. In this case, a project monitor, who is not a member of the group, will be used to answer questions about the specifications and make decisions about what is really meant. There will be two different project monitors, one for each group. This is to help guarantee that any information learned by the monitor will not be passed on unconsciously to the other group. When a question arises about interpretation, raised by one of the groups, the contract monitor will consult with the other contract monitor to see if it has been resolved; if not, he will resolve it. If it has been resolved, he will be told how it was resolved and then give a resolution to his group. Thus, if the methodology used by one group helps them understand where the specifications are unclear earlier in the development the other group will not benefit from this information. The only problem here is that the two groups may develop slightly different systems. In this case, we will go to outside arbitrators and to the project monitors to determine that both systems meet user requirements satisfactorily.

There is also the problem of keeping the experiment secret. That is, we do not want either group to know that the other is doing the exact same task and that they are being compared on certain measures of management control, reliability and the complexity of the final project. For this reason, we selected programmers from isolated, but hopefully similar groups. One group consists of programmers employed by a contractor. The other group is internal to NASA. They have both been told that they are being carefully monitored because we are interested in the effect of the forms and whether they can be filled out accurately. They know another group is also doing a "similar" set of tasks and our interest is in whether the forms get filled out in the same way.

From what has been said, it is clear that it is difficult to design a controlled experiment in a real environment. There are too many factors that cannot be measured accurately or completely controlled, such as programmer ability. Designs must be devised that minimize these problems. It is difficult to implement a design since it is difficult to control all the factors in a real environment; e.g., contracting software so that it be performed in a certain way, guaranteeing that the same people will be available throughout the experiment.

However, the authors believe that it is important to perform such studies, creating the most controlled environment possible, if we are ever to gain any real insights into the effects of various software methodologies on the development process and the resulting product. We feel the design proposed in this paper is a reasonable first attempt at approximating an effective experimental design in the given environment. It is clear that much will be learned about the development of program development experiments and it is hoped that a great deal will be learned about the development process. We hope to continue to report on the progress, problems, errors and successes of running controlled experiments in this area and hope that we can gain from the efforts of others as well.

V. OPERATIONAL ASPECTS OF THE LABORATORY

After being in operation for a year, we are now starting to process the data necessary to perform these evaluations from about a dozen NASA projects. From the experience gained by collecting data from these projects, our data gathering process has evolved into a more effective operation.

As NASA software is developed, auxilliary data for the laboratory are produced which passes through four distinct phases: project development, data collection, data processing and data evaluation. This section describes the organization of the laboratory and how we have handled some of these operational aspects.

NASA/GSFC launches approximately 25 unmanned satellites per year and most of these include the development of ground support software requiring about six man years of effort. These software projects generally take about a year and involve from 8 to 15 people, most of whom are working on several such projects simultaneously. The projects are supervised by NASA/GSFC personnel with most of the actual development by employees of an outside contractor.

NASA and contractor management were eager to participate in this study, but were concerned that the impact on current development schedules and costs be minimal. In order to minimize this impact, we decided early to use a set of reporting forms as the basic data gathering mechanism. It was hoped that the forms could be filled out easily, would not interfere with current techniques, and would not involve much overhead, yet would still give an accurate picture of development progress.

The laboratory was organized in August 1976 via a research grant to the University of Maryland. Currently, personnel include 5 employees of NASA/GSFC, 2 faculty and 9 students at the University, and approximately 75 programmers employed by the contractor on the projects that we are monitoring. The two major tasks of the laboratory are project management and data evaluation. The day-to-operational aspects currently are broken down into four areas:

1. NASA/GSFC personnel are the interface between the contractor and the laboratory. They are responsible for project development and for contractor compliance with laboratory requirements. They are supervising spacecraft projects as part of their normal workload in their normal environment.
2. The main research responsibilities are being performed by the authors of this report along with several graduate students. In addition, some of the NASA personnel are involved in this activity.
3. In order to evaluate the collected data, laboratory software for a PDP11-based system is being developed. This is being performed by two student programmers under faculty supervision.
4. The day-to-day data collection is currently being handled by one NASA employee and four university undergraduate students under the supervision of a graduate student.

Given this structure, data flows through the laboratory as outlined by Figure 1:

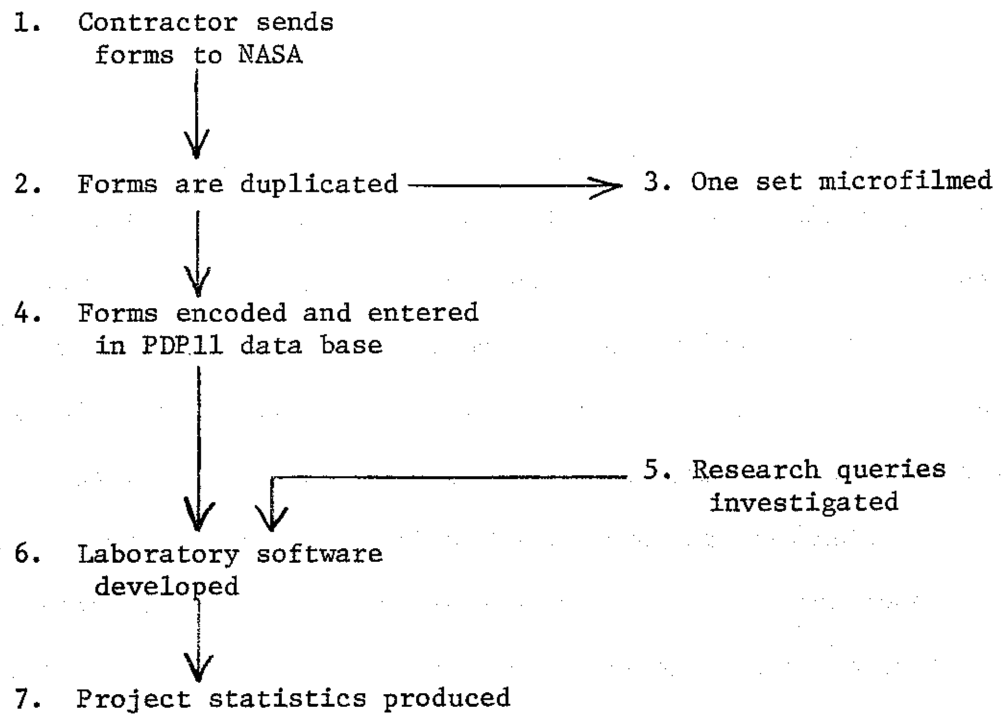
1. Contractor sends forms to NASA
2. Forms are duplicated
3. One set microfilmed
4. Forms encoded and entered in PDP11 data base
5. Research queries investigated
6. Laboratory software developed
7. Project statistics produced

FIGURE 1. LABORATORY DATA FLOW

1. The contractor, about once a week, sends all forms for all projects that have been filled out to NASA/GSFC (Figure 1.1).

2. The forms are duplicated at NASA (1.2), and one set is sent to be microfilmed for archiving (1.3).

3. The other set is sent to the University of Maryland where it is encoded onto coding sheets and entered into a PDP11 data base (1.4). As will be explained later, the data is read four times before being entered into the data base. Thus we believe that we have eliminated virtually all clerical errors in the process.

While the above are continuing operations, the following tasks are of a research nature and occur periodically:

4. The graduate students develop queries about aspects of the data to be tested against our collected data (1.5). At the same time, special purpose

laboratory software is developed for use in the laboratory (1.6). This includes enhancing the data base system, programs to validate the data and special purpose routines, such as a plotting package.

5. The queries are run against our collected data and statistics about project attributes are produced (1.7). The output is either a list of factors fulfilling the query, a table of statistics, or a plot of one factor versus another (generally some factor versus time). Some of the factors that we are measuring include size, cost and time. Less tangible factors include level of specification and development technique used.

Later sections of this report will describe most of these steps in greater detail.

Most of the fall of 1976 was devoted to developing the set of reporting forms to capture the data on each project, with the first projects submitting completed forms as of December, 1976. A set of seven forms have been developed. These forms can be divided into two classes - those that describe attributes of a project and are filled out relatively infrequently, and those that monitor progress and are filled out frequently. The forms that describe attributes of a project are the following:

1. General Project Summary - This form is filled out at each project milestone, and there are from five to ten milestones in the lifetime of a project (every six to eight weeks). This form describes the overall project structure, and the techniques used in its development. Such factors as project size, complexity, estimated milestones, required standards and documentation are given. How these change over time is one of the areas that we are investigating.

2. Component Summary - This form is similar to the general project summary except that it describes only one small section of a system

(e.g., function, subroutine, COMMON block, etc.). This form is filled out when that particular component is first designed in the design phase of the project and again when it is completed.

3. Programmer Analyst Survey - This form is used to collect general background information on project personnel, and is filled out only once by each programmer. It is used to compute a profile of project personnel in order to be able to compare two projects more equitably.

The forms that monitor project progress are the following four:

4. Resource Summary - This form is filled out weekly by the project manager and it lists the hours spent by each individual on the project. Since this information is already collected for accounting purposes, little additional effort was added. In addition, NASA/GSFC adds to this form a count of the total computer time used by the project and the total number of runs, as reported by the NASA computation center.

5. Component Status Report - This form is filled out weekly by each person on the project. It lists the number of hours spent on each component and the activity performed on that component (e.g., design code, test, etc.). Besides some of the obvious statistics we expect to obtain (e.g., number of hours spent in design vs. test vs. code, etc.), we can also use this form to verify the techniques used in project development. Unfortunately terms like "top down," "structured programming," "walk throughs," and "code reading" have become general purpose buzzwords with different inter-interpretations. One of the results that we expect to get from the component status report is to be able to classify the techniques that are being used rather than the ones we are told are being used. We can obtain this information by noting when certain activities occur for each component.

6. Computer Program Run Analysis - This form is filled out for each computer run. It is mostly a checklist of activities performed (e.g.,

compile, execute, utility, etc.). In addition, the error message and component terminating the run is listed. This will be correlated with the following change report form.

7. Change Report Form - For each change in the source code for a component (either to fix an error or to implement some change in specifications or design) a change report form is submitted listing the change and the reason for it. Since a form of this type has been in use for some time in this particular NASA environment, little additional work was added to existing NASA projects. One limitation, however, is that changes are not reported for any component until that component is placed into the project library. Thus, while we are getting all of the errors found in integration testing, we are not (except for a few of the monitored projects) getting much from the module design and development phases.

DATA COLLECTION

Broadly stated the purpose of the data collection phase is to transmit the raw data on each monitored project to the laboratory for processing and analysis. The primary requirement for each monitored project is to develop spacecraft support programs; therefore, the primary goal of the NASA/GSFC personnel is the management of project development on schedule. However, actual program development is by an outside contractor whose main goal is to deliver such projects on time in order to meet contractual obligations. Finally, the third group involved in the laboratory is the University of Maryland which is primarily interested in the analysis and measurement of the development process.

This three group organization increases the impartiality of the research group with the contractor group since it is independent of NASA and not involved in rating contractor performance and evaluating contractual arrangements. However, it does add communication problems.

to report back any data that can be identified to specific individuals. All programmer names are encoded in our data base, and all data reported back to the contractor (and to NASA also) is of a summary nature.

DATA PROCESSING

In the data processing phase, the data on the projects must be transcribed onto coding sheets and entered into a PDP11 data base. Such issues as data validity and completeness are of primary importance in this phase.

The raw data is checked four times before being entered into the data base. The forms are first encoded onto coding sheets and then checked for errors by a second person. The coding sheets are then typed into the computer and run through a special program which further checks for errors (e.g., format errors or improper fields such as a wrong component name in a certain project) and converts the data into a format suitable for inclusion into the data base. The output of this program is again verified against the coding sheets to further check for typing errors. By the time that the data is in our data base, we are confident that what has been entered is an accurate translation from the original forms. Whether these forms accurately describe the actual project, however, is another issue.

Forms validity is perhaps the hardest problem that must be tackled. Unfortunately, there are few general standards applying to all NASA/GSFC projects, and one of the requirements on the laboratory is to measure current techniques without impacting most projects. Thus, forms are filled out with varying degrees of completeness, and we must be careful in applying results across several projects. In one such case, there is about a factor of three in number of forms submitted between one project and others of a similar size.

The impact on projects brings to mind an example of the kind of problems we are faced with. One project manager has been reluctant to participate since his project is late and he sees no apparent benefit from participating in this research. While this group blames the forms for any increased lateness, they were already behind schedule before the forms were instituted. NASA/GSFC believes that this group has always been poorly organized and the problems with the forms only points out this problem and is not the cause of it. For well managed projects, the filling out of the forms is no more complex than any other administrative detail of a task.

This example shows one of the dangers in interpreting our collected data. Since forms may be the first to go when a project falls behind schedule, or since the more organized programming mind will process forms more accurately, the data may be biased towards projects that are on time This may lead to the unsubstantiated position that simply filling out of forms helps projects remain on schedule. While we believe that this may be true, it will be difficult to claim such interpretations.

The impact of the forms themselves is something which we would like to but cannot ignore. The contractor has asked for a 10% increase in costs for filling out of this data. While we do not believe that such estimates are realistic, NASA has agreed to it for now in order to develop a valid data base.

Another problem with the forms was their generality. In order to have a standard data base, a general purpose set of forms has been developed. Unfortunately, not every question is applicable in all situations. This adds overhead to filling out the forms since programmers do not know how to answer some of the questions. In our next iteration of the forms, we intend to abstract the typical answers being given at present and make the forms more of a checklist.

Given the completed forms, the next step was to encode them into the INGRES PDP11 data base ([HELD, 75]). Once in the data base, the first task has been to preserve data validity. In addition, special programs have been written to check for missing data. For example, since forms are uniquely numbered, missing numbers are easily spotted and usually mean missing forms. Also some forms are to be filled out weekly so it is easy to spot missing ones in that sequence.

DATA EVALUATION

The data evaluation phase consists of developing measures to evaluate the methodologies used on the projects. One technique recommended by the contractor was to provide instant feedback of the data on the forms. This was to help keep up the enthusiasm of the programmers for the entire endeavor. Since we did not want to report back any information that could be used for personnel evaluation, we designed some general purpose reports which give summary totals about each project.

The next step was to start developing more complex validity checks on the computerized data base. This has been the major emphasis of the laboratory for the last few months. Due to the large variation in programmer performance, we are trying as much as possible to make sure that what we have is correct. Comparing similar data in different ways on different forms is one way we are checking for consistency. For example, programmer times from the resource summary and from the component status report can be compared. Also, error runs from the computer program run analysis form can be compared to the number of change report forms submitted. Looking at gross data on projects that have not been monitored is another technique. Also, interviews with some of the contractor personnel has been used to validate certain answers.

INGRES queries are now being written to extract detailed information from the encoded data. A plotting package has been implemented to plot project characteristics. We are currently at the stage where some of these queries are being tested on the data base.

VI. SUMMARY

As this report describes, the Software Engineering Laboratory has been in operation for a year and is clearly a very complex operation requiring constant attention to detail and solving the many small and large practical problems as they arise. More effort than we originally planned is going into day-to-day operation and maintenance. Forms and data base validity have become the critical section through which all laboratory activities now depend.

We have been developing a prototype laboratory which can be used to perform the needed experimentation in order to develop theories of program development. Although developed within the NASA framework for software development, we do not believe that the problems we encountered or our solutions to them are unique. Given such care in maintaining the data, we believe that the data which is being collected is valid, useful, and should lead to important insights about program development.

ACKNOWLEDGMENTS

The efforts 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. DeSalvia, 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.

[LING, 77] Linger, R. C., Mills, H. D., Structured Programming Theory and Practice, Addison Wesley, 1977 (to be published).

[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.

[RAMA, 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 Corporation, Report MTR-2648, Vol. V, June 1973.

[TAUS, 76] Tausworthe, R. C., Standard Development of Computer Software, Part 1 Methods, Jet Propulsion Lab, Calif. Institute of Technology, Pasadena, California, 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 on 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 measurement 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.