ON CODING AND UNIT-TEST PROCESS MANAGEMENT FOR SOFTWARE DEVELOPMENT OF LARGE-SCALE

Yasunobu Kino
IBM-Japan
Roppongi 3-2-12, Minato, Tokyo, Japan
ykino@jp.ibm.com

Abstract

To manage a phase of coding and unit-test, project managers have used to pay attention to a number of completed programs. And the manager makes a graph of progress. Usually, this graph of progress has S shape and doesn't linearly depend on the workload. So the degree of progress seems to be behind. In actual, many projects tend to be behind the schedule. Because of this reason, it is difficult to judge whether the project is behind or not in the early stage. In this paper, We propose the "four-division model" to solve this difficulty.

1. Introduction

It is generally known that progress of coding and unit-test gives an S shape. (Figure 1) Some articles have been devoted to study of this fact. [1] For example, Abe [2] studied this S shape. And he proposed to take account of the number of on-coding programs, to monitor the degree of progress. In this study, We assumed this S shape due to the improvement of working procedures and the growth of human skill. In the following section, We show data and analysis.

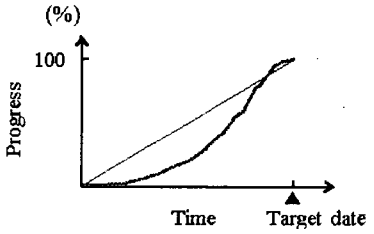

Figure 1: A sample of progress rate

2. Data and analyses

On-line system development data of one of major Japanese City banks was used. This project used PL/1. And five teams were organized by each application such as exchange, loan, finance, and so on. There is no administrator and worker overlapped. Each team worked on the same environment of host computer. And the progress was measured as the number of programs finished cording and unit-test.

Graph 2 shows the progress rate of each team. And this graph is modified as follows.

(a) In actual calendar, from April 29 to May 5, there were many holidays in two weeks. In this graph, we count those weeks as one week workload.

(b) We collected numbers on Mondays. Some Mondays have no data because of holidays. In this graph, we take an average of previous Monday and next Monday when there is a

holiday.
Additionally, the team C, D and E were added developers to catch up the planed schedule.

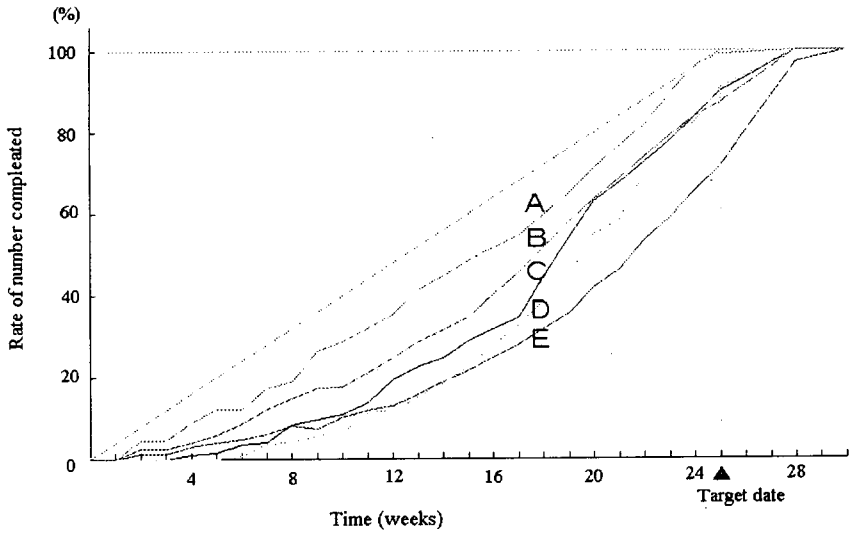

Figure 2: The accumulated production of each tesm

(The weight of programs and the fact of the holiday attendance aren't taken into consideration in this data.)

In figure 2, it shows number of accumulated production for each team. The schedule tends to behind in early days. And if a speed of the progress will be the same, we can't finish up the programming in the end. But if a speed of the progress grows rapidly, will catch up the schedule. This fact shows that the productivity will be changed as the work proceeds. Next, we calculate the amount of weekly production to confirm previous fact. (Figure 3)

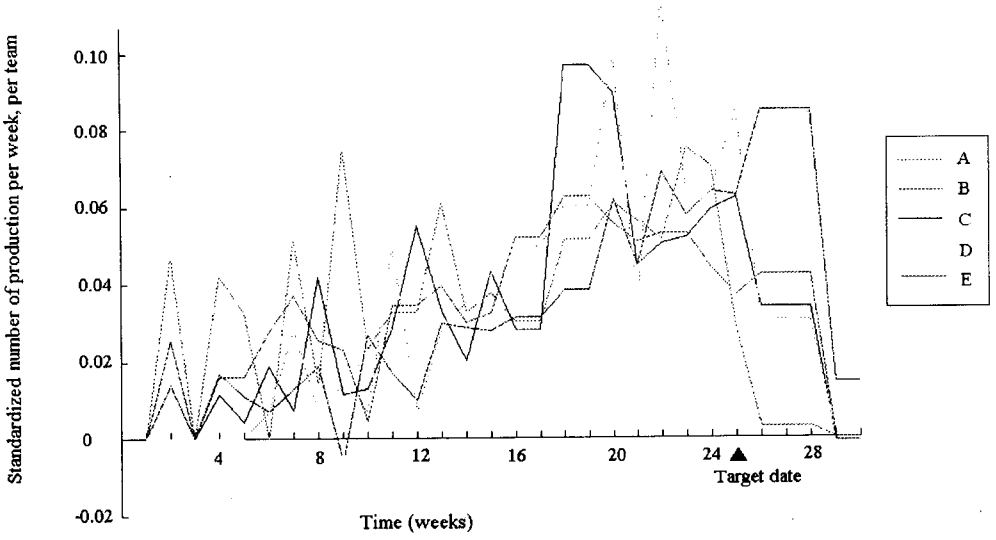

Figure 3, Productions per week

In figure 3, we can find a growing productivity trend in spite of a zigzag movement. To make it linear, we count the number by every four weeks. (Figure 4)

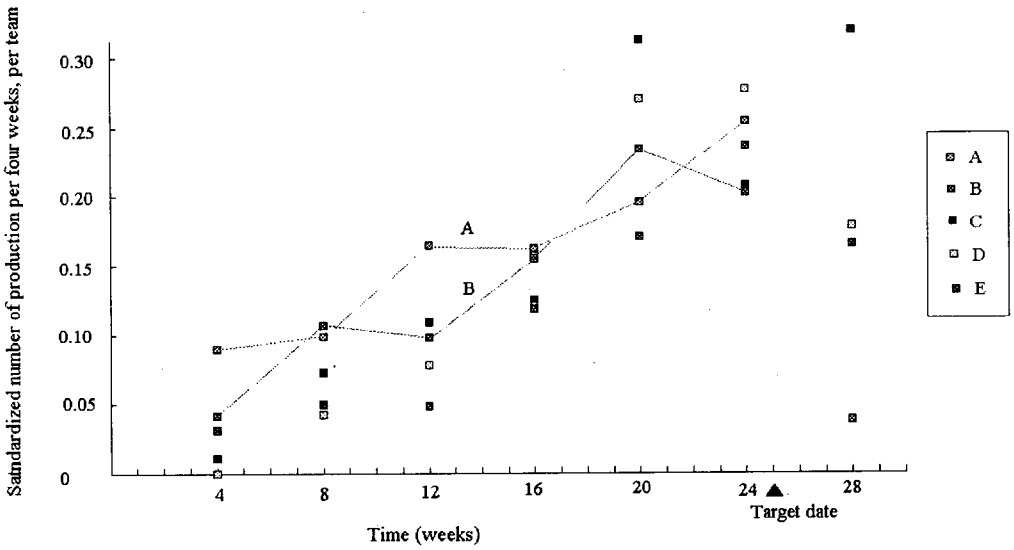

Figure 4: Productions per four weeks

In figure 4, we make lines for team A and B since those teams don't add additional members. And from this figure, we find productivity of each team grows.

3. Proposal

3.1 The four-division model

From previous data and our experiences as members of this project, We found there are differences in a nature of works among "Beginning", "Middle" and "Ending" of the process. So, We propose that the process should not be considered as one continuous process, but as four different parts.

(1) First part (Until the first program is completed)
We need more time than we expected when we use new computer environment at the first time. We often find errors of the environmental step and the inferior working process at this stage.

(2) Second part (Productivity improves by the improvement of working procedures)
The working environment of the system is almost completed after the first program. And we find many more problems on our procedures of coding and unit-test. So we try to improve procedures to work well. By improving procedures repeatedly, the productivity grows dramatically. We end the second part when the document of working procedures becomes a final edition.

(3) Third part (The improvement rate of productivity is settled)
After documents of working procedures are finalized, we concentrate on the production. This is the third part. The human learning skill contributes, the most, to productivity in this part.

(4) Forth part (A convergent part)
Though it tends to be forgotten, there is a part that reduces the productivity to the end. Reasons are followings.

- There are many members without the program. Since the number of uncompleted programs are shorter than the number of developers.
- Some developers start a preparation for the next phase.

Figure 4 shows the productivity of each part.

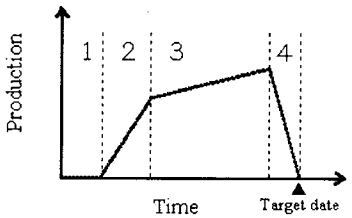

Figure 5: The production of four-dividing model

3.2 The estimation using the four-division model

Commonly, to estimate the necessary number of developers for a coding and unit-test plan, we calculate the number of developers by following formula.

$$Number\ of\ developers = \frac{number\ of\ necessary\ programs}{productivity(per\ person,\ per\ week) \times whole\ terms(weeks)}$$

We estimate productivity (per person, per weeks) by thinking of the efficiency when developers become tame enough through our experience. Though, productivity is not constant in actual. So that, in many cases, projects become death marches. To avoid this problem, We propose the following procedure using four-division model to estimate necessary number of developers more precisely.

[Procedure]

(1) First, keep two weeks each for the first and second part. And, keep another two or one weeks for the fourth part.

(2) Estimate maximum productivity "b", which is usually when the workers are well experienced. And also estimate minimum productivity "a", which is usually the beginning of the third part. If minimum productivity "a" is hard to estimate, use the next table for estimating "a".

The length of third part	4 month	5 month
The value of "***a***" when "***b***" as 1	0.5	0.4

Table 1: Estimation of minimum productivity "a" from maximum productivity. That estimation was done by graph 4.

(3) Assuming linear growth of the productivity at each part, we achieve the change of the estimated productivity through the phase.

(4) Calculate the area under the productivity line; in order to estimate the number of produced programs per developers of this phase.

(5) Finally, we calculate the necessary number of developers by dividing the necessary number of programs of this phase by the number of produced programs per developers, i.e.

$$Necessary\ number\ of\ developers = \frac{necessary\ number\ of\ programs\ of\ this\ phase}{number\ of\ produced\ programs\ per\ person}$$

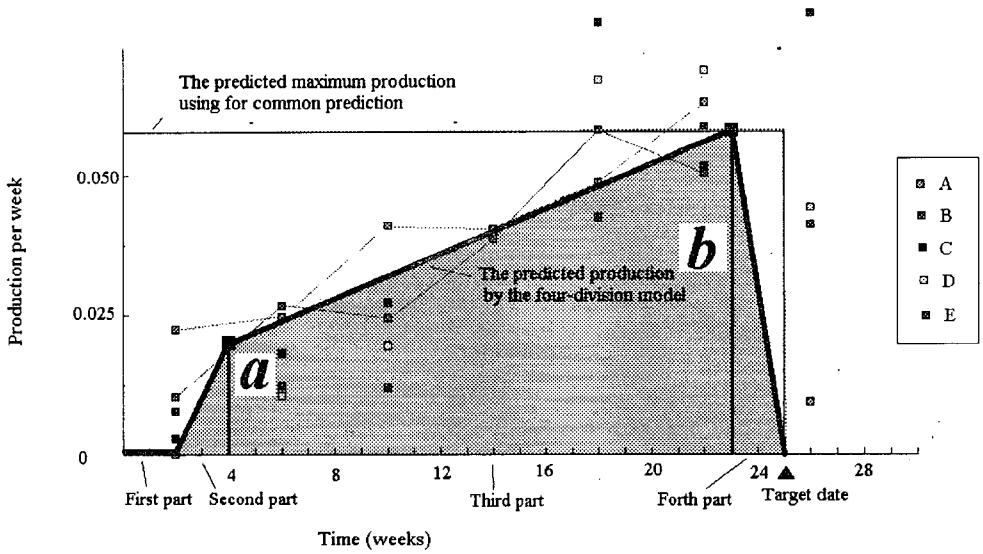

Figure 6: The productivity of four-dividing model on the case project

3.3 Suggestion for management

On managing the coding and unit-test phase, we always pay attention to a number of finished programs through coding and unit-test phase. However, if by this four-division model, we can make a different approach. We can summarize as follows.

(a) Managing first part: In order to complete the first program as soon as possible, the most important task is to find and solve the system environmental problems.
(b) Managing second part: Improve procedures to developers work efficiency, and educate the participant.
(c) Managing third part: Set each goal for the participant. Don't change any working procedures. And the manager should start thinking about next phase.
(d) Managing fourth part: If there is no delay at the end of the third part, it is good to give a few days off to those who finished their own job for morale.

Since management items become clear by using the four-division model, it is possible to manage the project in details.

4. Conclusion

To estimate and manage a project better, We propose the four-division model. In this paper, this model is discussed for large-scale project and a phase of coding and unit-test. But this model is not limited to only such situation. If there is a process that many people do repetition work, this way of thinking can be applied. And we would like to study further, how productivity growths by learning are different among each other.

References

1. Software Product Control Research Group of JUSE (ed.), *Advances in Japanese Software Quality Assurance*, JUSE Publishing, pp.565-595, 1994.
2. Akio, Abe, "A New Proposal for the Progress Management of Software Development Projects – To Grasp the Actual Progress", *Journal of Information Processing Society of Japan* Vol.36 No.6 pp.487-492, 1995