Title:	Evaluation of Software Development Projects
Course:	
Year:	1991
Author(s):	M. G. Iyigun
Report No:	P91015

Abstract: The growth of software use in recent years has focused attention on the problems long associated with its development. Uncontrollable costs, missed schedules and unpredictable quality have all been regrettable trademarks of the software industry. The focus of this paper was obtaining data through a survey to be used to investigate sources of design modifications, reasons for changes and user-developer interface problems. Due to low response rate, the focus of the study was changed to the interpretation of gathered data in the field of knowledge rather than using survey

Evaluation of Software Development Projects

M. Guven Iyigun

P9115

EVALUATION OF SOFTWARE DEVELOPMENT PROJECTS

EMGT.510P

ADVANCED PROJECT MANAGEMENT IN ENGINEERING AND TECHNOLOGY

Submitted to Professor Dundar F. Kocaoglu

M. Guven IYIGUN

Engineering Management Program
Portland State University
Spring-1991

INTRODUCTION

The growth of software use in recent years has focused attention on the problems long associated with its development. Uncontrollable costs, missed schedules and unpredictable quality have all been regrettable trademarks of software industry. Moreover, these problems appear to vary in some proportion to the size and the complexity of development project [12]. To mention some of the well known [16]:

- Lotus 1-2-3, version 3 contains 400.000 lines of code. Labor required for development is 263 man-years with a cost of $7 million. For this version, Lotus spent approximately $15 million in quality control testing.
- Space shuttle contains one of the longest, most sophisticated, and most expensive software programs yet developed, with 25.600.000 lines of code, 22.096 man-years of development effort and cost of $1.2 billion.
- Cars can't run without software these days. On 1989 Lincoln Continental, a $29.000 luxury sedan, programs control the digital dashboard, brakes, air bag, door locks, steering, suspension, engine and air conditioner. These software contain 83.000 lines of code with 35 man-years development effort and a cost of $1.8 million.

More important, all of these software programs are completed much later then they are expected and planned. What appears to be a straightforward technical task in the beginning turns into a management crisis in later stages. It takes many hours of debate, sorting through a ton of procedural issues, and requires policy decisions completely unrelated to the actual coding. As the computers grow more powerful and user demands grow more sophisticated, the process of developing new software and maintaining and modifying old systems has turned into an exercise in frustration at the cost of millions of dollars [16].

Error detection and error correction are now considered to be the major cost factors in software development [3], [20]. Much of the recent research is devoted to finding ways of preventing software errors. This research includes the areas of requirements definition [1], automatic and semiautomatic program generation, functional specification [11], verification [8], coding techniques [9], error detection and testing.

AIM OF THE STUDY

The purpose of this paper was to obtain data through a survey that may be used to investigate sources of design modifications, reasons of changes and user developer interface problems. Ninety-seven survey instruments are sent to Software Development Companies located all in Oregon. Addresses of the companies are found from Oregon and Southwest Washington Manufacturers Directory [21]. A copy of the mailed survey instrument can be found in Appendix. Fourteen responses are obtained, corresponding to 14.4% response rate. However low response rate unabled the processing of gathered data statistically. Therefore, focus of the study has been changed to the interpretation of gathered data within best of knowledge.

Overview of the Data

A summary of response data can be found in Appendix. Following statistics are given as indicators of <u>average</u> project characteristics of response data.

Effort Required (work-days)	:	235
Effort Required (man-month)	:	18.76
Number of Developers	:	2.39

New Lines of Code Developed	:	12850
Lines of Code Used, Developed Earlier	:	20800
Number of Modifications	:	12.86
Number of Errors	:	10.36
Changes per 1000 Lines of Developed Code	:	1.00
Errors per 1000 Lines of Developed Code	:	0.81
Errors per Developer	:	4.33
Errors per Developer per Month	:	0.55
Modifications per Developer	:	5.38
Modifications per Developer per Month	:	0.69
Errors per Work Month	:	1.32
Modifications per Work Month	:	1.64

Changes are divided into two categories; error corrections and modifications. Errors are defined to be the discrepancies between specifications and their implementations. Modifications are changes made for purposes other than error correction.

Questions of Interest

Main questions of interest prior to conducting the survey can be stated as follows:

1. What is the distribution of changes according to reasons for changes?
2. What is the distribution of effort required to design changes?
3. What is the distribution of error correction changes according to reasons for error correction?
4. What is the distribution of effort required to change errors?

5. When are the errors detected?

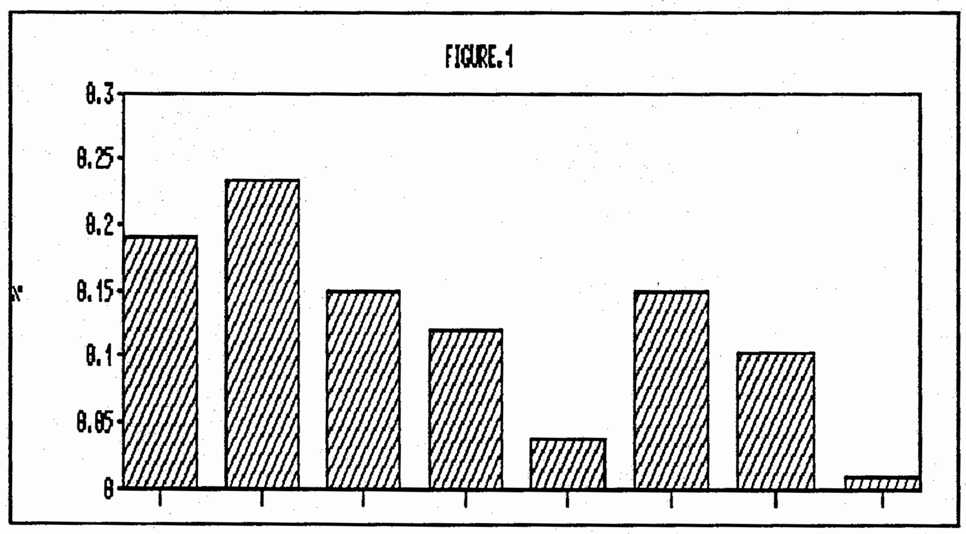
6. When do the errors enter the system?

As can be seen the survey instrument has been designed to look for these specific answers. Although, we have plotted the raw distribution of response data in an attempt to answer the questions of interest, inability to test the significance of distributions. In the next section, questions of interest, as stated above, will be answered without resorting to any statistical analysis.

Answering Questions of Interest

Following are the relations between the questions of interest and distributions.

Fig.1 : What is the distribution of changes according to reasons for changes?



A B C D E F G H

where A : Error correction

B : Planned Improvement

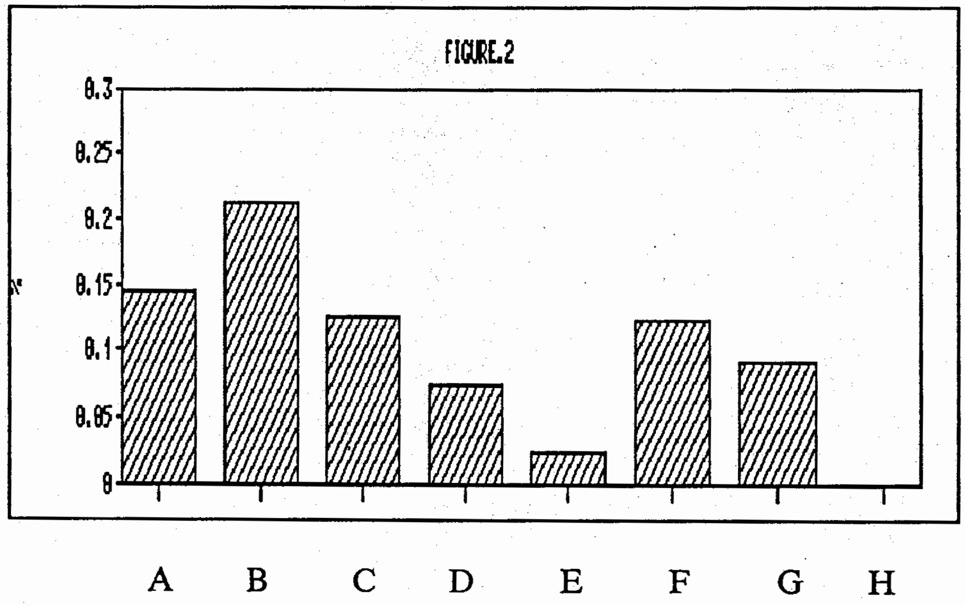
C : Implementation of Requirements Change

D : Improvement of Clarity, Maintainability or Documentation

E :	Insertion-Deletion of Debug Code
F :	Optimization of Time-Space-Accuracy
G :	Adaptation for Hardware Environment Change
H :	Other

As can be seen from the distribution approximately 75% to 80% of changes made to a software during its development cycle are due to user needs. Although a statistic giving the required time per change order is missing in the survey data, responding very late to 75% customer originating development change orders can be very costly in terms of time and customer satisfaction.

Fig.2 : What is the distribution of effort required to design changes?



where A :	Error Correction
B :	Planned Improvement
C :	Implementation of Requirements Change
D :	Improvement of Clarity, Maintainability or Documentation

E : Insertion-Deletion of Debug Code

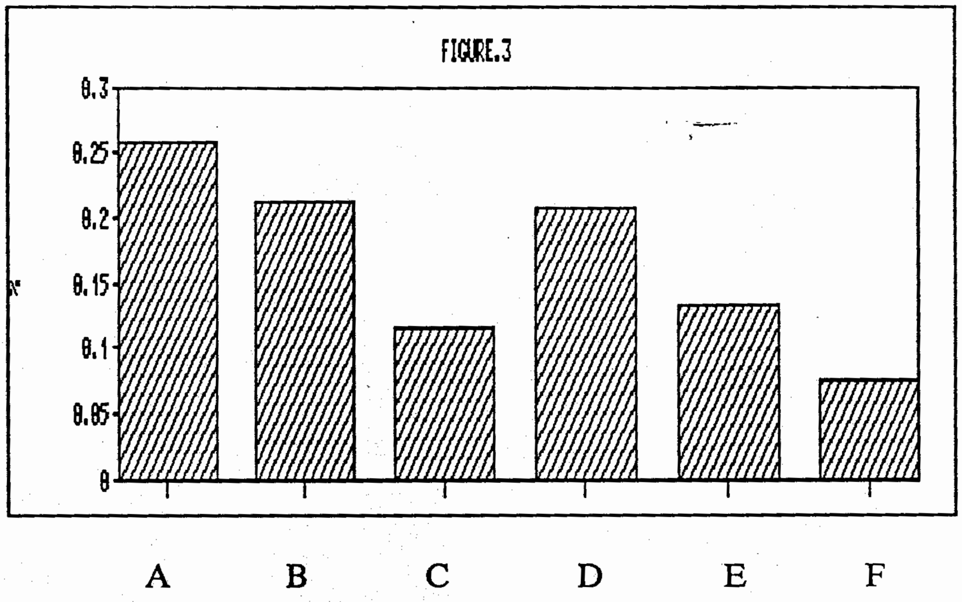
F : Optimization of Time-Space-Accuracy

G : Adaptation for Hardware Environment Change

H : Other

When inspected, we can see that approximately 65% of designing for change effort is related to customer needs. In other words, if a programmer is spending 100 hours of work on designing and performing changes associated with the software, he should spend 65 hours of his time for the design change order associated with the customer's needs. This figure gives the importance of how time and money saving it can be, understanding the requirements of customer as early as possible in the development phase. Probably some portion of time spend on customer design change orders can never be reduced due to customer's ever changing needs associated with today's world's requirements. But consider decreasing the time spend on requirements change by 50%. This will on the average decrease average project duration by 80 work-days, decreasing the software development time from 235 work-days to 160 work-days, or decreasing number of developers from 2.39 to 1.62 per project. This is a significant decrease. It means on the average approximately 50% more projects can be undertaken with the available development crew and available resources. It is worthwhile to note that these calculations and estimates are based on a realistic assumption of 50% decrease on the time spend for requirements changes. But on the other hand, of course all of these figures are reliable within a sample size of 14.

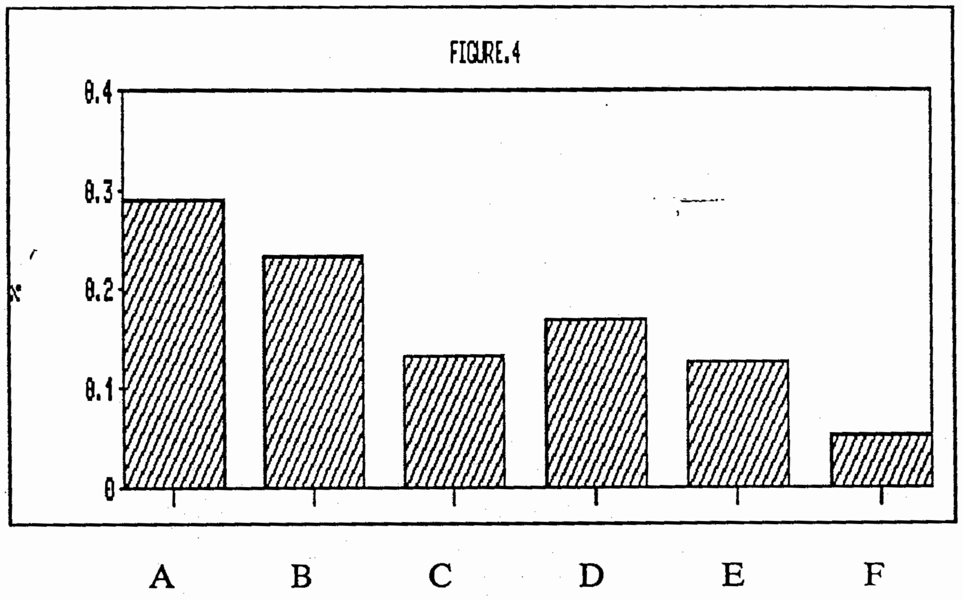
Fig.3 : What is the distribution of error correction changes according to reasons for error correction?



where A :	Requirements Incorrect or Misunderstood
B :	Design Error
C :	Misunderstanding Hardware Environment
D :	Error in Use of Programming Language
E :	Clerical Errors
F :	Others

In the above graph, category-A and category-C are of major importance to us. These are the time delaying activities which can be prevented if are attacked more systematically. Categories B, D and E are more related to the skills of programmer and the system analyst. More they are state of the art and scientific oriented, lesser the activities can be performed on these issues assuming the customer can sustains his or her integrity during the development process with minor deviations. Category-E (i.e. Other) responses include software compatibility and dialogue file changes at most.

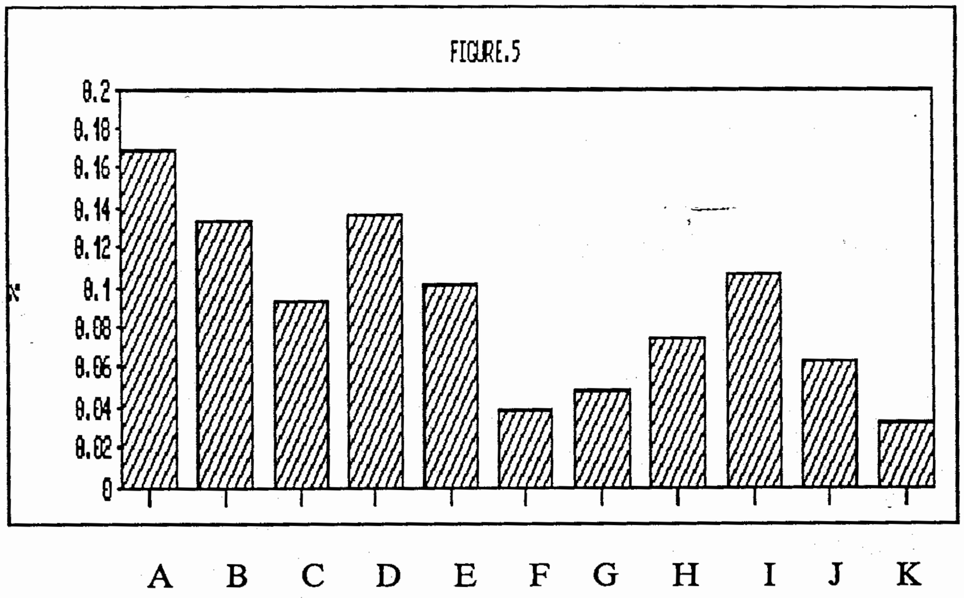
Fig.4 : What is the distribution of effort required to change errors?



where A :	Requirements Incorrect or Misunderstood
B :	Design Error
C :	Misunderstanding Hardware Environment
D :	Error in Use of Programming Language
E :	Clerical Errors
F :	Others

In the above graph, Category A and C, as stated in the previous distribution graph are of major concern to us. These show us the some percent of unnecessary time spent on correcting the misunderstood requirements of customer or hardware environment during the development process. In other words, some portion of these efforts can be eliminated by focusing more on requirements analysis, whether they are customer oriented requirements or customer's hardware requirements. Decreasing the rework time on these issues will obviously create some slack work force and slack time. These, in turn, could very well be diverted to some other projects.

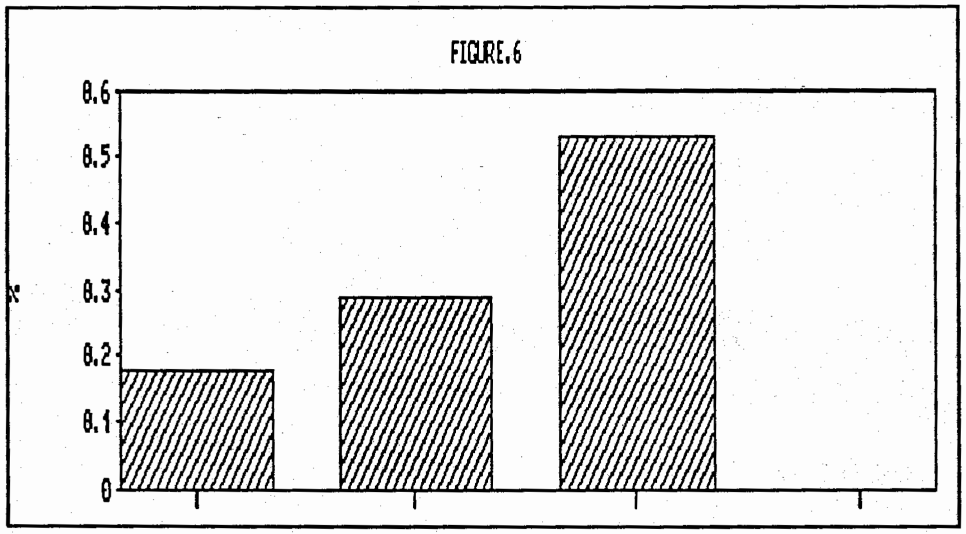
Fig.5 : When are the errors usually detected?



where A :	Pre-Acceptance Test Runs
B :	Acceptance Testing
C :	Post-Acceptance Testing
D :	Inspection of Output
E :	Code Reading by Programmer
F :	Code Reading by Other Person
G :	Talks with Other Programmers
H :	Special Debugging Code
I :	System Error Message
J :	Tracing
K :	Other

In above graph, we see that errors detected during post-acceptance use of programs or by a system error message, count 20% of error detection causes and time. 9% for post-acceptance use can be accepted as low, but a goal shooting for the decrease of this figure as much as possible must be present. High percentage associated with pre-acceptance test runs (17%), acceptance testing (13%), inspection of output (14%) and code reading by programmers (10%+4%) give us an idea why quality and reliability sustenance activities are so costly, as in the case associated with the development of Lotus 1-2-3 version 3 with a quality control cost of $15 million [16]. However, increasing use of special debugging codes for error detection may very well decrease the costs associated with quality and reliability sustenance.

Fig.6 : When do the errors usually enter the system?



A B C D

where A : During Requirements

B : During Design

C : During Coding

D : Other

Above graph is an interesting result. Although the previous distribution graphs show that there are some problems associated with customer-developer interface, above distribution shows exactly the contrary, stating that requirements analysis are well done such that only 18% of the errors enter the system during the requirements phase. From the responses to this question, one is really interesting to quote here:

'...customer don't know what they want.'

Is this really the case. Unfortunately, low response rate, unabled to test the different kinds of hypothesis, that were planned to be tested, associated with the causes of user-developer interface problems. However, literature state that, customer and developer are equally responsible for the introduction of errors to the system during the requirements analysis phase. Inability of most customers to think technically and usually the lack of a different perspective among the developers cause the entrance of errors into the system. Since no data is gathered from the customers, counter hypothesis could not been tested, which may very well state that;

'...programmers don't understand what we want.'

Another comment on the above graph is that, developers may not be aware of the real source of error. For example, a great portion of Category-B may be due to inefficient design due to insufficient understanding of customer needs. This in turn, an inefficient and poor design, may easily increase coding errors, resulting with a chain effect. However, these assumptions can not be verified or rejected, with the current sample size, but can only be speculated on.

As can be seen, upto now, focal point of our discussion has been the importance of requirements analysis and design for the software development projects. In the following section, some guidelines for overcoming the difficulties associated with the design for manufacturability of the software projects will be tried to be given.

RECOMMENDATIONS

As can be seen from the analysis above, understanding the requirements of the customer and designing the software according to the needs of the user are very important issues. In the following discussion, some guidelines will be stated to overcome these problems.

Most of the respondent comments stated the importance of working with modular development of large scale software projects and the importance of sustaining software reliability. Importance of working with modular designs is appear to be true with the reporting of high lines of code used in the current projects which were developed earlier. Therefore, we will mainly focus on the modular design improvement and reliability sustanence aspects of the development process. All of the guidelines-or better to say, the ideas-stated below are a synthesis of the articles given in the reference section.

The different modules of large scale programs must be considered early in the design process because of demands each type of module places on the design. This is one of the reasons why the design of the program and development process must occur simultaneously. The use of modules can streamline the production of software since modules can be developed, built and tested separately [4], [8], [9], [11]. Modules can be developed by specialized programmers which is especially advantageous if the functions which will be performed by different modules are within the specialty of a specific programmer or a group. The optimal method of combining the modules can be chosen early in the design process if the program and the development process are designed simultaneously. In other words combining the modules of a large program can be simplified if one the modules can serve as a base module upon which others are added. Ideally, the base module must be the one which is the first module to enter the programming process and all others are added to it until completion [8], [9], [16]. Another important point with the process of combining the modules is the ease of each module's independence and easiness to be replaceable. Programs with independently replaceable modules are easier to upgrade and debug because they can be debugged without having to modify other modules first. Extending the module's life is another factor important when dealing with modules. Early consideration of program development and upgrading strategy could be crucial to extending the life of a computer program [6], [16]. Advances in the hardware and software environment should be anticipated so the program can be developed or upgraded without a complete redesign. Modular design concepts can be used to allow modules that are prone to obsolescence to be replaced with upgraded ones. Extending program life through upgrading allows large programs generate even more profit after the development and introduction costs have been paid off. If the entire program is a collection of pre-tested modules, full program testing may be eliminated or reduced to only a final go/no-go test before shipment. In designs where potential reliability problems are concentrated in one module during development, test and diagnostic attention could be focused on that module. Programs comprised of modules are easier for maintenance by simply debugging the faulty module [8], [9], [11], [14].

If confidence on development process control is not high enough to ship program without testing, the program will need to be tested, prior to shipment. The program must be designed so that it will allow efficient testing. On very complex programs needing diagnostics, test development may cost more and take more calendar time than program development. This was one of the reasons Lotus 1-2-3 version 3 came so late to market [16]. Diagnostics of Lotus 1-2-3 version 3 cost Lotus more than $15 million. As stated previously, modules must be structured to allow modules to be tested separately prior to combining modules [11]. The interaction between modules should be avoided if possible otherwise be predictable. It can also be useful to be able to test modules separately after combining into the base program. Tests should be designed to be accomplished quickly by standard test instruments (such as with special debugging codes), which are easier to obtain and do not themselves, need to be debugged, as may be necessary with custom built test instruments. Since testing itself, is not a value-added activity, program reliability goals should be achieved with the minimum test effort.

Another issue at the moment is the designing for reliability. A program with good reliability has the freedom from failure in use. The elements of reliability includes probability, performance, time and usage conditions. If the program has reliability problems, engineering resources will be depleted by excessive test development [6], [16]. In the worst cases, reliability problems can force a redesign. In the cases where a large program consists of extensive number of modules, reliability can be a serious problem, since failure of a single module may cause the whole program to fail, if the program has redundancy or back-up features. Other issue with module reliability is how the modules are integrated into the program and how they interact with other modules. Designs should from the beginning have goals of good integration of modules and minimum chance of interaction between modules. More program complexity and modules means more attention needs to be paid to the integration of modules and to minimizing module interaction.

Designing for upgrading and maintenance is also an important issue. The need for ease of maintenance is proportional to the need for the program to be maintained in the house or at user. Design principles for ease of maintenance in post-acceptance use of software may be stricter than pre-acceptance maintenance, due to repetitive design reviews for already archived programs and due to obsolescent know-how on the program developed sometime in the past. Therefore testing procedures must have the capacity to diagnose problems. Anticipating the most likely maintenance tasks and planning for ease of maintenance and upgrading will probably help a lot too [8], [9], [14]. This can very well apply to module removal and/or module reinstallation. Modular maintenance is especially applicable for modules which need specialized development crew.

APPENDIX

Portland State University
Engineering Management Program
P.O. Box 751 Portland, OR 97207-0751

April 26, 1991

Dear Sir/Madam,

As part of the graduate degree requirements in the Engineering Management Program at Portland State University, we are conducting a study to investigate the sources of design modifications, reasons for changes and user-developer interface problems in software development projects.

We hope that you will be willing to spend a few minutes to answer the enclosed survey. Your response will be an important contribution to the study. Questionnaires are being sent to a select group of people in the software development industry in Oregon. The reliability of the findings depends heavily on the response of each individual.

We will appreciate it very much if you would please respond to the enclosed survey and return it by May 6, 1991.

If you would like to receive a summary of our results, please indicate it by checking the box at the beginning of the survey. We would be happy to send it to you.

Thank you for taking time to assist us in our study.

Sincerely,

M. Guven IYIGUN, MSEM
Teaching and Research Assistant

Attachment

Phone: (503) 725-4660 FAX: (503) 725-4667 BITNET: d6emp@psuorvm INTERNET: d6emp@psuorvm.cc.pdx.edu

PLEASE FEEL FREE TO DISTRIBUTE COPIES OF THIS SURVEY

Please complete this survey instrument by May 6, 1991 and return to:

M. Guven IYIGUN
Engineering Management Program, Portland State University, P.O. Box 751
Portland, OR 97207-0751

Respondent's Name	______________
Position	______________
Company	______________
Address	______________
	______________

Would you like to receive a summary of the report :____YES____NO

NOTE : Throughout this survey,
The term "ERROR" is used as a discrepancy between a specification and its implementation
The term "MODIFICATION" is used as the planned changes

Please consider the last software project you have worked on and respond to the following questions:

A. PROJECT CHARACTERISTICS:

1. Time required to finish the project
   - [ ] Less than 3 months (please specify____________)
   - [ ] 3 to 6 months
   - [ ] 6 to 12 months
   - [ ] More than 12 months (please specify____________)
2. Number of software developers involved in the project
   - [ ] Less than 3
   - [ ] 3 to 6
   - [ ] 6 to 10
   - [ ] More than 10 (please specify____________)
3. New lines of code developed (please specify within the applicable range)
   - [ ] Less than 10,000 lines (____________)
   - [ ] 10,000 to 20,000 lines (____________)
   - [ ] 20,000 to 30,000 lines (____________)
   - [ ] More than 30,000 lines (____________)
4. Lines of code used which were developed before (please specify within the applicable range)
   - [ ] Less than 10,000 lines (____________)
   - [ ] 10,000 to 20,000 lines (____________)
   - [ ] 20,000 to 30,000 lines (____________)
   - [ ] More than 30,000 lines (____________)
5. Number of modifications
   - [ ] Less than 5
   - [ ] 5 to 10
   - [ ] 10 to 15
   - [ ] More than 15
6. Number of errors
   - [ ] Less than 5
   - [ ] 5 to 10
   - [ ] 10 to 15
   - [ ] More than 15

B. THESE ARE THE CHANGES MOST OFTEN PERFORMED:
(Please rank: 7=most frequently.....1=least)

- [ ] Error Correction
- [ ] Planned Improvement
- [ ] Implementation of Requirements Change
- [ ] Improvement of Clarity, Maintainability or Documentation
- [ ] Insertion/Deletion of Debug Code
- [ ] Optimization of Time/Space/Accuracy
- [ ] Adaptation for Hardware Environment Change
- [ ] Other (________________________)

C. TIME SPENT TO MAKE CHANGES: (Please rank: 7=most.....1=least)

[ ] Error Correction
[ ] Planned Improvement
[ ] Implementation of Requirements Change
[ ] Improvement of Clarity, Maintainability or Documentation
[ ] Insertion/Deletion of Debug Code
[ ] Optimization of Time/Space/Accuracy
[ ] Adaptation for Hardware Environment Change
[ ] Other (________________________________)

D. THESE ARE THE MOST OFTEN PERFORMED ERROR CORRECTION CHANGES:
(Please rank: 5=most frequently.....1=least)

[ ] Requirements Incorrect or Misunderstood
[ ] Design Error
[ ] Misunderstanding Hardware Environment
[ ] Error in use of Programming Language
[ ] Clerical Errors
'Clerical Errors are the errors that occur in the mechanical translation of an item from one format to another or from one medium to another'
[ ] Other (________________________________)

E. TIME SPENT ON 'ERROR CORRECTION' CHANGES: (Please rank: 5=most.....1=least)

[ ] Requirements Incorrect or Misunderstood
[ ] Design Error
[ ] Misunderstanding Hardware Environment
[ ] Error in use of Programming Language
[ ] Clerical Errors
[ ] Other (________________________________)

F. ERRORS ARE USUALLY DETECTED DURING: (Please rank: 10=most.....1=least)

[ ] Pre-Acceptance test runs
[ ] Acceptance testing
[ ] Post-Acceptance use
[ ] Inspection of output
[ ] Code reading by programmer
[ ] Code reading by other person
[ ] Talks with other programmers
[ ] Special debugging code
[ ] System error message
[ ] Tracing
[ ] Other (________________________________)

G. ERRORS ENTER THE SYSTEM DURING: (Please rank:3=most frequently...1=least)

[ ] Requirements
[ ] Design
[ ] Coding
[ ] Other (________________________________)

H. ADDITIONAL COMMENTS:

very much for spending your valuable time to assist us in our study.

	NUMBER OF RESPONDENTS														STATISTICS		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14			
A-1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0.36	0.38	7.85
	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0.21	0.23	
	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0.07	0.08	
	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0.29	0.31	
2	1	1	1	1	1	1	0	1	0	1	1	0	1	1	0.79	0.79	2.39
	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0.14	0.14	
	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0.07	0.07	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	
3	1	1	1	1	1	0	0	1	0	0	1	1	0	1	0.64	0.64	12857
	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0.14	0.14	
	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0.07	0.07	
	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0.14	0.14	
4	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0.36	0.42	20833
	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0.07	0.08	
	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0.14	0.17	
	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0.29	0.33	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	12.86
	0	0	0	0	1	1	1	0	0	1	0	0	0	1	0.36	0.36	
	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0.21	0.21	
	1	0	0	0	0	0	0	1	1	0	1	1	1	0	0.43	0.43	
6	0	0	1	1	1	0	0	0	0	1	0	0	0	1	0.36	0.36	10.36
	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0.14	0.14	
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0.07	
	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0.43	0.43	

B	7	7	7	4	0	7	3	3	5	5	7	7	0	3	4.64	0.19	
	6	4	5	6	7	6	7	7	7	0	5	6	7	6	5.64	0.23	
	5	5	6	7	0	0	6	5	6	0	4	3	0	4	3.64	0.15	
	3	3	4	5	0	3	0	2	2	7	5	2	0	5	2.93	0.12	
	4	1	1	1	0	0	0	1	1	0	1	1	0	2	0.93	0.04	
	2	6	3	2	6	4	0	4	4	6	2	5	0	7	3.64	0.15	
	1	2	2	3	5	5	0	3	3	4	2	4	0	1	2.50	0.10	
	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0.21	0.01	
C	1	7	5	5	0	5	3	4	3	5	2	4	0	5	3.50	0.14	
	5	4	7	6	7	7	6	7	7	0	6	7	7	6	5.86	0.21	
	4	5	6	7	0	0	7	6	6	0	6	3	0	3	3.79	0.13	
	3	3	4	4	0	0	0	1	4	4	4	2	0	4	2.36	0.08	
	2	1	1	1	0	0	0	2	1	0	1	1	0	2	0.86	0.03	
	6	6	3	3	6	4	0	5	5	7	5	5	0	7	4.43	0.12	
	7	2	2	2	5	6	0	3	2	6	5	6	0	1	3.36	0.09	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	
D	3	5	3	5	1	3	0	4	3	5	4	2	5	2	3.21	0.26	
	5	4	4	3	2	0	2	2	4	0	5	3	0	3	2.64	0.21	
	1	1	2	1	0	0	0	5	2	0	3	4	0	1	1.43	0.11	
	2	3	5	4	0	4	0	3	5	4	1	1	0	4	2.57	0.21	
	4	2	1	2	0	5	0	1	1	0	2	0	0	5	1.64	0.13	
	0	0	0	0	0	0	5	0	0	3	0	5	0	0	0.93	0.07	

E	9	5	4	3	2	3	0	3	5	5	5	2	5	4	3.93	0.29	
	8	4	5	5	1	0	2	4	4	0	5	3	0	3	3.14	0.23	
	7	1	2	2	0	0	0	2	2	0	4	4	0	1	1.79	0.13	
	5	3	3	4	0	5	0	5	3	0	1	1	0	2	2.29	0.17	
	3	2	1	1	0	4	0	1	1	4	2	0	0	5	1.71	0.13	
	0	0	0	0	0	0	5	0	0	0	0	5	0	0	0.71	0.05	
F	9	6	8	10	3	9	10	7	8	7	8	8	5	10	7.71	0.17	
	8	7	3	8	2	10	0	6	7	0	6	10	10	8	6.07	0.13	
	7	5	2	7	1	7	3	4	6	0	3	9	0	5	4.21	0.09	
	5	8	9	9	7	8	0	1	9	9	10	3	0	9	6.21	0.14	
	3	9	10	6	0	0	0	10	2	10	5	6	0	4	4.64	0.10	
	2	2	7	3	0	0	0	5	1	0	2	2	0	1	1.79	0.04	
	4	1	1	2	0	0	0	9	3	0	2	7	0	2	2.21	0.05	
	1	4	6	1	5	0	0	3	10	0	7	4	0	6	3.36	0.07	
	10	10	5	5	6	0	0	8	4	0	8	5	0	7	4.86	0.11	
	6	3	4	4	4	0	0	2	5	8	0	1	0	3	2.86	0.06	
	0	0	0	0	10	0	0	0	0	0	10	0	0	0	1.43	0.03	
G	1	2	1	1	1	1	0	1	1	1	2	1	0	1	1.00	0.18	
	2	1	2	2	2	2	0	2	2	2	2	2	0	2	1.64	0.29	
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3.00	0.53	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.00	

REFERENCES

As can be seen some of the following articles are not cited in the body of the paper. Their main contribution to the paper was in the phase of developing the survey questionnaire.

[1] T. Bell, D. Bixler and M. Dyer, 'An Extendable Approach to Computer-Aided Software Requirements Engineering', IEEE Transactions on Software Engineering, vol.SE-3, January 1977, pp:49-60

[2] B.W. Boehm and R. Ross, 'Theory-W Software Project Management: Principles and Examples', IEEE Transactions on Software Engineering, vol.15, no.7, July 1989, pp:902-915

[3] B.W. Boehm, 'Software and its Impact:A Quantitative Assessment', Datamation, vol.19, May 1973, pp:48-59

[4] W.C. Cave and A.B. Salisbury, 'Controlling the Software Life Cycle-The Project Management Task', IEEE Transactions on Software Engineering, vol.SE-4, no.4, July 1978, pp:326-334

[5] J.D. Cooper, 'Corporate Level Software Management', IEEE Transactions on Software Engineering, vol.SE-4, no.4, July 1978, pp:319-326

[6] E.B. Daly, 'Management of Software Development', IEEE Transactions on Software Engineering, May 1977, pp:229-242

[7] A.M. Davis, E.H. Bersoff and E.R. Comer, 'A Strategy for Comparing Alternative Software Development Life Cycle Models', IEEE Transactions on Software Engineering, vol.14, no.10, October 1988, pp:1453-1461

[8] E.W. Dijkstra, A Discipline of Programming, Englewood Cliffs, New Jersey, Prentice Hall, 1976

[9] E.W. Dijkstra, 'Notes on Structured Programming' in Structured Programming, London, England, Academic Press, 1972

[10] C.W. Doerflinger and V.R. Basili, 'Monitoring Software Development Through Dynamic Variables', IEEE Transactions on Software Engineering, vol.SE-11, no.9, September 1985, pp:978-985

[11] K. Heninger, 'Specifying Requirements for Complex Systems:New Techniques and Their Application', IEEE Transactions on Software Engineering, vol.SE-6, January 1980, pp:2-13

[12] C.P. Hollocker, 'Finding the Cost of Software Quality', IEEE Transactions on Engineering Management, vol.EM-33, no.4, November 1986, pp:223-228

[13] F.N. Parr, 'An Alternative to the Rayleigh Curve Model for Software Development Effort', IEEE Transactions on Software Engineering, vol.SE-6, no.3, May 1980, p:291

[14] L.H. Putnam, 'A General Empirical Solution to the Macro Software Sizing and Estimating Problem', IEEE Transactions on Software Engineering, vol.SE-4, no.4, July 1978, p:345

[15] C.L. Ramsey and V.R. Basili, 'An Evaluation of Expert Systems for Software Engineering Management', IEEE Transactions on Software Engineering, vol.15, no.6, June 1989, pp:747-759

[16] B.R. Schlender, 'How to Break the Software Logjam', Fortune, September 25, 1989, pp:100-112

[17] R.F. Scott and D.B. Simmons, 'Predicting Programming Group Productivity: A Communications Model', IEEE Transactions on Software Engineering, vol.SE-1, no.1, December 1975, pp:411-414

[18] D.M. Weiss and V.R. Basili, 'A Methodology for Collecting Valid Software Engineering Data', IEEE Transactions on Software Engineering, vol.SE-10, no.6, November 1984, pp:728-738

[19] D.M. Weiss and V.R. Basili, 'Evaluating Software Development by Analysis of Changes:Some Data from the Software Engineering Laboratory', IEEE Transactions on Software Engineering, vol.SE-11, no.2, February 1985, pp:157-167

[20] R. Wolverton, 'The Cost of Developing Large Scale Software', IEEE Transactions on Computers, vol.C-23, no.6, 1974

[21] Manufacturers Directory, Resource Guide Oregon High Technology, Oregon & Southwest Washington