A Case History Analysis of Software Error Cause-Effect Relationships

Takeshi Nakajo and Hitoshi Kume

Abstract—Software errors have been studied from various perspectives; however, most investigations have been limited to an individual section or a partial path of the cause-effect relationships of these errors. The present study analyzes approximately 700 errors in 4 commercial measuring-control software products, and then identifies the cause-effect relationships of errors occurring during software development. The analysis method used was: (i) defining appropriate observation points along the path leading from cause to effect of a software error, followed by gathering the corresponding data by analyzing each error using Fault Tree Analysis, and (ii) categorizing each observation point’s data, and then summarizing the relationships between two adjoining points using a cross-indexing table. This paper presents four major cause-effect relationships and discusses the effects of the Structured Analysis and Structured Design methods on these relationships.

Index Terms—Software development, cause-effect relationships, case-based error analysis, program faults, human errors, process flaws, structured analysis and structured design, Fault Tree Analysis.

I. INTRODUCTION

Many problems due to human error occur during software development, yet effectively eliminating root causes from design processes to prevent their recurrence is difficult unless the types of human errors and their cause-effect relationships are identified [1]. This identification is especially important for long-term improvement of software-development techniques.

Although there have been many quality investigations of human errors in software development (e.g., [2]–[5]), most have been limited to examining an individual section or a partial path of the cause-effect relationships of software errors, with no previous report systematically investigating these relationships.

This paper examines empirical errors in four commercial, measuring-equipment, control software packages, and then identifies the cause-effect relationships of these errors. Section II describes a method used to analyze the cause-effect relationships based on software-error case data. Applying this case-based method, Section III shows that four major error occurrence mechanisms exist, whereas the effects of Structured Analysis and Structured Design methods [6], [7] on these mechanisms are discussed in Section IV. Finally, Section V compares the presented analysis method with the common empirical methods used to identify software-error cause-effect relationships.

II. IDENTIFICATION OF CAUSE-EFFECT RELATIONSHIPS BASED ON ERROR CASE DATA

One method of identifying the cause-effect relationships of software errors is to analyze error case data. Fault Tree Analysis (FTA) accomplishes this by using a logic tree diagram to present each error’s cause-effect relationships, and develops a countermeasure for each error [8]. Although this method can handle the complex structure of cause-effect relationships, it is difficult for FTA to extract the relationships common to all errors under investigation, because it analyzes each error individually.

The presented analysis method basically follows FTA, yet facilitates the extraction of the cause-effect relationships inherent in software errors by restricting the logic tree diagram’s structure using predefined observation points, and presenting the relationships between two observation points using cross-indexing tables, i.e.,

1) Define observation points along an error’s logical path from cause to effect
2) Gather observation point data by analyzing each error using FTA
3) Categorize each observation point’s data
4) Identify the relationships between each two adjoining observation points using a cross-indexing table
5) Summarize the major cause-effect relationships based on a series of the cross-indexing tables.

The key elements in the above analysis method are: 1) observation points, and 2) categorization of each observation point’s data.

Fig. 1 shows various types of error information along the path from cause to effect. The following considerations must be incorporated when choosing the observation points:

1) A human error occurring during software development is first detected as “system failure”; however, the “system failures” vary widely. Tracing back to the source of a human error reveals that “inappropriate work system management” was ultimately responsible for the error, although the details surrounding this state vary and are strongly dependent on the organization’s characteristics.
III. CAUSE-EFFECT RELATIONSHIPS IN MEASURING EQUIPMENT-CONTROL SOFTWARE

The method described in Section II was applied to 670 software errors occurring during the software development phase of 4 commercial measuring equipment control packages. This analysis identified the cause-effect relationships of software errors in order to improve the development techniques of high-reliability software.

Two of the four software packages investigated were subsystems of an IC measuring system, with one being a set of "intrinsic routines" for users' measurement programs, and the other an application program using these routines. The other two were firmware products which control measuring instruments for electronic components. The four products, respectively referred to as A, B, C, and D, have the following common characteristics:

1) Significantly hardware-specific, since they are designed for initializing, setting, and measuring hardware
2) Consist of a set of unit functions that can be combined to produce various required functions
3) Numerous global variables for memorizing hardware equipment status are required.

The errors used in the analysis were all found during or after the module testing phase. The three following observation points were selected:

1) Program fault
2) Human error
3) Process flaw.

The specifics of each error corresponding to these observation points were collected through source code investigation, design documents and their change histories, and an interview with the designer or programmer who made the error. The Affinity Diagram method was selected for observation point data classification, with Table I showing an example of data collected.

A. Types of Program Faults

Fig. 2 shows the results of "program fault" data classification.
Mismatch between a failure to set hardware to a receiving state and the expectation that the proportion of internal products caused by failure to set hardware to a receiving state reject the hypothesis that the proportion of internal products caused by failure to set hardware to a receiving state is the same as that in Products C and D.

— Mismatching in the data transference or control between a module and its environment, i.e., other modules, global variables, data files, and hardware (Not caused by Fault 1).
— Name Faults: Name mismatch (53/10.4)
  Ex) Referring to a nonexistent local variable.
— Structural Faults: Structure, type, or configuration mismatch (99/19.5)
  Ex) Mismatch between a subprogram’s argument type and its calling statement’s corresponding variable.
— Value Faults: Value meaning or possible value range mismatch (99/19.5)
  Ex) Substituting the wrong value for an argument that switches a sub-module’s function.
— Procedural Faults: Data transfer or control procedure mismatch (38/7.5)
  Ex) Failure to set hardware to a receiving state before transferring data.
— Module Function Faults (169/33.3)
  Incorrect operations of a module resulting in unsatisfied upper functionality, requirements (159/31.8)
  Operating Faults: Operation omission or unnecessary operations (59/11.8)
  Ex) Failure to set an item to hardware that will be used in the following commands or modules.
  Condition Faults: Incorrect operation conditions (79/15.8)
  Ex) Use of an incorrect limit value in judging whether or not to reset hardware.
  Behavioral Faults: Incorrect behaviors (31/6.1)
  Ex) Displaying terms not conforming to requirements.

Notes:
1) Goodness of fit test [10] using a 5% significance level did not reject the hypothesis that the proportion of internal errors is the same for all products.
2) Goodness of fit test using a 1% significance level rejected the hypothesis that program fault distribution in Products A and B is the same as that in Products C and D.

3) In the system software programs, the name, structural, and value faults were the most common types of interface faults, with the operating and condition faults being the most common types of module function faults.

B. Cause-Effect Relationships Between Human Error and Program Fault

This section focuses on the two major program fault types identified in Section III-A: interface fault and function fault, and describes human-error categorization results, followed by examining the relationships between human error and program faults.

An interface fault-mismatching between various software components is closely related to technical information communication between the various component’s development teams. On the other hand, a module function fault is closely related to designing software which satisfies the users’ needs. Because it is well known that differences in these types of works greatly affect the types of human errors which occur during the work process, 289 human errors causing interface faults and 169 human errors causing function faults in system software products A and B were independently classified. Fig. 4 shows the human-error categorization results.
Table II summarizes the relationships between human error and program faults using the categorizations in Figs. 2 and 4. From Table II the following can be seen:

1) More than 50% of interface faults were due to a misunderstanding of software interface specifications; i.e., communication errors within a development team. On the other hand, in the case of communication errors occurring between a development team and others, misinterpretation of hardware interface specifications occurred more frequently.

2) Misunderstanding of software interface specifications causes primarily name, structural, or value faults, while misunderstanding of hardware interface specifications causes structural, value, or procedural faults.

3) Approximately 45% of the function faults were caused by a misunderstanding of system functions (i.e., an error in recognizing requirements), with about 40% of function faults being caused by a misunderstanding of module functions.

4) System function misunderstandings cause mainly condition and behavioral faults, while module function misunderstandings cause operating faults more than condition faults.

C. Cause-Effect Relationships Between Process Flaw and Human Error

This section focuses on the following four major human-error types by describing the process flaw categorization results, and then examines the relationships between process flaws and human error.

Error I: Human errors causing interface faults:

1) Misunderstanding of software interface specifications
2) Misunderstanding of hardware interface specifications.

Error II: Human errors causing function faults:

1) Misunderstanding of system functions
2) Misunderstanding of module functions.

Process flaws causing 406 human errors in Products A and B (falling under the 4 above categories) were classified based
on their similarity. Fig. 5 shows the process flaw categorization results.

Tables III and IV summarize the relationships between process flaws and human errors using the categorizations in Figs. 4 and 5. From these tables the following can be seen:

1) Inconsistent and distributed interface definitions were the major design-principle flaws that cause the misunderstanding of software and hardware interface specifications.

2) The design-management flaws that cause the misunderstanding of software interface specifications were inappropriate documentation for module calling methods and/or file and global variable definitions.

3) The major design-management flaws that cause the misunderstanding of hardware interface specifications were unclear documentation of hardware accessing methods.

4) Dependent/unsymmetrical system unit or module function definitions were the design-principle flaws that cause the misunderstanding of system and module functions.

5) The design-management flaws that cause the misunderstanding of system functions were unsystematic requirement documentation and insufficient documentation on how to combine system unit functions to satisfy these requirements.

6) The design-management flaws that cause the misunderstanding of module functions were inappropriate documentation of the relationships between system and module functions.

As a result, it is concluded that the error-occurrence mechanisms in the development of these products consist of the four major paths shown in Fig. 6.

IV. Effects of Structured Analysis and Structured Design Methods on Error-Occurrence Mechanisms

This section describes the effects of Structured Analysis (SA) and Structured Design (SD) methods [6], [7] on the previously identified four error-occurrence mechanisms. Although the SA/SD methods are generally used in designing simply structured software, when used as design-management tools they can help suppress these error mechanisms.

A measuring equipment control system product, similar to Products A and B, was chosen to evaluate the SA/SD methods’ effects on error-occurrence mechanisms. For comparison, six
product subsystems were developed using SA/SD methods (SA/SD group), and six other subsystems of the same prod-

ut were developed without using SA/SD methods (Control group). Table V shows the characteristics and development conditions for both groups. All errors discovered during and after the module testing phase were analyzed and compared.

Fig. 7 shows the occurrence rates and proportions of the three program fault types for both groups, with Table VI giving the relationships between human error and program fault for the SA/SD group. This information, combined with that of Fig. 3 and Table II, yielded the following:

1. The internal fault proportion was constant for the four test groups; i.e., SA/SD, Control, and Products A and B
2. The internal fault-occurrence rate in the SA/SD group was the same as in the Control group
3. The interface fault proportion in the Control group was
the same as in Products A and B, whereas its proportion in the SA/SD group was smaller than in Products A and B.

4) The interface fault-occurrence rate in the SA/SD group was about a half of the Control group’s.

5) The function fault-occurrence rate in the SA/SD group was approximately 35% smaller than that in the Control group.

6) Although the amount of data was not sufficient to determine the relationships between human error and program faults, for the SA/SD group the proportion of structural faults due to software-interface specification misunderstandings increased, and the proportion of condition faults due to system-function misunderstandings decreased.

It is concluded that in this experiment the SA/SD methods suppressed the four error-occurrence mechanisms, and also that the effects on 1 and 2 were greater than those on 3 and 4.

V. DISCUSSION

The metric-based method is a common method of empirically analyzing software error cause-effect relationships [5], [11], [12]. This is an application of the method used to analyze hardware cause-effect relationships, and primarily uses correlation coefficients between two metrics measuring results and their causes. The method’s general outline is:

1) Divide the products under investigation into appropriate-sized units for analyzing cause-effect relationships, i.e., systems, subsystems, or modules.

2) Define the metrics which measure the unit characteristics and the unit-producing processes and then gather data.

3) Calculate the correlation coefficients between the unit’s metrics and the processes’ metrics and then evaluate them.

The key elements of this metric-based method are the size of the units used in Sten 1 and the metric types defined in...
Step 2, especially the process characteristic measurements. Method effectiveness is highly dependent upon these elements (e.g., large units decrease the amount of data for calculating correlation coefficients and do not isolate the effects of various causes, whereas small units increase data variance and the respective correlation coefficient variance). Usually trial-and-error techniques determine the appropriate unit sizes, and similarly the process characteristic metrics are primarily selected based on either specific engineering models or experience.

Another approach in determining the cause-effect relationships of software errors is to individually analyze each error. FTA is one such method, where unit sizes and process characteristic metrics are not required to be determined in advance. FTA is flexible enough to handle each error’s complex cause-effect relationships; however, this flexibility makes it difficult for FTA to extract the relationships common to various errors. Additionally, analyzing more than one error at a time in order to obtain a fault tree diagram presenting common cause-effect relationships often results in a subjective analysis unrelated to individual errors. Another FTA method weakness is the number of man-hours necessary to collect data. In the metric-based method, data can be automatically collected once the metrics are established. Contrastingly, the FTA method is difficult and inappropriate to limit the data range in advance, and as a result requires more data-collection time. This degree of difference significantly depends on the development environment, especially on the error-tracking system.

As in FTA, the presented case-based analysis method does not require the unit sizes and process characteristic metrics to be determined in advance and can be applied without specific engineering models of the cause-effect relationships. The case-based method overcomes the FTA weakness by using both predefined observation points and a phased approach which individually tracks each error’s case data to extract the common cause-effect relationships inherent in the data. On the other hand, this case-based analysis method still requires a considerable amount of man-hours, because it basically falls under the category of the FTA method. The man-hours necessary for data collection are more critical than those necessary for data analysis; however, the data collection time can be fairly reduced once appropriate observation points are established.

VI. Conclusions

The paper focused on measuring-equipment control software and identified cause-effect relationships of software errors. Four major error-occurrence mechanisms were identified—two are related to hardware and software interface specification misunderstandings, and the other two are related to system and module function misunderstandings. The effects of the Structured Analysis and Structured Design methods on software errors were also evaluated using these mechanisms. The structured methods could suppress the error-occurrence mechanisms, being more effective on hardware and software interface mechanisms than on system and module function mechanisms. The presented case-based cause-effect relationship analysis method can be expected to supplement the metric-based and FTA methods. These conclusions were drawn from specific software packages and are thus limited to them. Given the similarities of software development procedures, however, the presented results can be generalized and applied to the development of other software products.

Acknowledgment

The authors wish to thank the following employees of Yokogawa-Hewlett-Packard Co. Ltd.: M. Mori (Instrument Division Manager) for his beneficial advice, and K. Sasabuchi, T. Akiyama, and I. Azuma (Product Assurance Section), and F. Turuda, M. Tada (Research and Development Sections), and many other software engineers for their help with the collection and analysis of software-error information.

References


Takeshi Nakajo received the B.S., M.S., and Ph.D. degrees in engineering from the University of Tokyo, Japan, in 1979, 1981, and 1986, respectively. He was a Research Assistant at the University of Tokyo from 1987 to 1990, and has been a Lecturer at Chuo University since 1990, where his special field is in quality improvement, especially the prevention of human errors, with his current research interests being in software-error analysis and design and testing methods. Dr. Nakajo is a member of IEEE Computer Society.
Hitoshi Kume received the B.S., M.S., and Ph.D. degrees in engineering from the University of Tokyo, Japan, in 1960, 1962, and 1965, respectively. He was an Associate Professor at the University of Tokyo from 1974 to 1980, and has been a Professor since 1980, where his special field is in quality control. Since 1960 he has been engaged in consulting work for the improvement of quality systems in Japan's various manufacturing industries. Dr. Kume is a member of the American Society for Quality Control and a Japan Delegate to TC176 (Quality Management and Quality Assurance) of the International Organization for Standardization.