AN ANALYSIS OF SOFTWARE PROJECT FAILURE

Joichi Abe
Hitachi Software Engineering Co., Ltd.
29 Yabe-cho, Totsukaku
Yokohama, 224 Japan

Ken Sakamura
Department of Information Science
University of Tokyo

Hideo Aiso
Department of Electrical Engineering
Keio University

ABSTRACT

The main aim of this paper is to indicate how various losses may be reduced or avoided when the development of software does not proceed according to its schedule; i.e., if what we call "bankruptcy" occurs. Data were collected from twenty three projects in various types of applications, the projects together containing a million lines of code. The causes of failure in developing software were obtained by interviewing the managers of the projects under observation. Having analysed these two aspects, this paper points out under what circumstances managers are likely to fail and proposes a method of detecting failures in the software development.

I. INTRODUCTION

It often happens that the target data of a software project cannot be met or that the amount of money spent exceeds the budget. In this paper these management failures are referred to as "bankruptcy" of projects. Since software is becoming more important in the computer industry these days, the necessity increases for detecting and preventing the occurrence of bankruptcy in the software development. There have been articles dealing with project management.[1,2,3,4,6] However, as far as we know, no articles have been published which systematically investigate the failure of project management. In the industrial field, the cause of each failure of software development project has been analysed and a lot of know-how has been accumulated on how to handle such cases. Meanwhile, only little is known about the general and common characteristics of these failed software development projects. The major obstacles in software project management are summarized as follows:

(1) Attempts to analyse the nature of the bankruptcy phenomena have not successfully been made yet.
(2) The occurrence of bankruptcies are rarely detected at an early development stage. Many of them are detected only in the final development period, i.e., during the test period and some just at the dead line of the target date.
(3) Effective methodologies for software management are under development in order to prevent bankruptcies or to minimize losses when they are inevitable.

In this paper we approach to the analysis of the bankruptcy characteristics from the view points of (1) and (2), and leave (3) for latter investigation. To assure generality we collected data from twenty three selected projects which together contain a million lines of code. The software for various computers ranging from minicomputers to large computers was developed in these projects. The software included part of operating systems, language processors, and of large scale online systems. Attention was focused on the final development stage, i.e., the test period. The progress information, which is a record indicating the status of the progress in a project, for instance, an arrow diagram of the PERT or a Guntt chart, does not always show difficulties inherent in the project in general. Most bankruptcies, however, become apparent during the test period.

In order to accomplish the project satisfactorily the test points called "check items", which are the test items to be checked in debugging the program, are always taken into consideration. Figure 1 is an example showing the relationships between the number of check items as well as the number of bugs detected and the number of working days in the test period. For each project data showing these relationships were collected and are normalized in order to extract common characteristics from the whole projects. The implication of normalization will be described in III-4. The results of our study are as follows:

(1) We found out several properties common to every software development project throughout the test period.
(2) We proposed a method for detecting the occurrence of a bankruptcy and evaluated the effectiveness of the method.
(3) The causes of bankruptcies were classified into a number of categories. We discussed under what circumstances the project managers were likely to fail.
(4) Combining the method of detecting the occurrence of bankruptcies (2) with the cause analysis results (3), a new method of project management was proposed and we pointed out the expected effect of the proposed method.

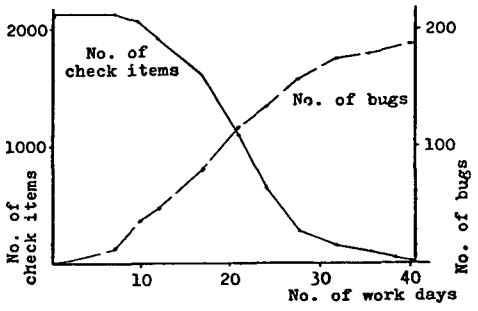

Figure 1 A PB Curve

II. BANKRUPTCIES OF SOFTWARE PROJECTS

Bankruptcy of a software project is defined as the status that is accompanied with heavy financial damage and or loss of reputation by not meeting the target date or an excess over the budget by approximately 20%. The schedule delay keeps many staffs of the project team from performing other tasks. Hence,

Reprinted from PROCEEDINGS OF 4th SOFTWARE ENGINEERING, September, 1979

the observation of bankruptcies can be performed by measuring the delay, i.e., the progress of the project.

II-1 Real Bankruptcy with Huge Losses

The computer application range has become wider, including cases in which failure leads to grave results. For example, there are applications for controling modern ironworks with a production capacity of 200K tons per month. They require an investment of the order of one billion dollars.[7,8] Therefore, it would loose at least five million dollars due to interest alone on the investment if its opening would be delayed by one month. Every software manager is sure to feel grave responsibility, once he has visited the site and he has seen the pier and harbor facilities for vessels carrying 100K tons of iron-ore, the blast-furnaces under construction, the converters, continuous roll works, and so on. The same is true for the software project of control system for large systems such as super express trains or a gigantic petroleum kombinate.

Therefore, it becomes more and more important to meet the target dates and to improve the quality of the software involved. The promoter of a big project has to make up for any delay occurring in one of its sub-projects including the software development. Under these circumstances, when a delay in software development occurs, the only thing that matters is to make up for the delay with all available resources such as manpower or computer facilities. Consequently, the relationship between the investment per month and time looks like the curve in Figure 2. The huge loss takes place after the bankruptcy occurs.

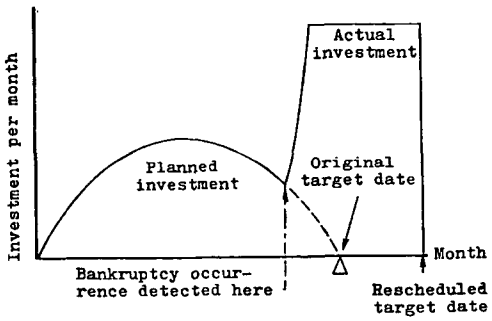

Figure 2 Investment Rate vs. Time of a Bankrupt Software Project

II-2 What Happens When a Project Goes Bankrupt.

After the occurrence of a bankruptcy the following will happen. If adequate countermeasures can be taken, or if it seems that the target date can be met, the height of the investment peak in Figure 2 can be limited to a reasonable boundary. If there is no possibility at all of meeting the target date, or if it is impossible to decide a new deadline, the height of the investment peak can become unbounded. The height now is dependent on the maximum available resources, i.e., manpower and computer facilities. Especially in cases when it is not possible to set a new target date, the investment peak cannot be estimated. No manager can foretell how much money will be lost.

The main objective of this additional investment is to shorten the delay time. If a bankruptcy occurs while developing software as part of a big project, it might be reasonable to try to minimize the gross loss, i.e., the sum of two losses: One is due to payment of interest and the other is due to the additional investment required to complete the software project. However, it is usually assumed that the gross loss is shortened by decreasing the delay time of the software development.

Even if it appears that the project is going to be delayed by only one day, it becomes necessary to invest all available resources for that one day. Furthermore, even if a new deadline can be set, it will be impossible to reduce the height of the investment peak, only its duration can be limited.

II-3 When to Detect the Occurrence of Bankruptcy.

It is desirable that a bankruptcy can be detected as long as possible before the occurrence. If the occurrence of a bankruptcy is detected halfway through the test period, the prospects for the software builder are not hopeless (this is not the case, if say, 90% of the available manpower has been spent). It should be noted that a major loss in the additional investment can be anticipated before it actually takes place. Moreover, the software project manager can determine the delay some time before the target date and consult the staff of the big project about countermeasures to be taken.

Whenever a bankruptcy occurs, a huge monetary loss is inevitable. Therefore, the project manager needs clear evidence of the trend toward a bankruptcy before he decides the occurrence. As we will see later, the occurrence of a bankruptcy becomes evident halfway through the test period.

II-4 How to Minimize Loss When Bankruptcy Occurs

It may be desirable to mention some countermeasures that can be taken in order to minimize loss. There are many possible methods which can be used. The following two methods appear to be effective. However, the effectiveness has not been systematically analyzed:

Method (1): Increasing planning reliability

Since the decision is based on the prospect for the project, the schedule planning reliability should be as high as possible. The higher the reliability is, the earlier the effective decision can be made. Hereby the possibility of finishing the project in time is increased and it leads to keeping the investment peak height within a reasonable boundary. The reliability can be improved, in most cases, by collecting information about troubles or by anticipating future shortages encountered in the software development.

Method (2): Preparing for possible emergencies

The only way to decrease the investment peak height is to find a way of finishing the project on schedule.

One way to realize this lies in training of engineers for possible emergencies. If one lets them investigate the system specifications or the functional specifications prior to the judgement about a bankruptcy, it will be helpful for a quick start in the bankruptcy aid operation.

Another way may be to determine which modules or functions are at the root of the trouble in order to provide the needed assistance timely.

III. ANALYSIS OF PROJECT CHARACTERISTICS

III-1 Macro Estimate of the Progress

We will propose a simple method to estimate the status of the progress, which is called the "macro estimate" of the progress, and we also propose a method to obtain a curve from which it is possible to decide if bankruptcy of a software project is going to occur or not. There are two kinds of bankruptcy as mentioned before: One is caused by an expenditure over the budget. The other involves not meeting the target date. In the first case, an estimate has to be made on what

degree manpower and other resources can be reduced to minimize the expenditure over the budget while keeping on schedule. This can be obtained by estimating the progress speed with reduced resources and determining the minimal resources with which the project can be finished just before the dead line. Then the manager can determine the ultimate financial loss. In both cases, the main obstacles to forecasting bankruptcy can be removed by a correct progress estimation and by judging whether or not the project can be finished on time.

L. Putnum has proposed a method to obtain a macro estimate of the manpower needed for a project.[1] According to his method, the manpower to be invested during the development can be estimated by the amount of the manpower invested per month so far. However, in our opinion, it is also necessary to have a data base containing the information on the projects, which are almost the same as the project under development, as for the number of lines of code generated or the number of files required. Without these data, the estimate of manpower and time would be widely scattered because the standard deviation in the data base is spreaded and the information may not be accurate enough to judge whether or not the project will be finished in time. Moreover, the Putnum's method barely mentions the relation between the capability of the project team and their task within the project. For instance, it could happen that the estimate of manpower and time would be the same, even if the project team were replaced by staffs with lower capability.

In this paper the estimate is derived directly from the current progress of the development. We have divided the development progress into three parts: (i) planning period, (ii) programming period and (iii) testing period. In the present case the planning begins with the discussion of the requirements for the project and ends with deciding which computer program functions have to be provided. The programming begins with deciding how to realize the functions and ends after coding, compiling and debugging all program modules. The testing begins with binding all modules together and ends with an overall test of the functions provided in the program.

Usually bankruptcies are detected only during the testing period. In order to observe progress during the test period, a graph called the PB graph is introduced, which indicates the number of items that the check have not been carried out and the number of bugs detected versus the number of working days elapsed.[5] The check items are defined in a document called the program check list (PCL). In the PB graph the number of check items that remain unprocessed in the PCL represents the remaining amount of work, and this number decreases as the number of working days increases.

For preventing bankruptcy the managers usually have to pay attention to the quality of the software under development, and the contents of the PCL have been set so as to satisfy the quality level required in the market. Some managers claim that they can not judge their progress in the test process before ascertaining the testing reliability.

Project management in general has many aspects as quality management or schedule management. Ideally speaking, schedule management should be independent of the other management activities. Meanwhile, the quality management embedded into the schedule management tends to decrease the independence of both ones. This makes it impossible to compare schedules of projects with different quality managing strategies in progress. Consequently we have not been able to gather data about schedule management in which the quality management is embedded.

We made the assumption that there can be an investigation on schedule management which is independent of the quality management progress and also of the design methodology. This paragraph deals with the PCL from the scheduling point of view and it was assumed that managers are willing to try their best to maintain the contents of the PCL at a reasonable level using their experience in product assurance.

III-2 Interpretation of the PB Graph

Figure 1 shows a PB graph. The number of items that remain unprocessed in the PCL is represented on the vertical axis and the number of working days on the horizontal axis. Some items in the PCL may contain rather easy checks, others may be elaborated. We assume that the tests are carried out in a sequence that is independent of the complexity of the test items. Then the number of unprocessed check items, which is denoted by NCI, in a PCL can be regarded as representing the amount of remaining work, just as the number of the lines of code represents the program size, and when the NCI is lowered to zero the testing is over. A bug detected has to be removed and the program has to be tested again after the removal of the bug. After the removal of a bug and its corresponding test are finished, the test item is deleted from the PCL. Thus the overall progress of testing is represented by the NCI on the PB graph.

The PB graphs, obtained from many projects, show a common tendency which can be represented by a curve with solid three-line as shown in Figure 3. Thus the period can be subdivided into three parts. During the first period, the program to be tested satisfies the PCL items at a very low rate because in general every part of the program cannot be executed completely. As the fixing of bugs goes on, debugging begins to permeate every part of the program, thereby enabling the program to prepare the test conditions, and consequently accelerating the rate of accomplishing the PCL items. This is the 2nd period indicated by t_2 in Figure 3. At the end of t_2, it is difficult to accomplish the remaining check items in the PCL, because of, for example, timing bugs may rarely reappear or bugs may emerge only under very special conditions. Thus the rate of decreasing the NCI slows down again during the last period shown as t_3 in Figure 3. Next we will introduce a method by which the solid three-line in the PB' curve model can be derived from the PB curve.

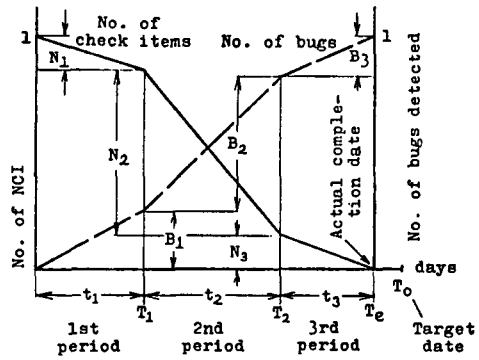

Figure 3 PB' Curve Model

III-3 Polynomial Approximation of the PB Curves

The PB curve is not smooth because the NCI decrement varies widely from day to day. Therefore, it is necessary to smooth this curve in order to make macro judgement on it. The intent of the approximation results in repeatable curve fitting, general grasping and the form comparability of the PB

curves. A polynomial approximation of the 6th degree is performed by feeding the coordinates of the points on the PB curve into a computer. Figure 4 shows a polynomial approximation of a PB graph. These curves are referred to as PB' curves in this paper. The PB' curves give a normalized value for the NCI, bugs and test days.

We normalized the NCI-time curves in the PB' graphs in such a way that the initial NCI value becomes equal to 1. Since the total number of bugs cannot be estimated during testing, the current cumulative amount of bugs detected is normalized and is put equal to 1. Therefore, at the end of the test period the bug-time curves in the PB' graphs give normalized values. The normalization of the curves implies the ratio of the number of bugs detected before a specified day to the total number of bugs detected in the whole test period. The number of test days also has to be normalized. This has been done by setting the number of days from the beginning of the test untill the target date equal to 1. These normalizations provide a common basis for comparison of the PB' graphs of various projects.

A differentiated curve for a NCI-time curve is referred to as a PB'' NCI-time curve as shown in Fig. 4. Let T_1 be defined by

$$PB''(T_1) = -\frac{1}{2}\frac{NCI_0}{T_0}$$

here the NCI_0 denotes the initial NCI value, T_0 the number of days of the planned test period. Therefore, NCI_0/T_0 indicates the average rate of the NCI reduction, and T_1 is the period of time required for the reduction rate to become 50% of the average rate of the NCI reduction. At T_1, the project is said to enter into the 2nd period. The PB'' NCI-time curve keeps decreasing for a while after T_1, and approaches to zero at the end of the 3rd period. The PB'' NCI-time curve again intersects the line $y = -\frac{NCI_0}{2T_0}$ at T_2 marking the end of the 2nd period.

III-4 Data Normalization

Let the NCI reduction in the 1st, 2nd and 3rd period be n_1, n_2 and n_3 respectively, and let N_i $(i = 1, 2, 3)$ be defined by

$$N_i = \frac{n_i}{NCI_0}$$

Just as with the NCI, the number of bugs detected in the 1st, 2nd and 3rd periods are given by b_1, b_2 and b_3 respectively and B_i $(i = 1, 2, 3)$ is defined by $B_i = \frac{b_i}{b_1+b_2+b_3}$

The data collected are processed to draw PB' and PB'' curves and the number of the NCIs and bugs detected can be obtained. The number of NCIs at T_1 is $(1-N_1)$. At T_2 $1-(N_1+N_2)$. The number of bugs detected at T_1 is B_1 and at T_2 (B_1+B_2). Thus the PB graph of each project is converted into a PB' curve model as shown in Figure 3.

There are two ways to normalize the number of working days. To investigate the nature of a software development, it is convenient to set the actual number of days used (T_e) for testing equal to 1. If the actual number of days is not known yet, the number of days from the beginning of tests until the target date (T_0) has to be set equal to 1 (See Figure 3). Table 1 shows the average values of all the data for the twenty three projects.

The normalized length for each period is denoted as t_i (i=1,2,3)

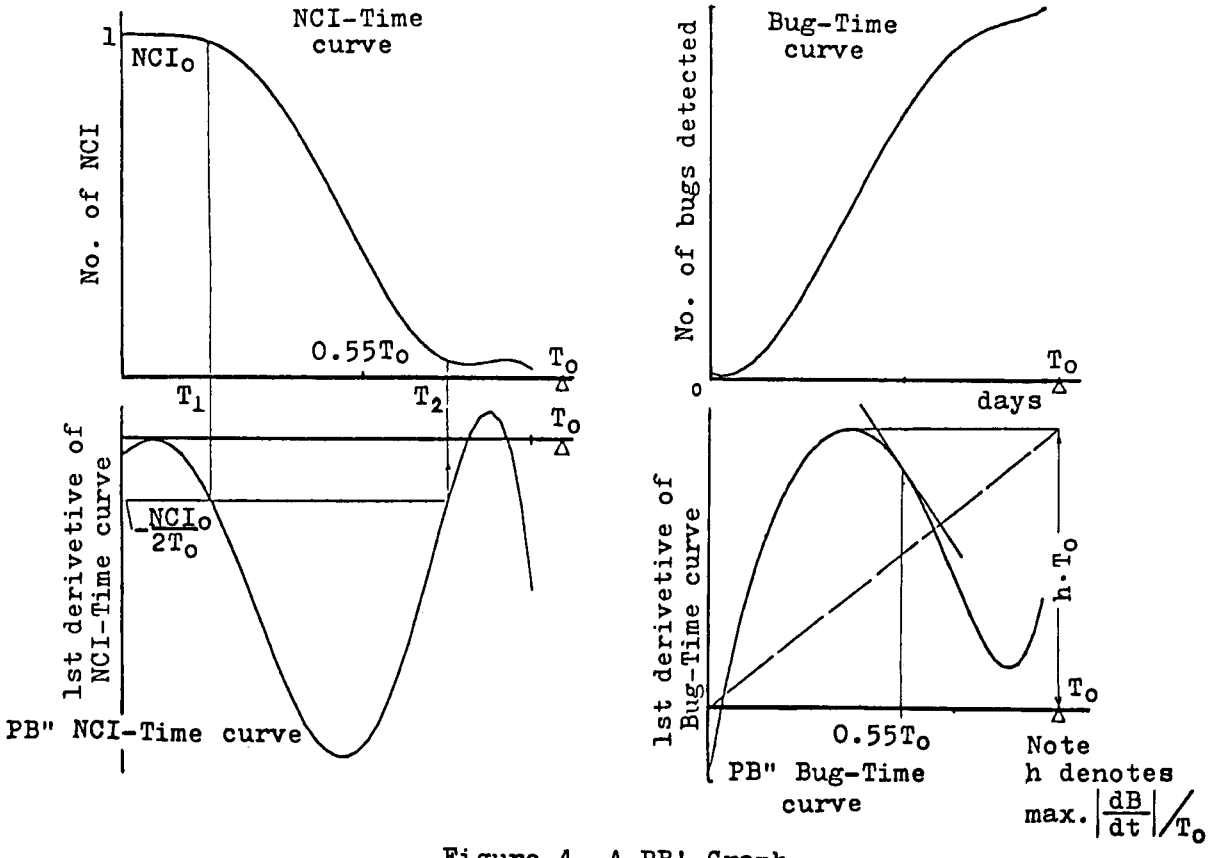

Figure 4 A PB' Graph

and is given as follows.

$$t_1 = \frac{T_1}{T_e}\ ,\quad t_2 = \frac{T_2 - T_1}{T_e}\ ,\quad t_3 = \frac{T_e - T_2}{T_e}$$

IV. RELATION FOUND BETWEEN NCI, BUGS AND WORKING DAYS.

Study of the data shown in Table 1 revealed the following relations between the three variables mentioned above.

IV-1 Characteristics of the 1st Period.

(1) The debugging efficiency

The value B_1 shows the ratio of the number of bugs detected during t_1 to the total number of bugs detected by the end of testing. It may be regarded as an approximation of the debugging efficiency without taking into account bugs occurring after the delivery of the program to users. In this section, all values discussed are the average values unless otherwise specified. The average value of B_1 for all projects being studied becomes 0.218 as indicated in Table 1a.

According to the definition in III-1, every module has to be debugged before testing begins. Then, the bugs removed during this period can be considered to be located in restricted parts of the program. Every part of the program would not be scanned for debugging in this period. The number of bugs of this kind amounts to about 20% of the total number of bugs throughout the test.

(2) Difficulty in detecting bugs

The detection of bugs in the 1st period is not as difficult as that in the 2nd period. According to Table 1b, the average bug detection rate B_1/t_1 is equal to 0.8776. This is 67% of the value of B_2/t_2 in the 2nd period. This rate variation is far less than the change of the NCI reduction rate over both periods. The latter is 6.8 times faster than the former.

These facts lead to the presumption that for an average software development the bug detection rate, or the skill in examining the test output, is a particular value for the project team, and is nearly constant throughout the whole test period. This rate can be more or less affected by the number of test items applied to the program in a certain unit of time, i.e., the NCI reduction ratio.

(3) The Duration of the 1st Period.

The bugs detected in the 1st period are relatively easy to fix. These bugs are located in the restricted areas of the program as mentioned before and are normally limited to interface bugs, because modular debugging has already been finished before testing. Thus, they can be detected efficiently by reviewing the codes in the restricted areas.

The average value for t_1 is equal to 0.208, and 0.208 times the total test cost gives the upper boundary for the profit obtained by this review.

Table 1a Average Data Values Derived from the Model

Items	B_1	t_1	B_1/t_1	N_1/t_1	N_2	B_2	B_2/t_2
Values	.2178	.2048	.8776	.1343	.8847	.6531	1.307

Items	N_2/t_2	B_3	B_3/t_3	N_3/t_3	B_1/N_1	B_2/N_2	B_3/N_3
Values	1.720	.1553	.7081	.3212	11.2	.8009	2.700

Table 1b Average Data Values in the 2nd Period

Average	For All Projects	For Regular Projects	For Semi-Regular Projects	For Bankrupt Projects
t_2	0.5735	0.6533	0.5679	0.2881
$D.E_2$	0.8350	0.8900	0.7335	0.6368

Note

$D.E_2$ denotes debugging efficiency during 2nd period i.e. $\frac{B_2}{B_2+B_3}$

IV-2 Characteristics of the 2nd Period.

(1) Rate of the NCI reduction

Table 1a gives the NCI reduction rate as 1.72, 12.9 times that of the 1st period.

(2) Debugging efficiency in the 2nd period

The remaining work is represented by the remaining NCI. Work done per a certain unit of time (also a certain unit of cost) is dependent on the NCI reduction rate. This rate is maximal during the 2nd period. It is desirable that all bugs that have not been found at T_1 be detected during the 2nd period. The 2nd period debugging efficiency is defined as B_2/B_2+B_3. The average value is shown in Table 1b. For obtaining the average values we took account of all projects, regular projects, semi-regular projects in which the manager detected a trend toward bankruptcy and succeeded in overcoming the crisis, as well as bankrupt projects. The average value for this efficiency of the semi-regular and bankrupt projects was 0.714. The average value for the regular projects was 0.89. Therefore, there were 2.6 times as many bugs left in the semi-reguiar and bankrupt projects as in the regular projects.

(3) The duration of the 2nd period

The average values for t_2 are given in Table 1b. The average value for the regular projects is 0.5753, while for the bankrupt projects, it is 0.288.

This shows how short the highly efficient period in a bankrupt project is.

(4) The coefficients of variation

The coefficients of variation for NCI reduction rate N_i/t_i ($i = 1,2,3$) and for detection rate B_i/t_i ($i = 1,2,3$) are listed in Table 2 for all projects. The coefficients of variation for N_2/t_2 and B_2/t_2 are minimal during the 2nd period. Thus, the attempt to extract common characteristics by means of a PB' curve model is most successful during the 2nd period.

Table 2 Average Values & Coefficients of Variation for Test Periods

period	Average Value N_i/t_i	Average Value B_i/t_i	Coefficients of Variation of N_i/t_i	Coefficients of Variation of B_i/t_i
1st	0.1343	0.8776	1.225	0.6508
2nd	1.720	1.307	0.3330	0.2672
3rd	0.3212	0.7081	0.8453	0.9532

IV-3 Characteristics of the 3rd Period.

(1) NCI reduction rate

Table 1a gives this reduction rate as 0.321, 0.186 times that of the 2nd period.

(2) Ratio of NCI reduction and bug detection rates

The ratio of the number of detected bugs to the NCI reduction can be regarded as the "bug hit ratio". Table 1b gives these values for the three periods. B_3/N_3 is equal to 2.7, 3.4 times that of B_2/N_2.

IV-4 Bankruptcy Occurrence Detection

(1) Using T_1 value

T_1 is the time at which the NCI reduction rate becomes half of the average NCI reduction rate. If the value T_1/T_0 tends toward 1, bankruptcy will probably occur. T_1/T_0 values for the regular, semi-regular and bankrupt projects are shown in Figure 5. Based on experimental results, 0.55 was found to be the critical value of T_1/T_0. Bankruptcy is forecasted as being inevitable, whenever T_1/T_0 exceeds this value.

Next, the detection method described above is evaluated. The average T_1/T_0 value for the regular and semi-regular projects is 0.146, while that for the bankrupt projects is 0.966. The estimated values for the T_1/T_0 standard deviation are equal to 0.140 and 0.659, respectively. The probability of misjudging a trend toward bankruptcy in a regular or semi-regular project can be estimated to be of the order of 0.5%.

On the other hand, the probability of misjudging that bankruptcy will not occur in a bankrupt project is estimated to be about 24%. The latter probability is so high that a method of improving pertinent determination has to be found. One possible method is as follows:

(2) Using bug detection rate variation

If the bug-time curve derivative for a PB' graph is decreasing, while the NCI-time curve reduction rate is accelerating, then bug density is decreasing. Therefore, the outlook for improving the program quality is promising. Bug-time curve 2nd derivative values on the PB' graphs were determined at $t = 0.55T_0$. Then these values were normalized to the maximum (occurring before $t = 0.55T_0$) of the 1st derivative's absolute value for the bug-time curve divided by T_0, i.e., max. $dB'/dt \frac{1}{T_0}$. These values are shown in Figure 6. +0.3 was found to be the critical value for this normalized value. Bankruptcy is forecasted as being inevitable, if the normalized value exceeds the critical value.

There are thirteen cases below the critical value in Figure 6, including one bankruptcy. The probability of not being able to predict bankruptcy occurrence is expected to be of the order of 8%.

By combining these two methods, the probability of not being able to predict bankruptcy occurrence will be reduced to the order of 1.8%.

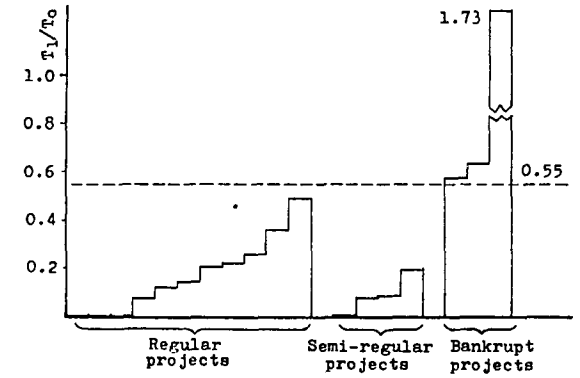

Figure 5 Distribution of T_1/T_0

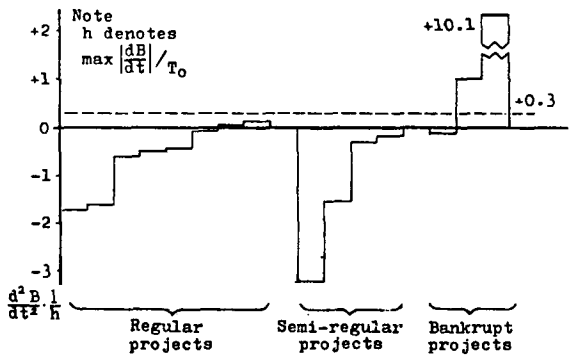

Figure 6 Distribution of $\frac{d^2B}{dt^2}/h$

V. BANKRUPTCY CAUSES ANALYSIS

By analyzing the different bankruptcy causes it is possible to collect information about how to handle similar cases. Moreover, by collecting data on bankruptcy causes, classifying them into a number of categories and extracting common properties, it is possible to obtain new ideas that can be helpful in project management.

Bankruptcy causes generally include many factors. As an example of what can happen, the following case is presented. The manager is not expecting bankruptcy, judging from the progress report. However, halfway through the development he may discover to his embarrassment that the number of lines of code has become twice the estimated amount. Meanwhile, one of the members of this project team has no experience in this particular application field, and he may not be aware of the existence of test cases which should be considered. This may cause a delay in detecting problems. Therefore, just before the target date modification of a thousand lines of code may be found necessary.

Bankruptcy causes were analyzed by separately considering managers, members and environment. The environmental factor is subdivided into two independent factors: interference from other projects and whether the software is new or whether it is a reconstruction of previously used material. Each of these factors that can cause a bankruptcy is shown in Table 3. This information was obtained by interviewing the managers of projects checked during the present study. The interviewer selected was to be an experienced software projects manager and to have a close personal relation with the managers to be interviewed, so that frank discussion could result. Thereby, the underlying reasons for failure should appear. The managers were asked to choose from the failure reasons listed in Table 3 exactly one for each factor, which they thought to be mainly responsible for bankruptcy occurrence.

Table 4 gives a numerical analysis of elevan reasons for complete bankruptcy or near bankruptcy. It appears that the majority of managerial reasons are misjudgments. The reasons ranked next are far less frequent. Even though the number of bankruptcy projects is fairly limited, it seems recommendable

that managers should pay more attention to possible misjudgments than to other reasons in Table 3, provided that this sample represents the general situation in the software industry. Table 4 shows the number of projects that suffered from other circumstances (e.g., inexperience on the part of members) within the number of projects that failed because of some managerial mistakes. There are only two projects without managerial mistakes; both claimed inexperience on the part of the members on their team and neither experienced interference from other projects.

For the most frequent case, that is misjudgment, the relation to other reasons is shown in Table 5 in order to indicate under what circumstances managers are likely to fail. Four cases out of six use reconstructed material. Each of these cases claims that there was interference by manpower and feeding requirements from other projects. The 6th column in Table 5 shows the ratio of number of projects using reconstructed material to the number of all projects (these eleven projects in the table were developed by the same company during the same time period). The 7th column in Table 5 shows the same ratio for completely bankrupt and near bankrupt projects.

It can be observed that for those projects using reconstructed material and suffering from interference, managers should try to prevent misjudgment like those listed in Table 3. That is, managers should check the basis of case study, fact findings and of judgment on optimistic reports. Timely judgment should be accomplished on the basis of the actually observed data and decisions must be made which will forestall a bankruptcy, once such a trend becomes evident.

VI. CONCLUSION

This paper analyzed the possible causes of bankruptcy for a wide variety of software development projects. A method was proposed for detecting bankruptcy occurrence, aiming at reducing or avoiding losses that a bankruptcy entails. Development process was divided into planning period, programming period and test period. Twenty three projects were studied which together contain a million lines of code, taking into account that most bankruptcy occur during the test period. Then, a description on how relations between the number of items in the program check list, the number of bugs detected and the number of working days elapsed during the test period, could be converted into a simple model. Using this model it became possible to detect a tendency toward bankruptcy before they actually occurred. Further, nine test period characteristics for a software project were pointed out that could be helpful in increasing planning reliability. Next, the causes of complete bankruptcy or near bankruptcy were analyzed and the causes were traced back to particular reasons. Circumstances under which managers are likely to fail were discussed.

If it is possible to judge, before the beginning of the test period, whether or not there is a strong possibility that a project might go bankrupt by the check stated in chapter V, then

Table 3 Factors Pertaining to Bankruptcy

Factors	Reasons for failure (or the status of) : Definition
Manager	Inadequate planning : incorrect estimates : failure to plan a test : delay in fixing the task description Misjudgment : inadequate case study : insufficient fact finding : judgment based on over-optimistic reports Inadequate leadership : lack of leadership : over-control No mistake : according to the manager, there are no mistakes
Members	Inexperience Over 50% Of members are : inexperienced in this particular application field. : inexperienced with the resources used. Experienced : Over 50% of members have experienced in this particular application field and with the resources used.
Interference from other project requirements	There was interference : This project team had to assign some of its members to other projects in trouble. Therefore, project start was delayed more than 10% of the total period. There was no interference : Interference was less than 10%.
The software is new or reconstructed	Reconstructed : making use of other programs already developed, reconstructed to meet project requirments New : new development

Table 4 Distribution of Reasons for Bankruptcy or Near Bankruptcy

Managerial Reasons	Number of Projects in Trouble		Number of Projects in Trouble					
			Members are		Interference		Software is	
			Experienced	Inexperienced	Yes	No	New	Reconstructed
Inadequate Planning	2	18%	1	1	1	1		2
Misjudgment	6	55%	2	4	4	2	2	4
Inadequate Leader-ship	1	9%		1		1	1	
No mistake	2	18%		2		2	1	1

Table 5 Relation Between Misjudgment and Other Reasons

Software is	No. of Cases	Members are		Interference		Ratio of the Number of the Projects		
		Experienced	Inexperienced	Yes	No	For All Projects	For (Near) Bankrupt Projects	For Misjudged Projects
New	2	2			2	53%	36%	33%
Reconst-ructed	4	2	2	4		47%	64%	67%

the managers can take the necessary steps to minimize any financial and manpower losses. In the test period the method of detecting bankruptcy can be applied. Then, the software builder and the manager of a big project can reduce the probability of suffering losses due to software development uncertainty.

ACKNOWLEDGEMENT

We wish to thank to Mr. W. Sakamae and our colleagues at Keio University for their helpful discussions. We also want to acknowledge the support of the managers of Hitachi Software Engineering Co., Ltd. for this study.

REFERENCES

1 L. H. Putnaum : "A Macro Estimating Methodology for Software Development", COMPCON-FALL 1976, pp.138–143.
2 M. K. Starr : "Systems Management of Operations", pp.216–218, Prentice Hall Inc. Englewood Cliffs, N.J., 1974.
3 T. C. Johnes : "Measuring Programming Quality and Productivity", IBM Systems Journal, Vol.17, No.1 pp.39–63 1978
4 F. Brooks : "The Mythical Man-Month", pp.14 f. Addison-Wesley Publishing Co., Reading, Mass.
5 Kanji Shibata : "Improvement in Software Development Management" Tekko no I.E. Vol.15-4 pp.23–27, 1977 (in Japanese).
6 G. M. Weinberg : "The Psychology of Computer Programming" pp.100–112, Van Nostrand Reinhold Co., New York, N.Y., 1971.
7 Ministry of Finance Report : "Report on Stocks No.11–17 Nippon Steel Co.", 1978 (in Japanese).
8 Ministry of Finance Report : "Report on Stocks No.11–4 Nippon Kokan Co.", 1978 (in Japanese).