New Trends in Software Methodologies, Tools and Techniques
H. Fujita and D. Pisanelli (Eds.)
IOS Press, 2007

An Introduction to the Quantitative, Rational and Scientific Process of Software Development (Part 2)

Zenya Koono[a,1], Hui Chen[b] and Hassan Abolhassani[c]
[a] *Creation Project, Kanagawa, Japan*
[b] *Information Science Center, Kokushikan University, Tokyo, Japan*
[c] *Computer Engineering Dept., Sharif University of Technology, Teheran, Iran*

Abstract. "Product" and "process" are two mutually orthogonal and important aspects of software. As Part 1 discusses "product", Part 2 discusses "process". "Process" is independent of "product" and is used commonly by various "producers". This paper first explains the structure of "process" and then introduces important technique by "divide and conquer". Various important management issues are discussed.
Keywords. Process, Product, Productivity, man-hours, Defect intensity, Industrial Engineering, Learning Effect, Human intelligence

Introduction

In software, there are two aspects, namely "product" and "process". "Product" and "process" are orthogonal concepts. 'The standpoint of both Part 1[1] and 2 is human intentional activity ranging from physical activities to mental operations in design and management. In Part 1, the structure and the quantitative characteristics of a "product" have been shown, which may be applicable in any "process". In Part 2, the main theme is "process[2,3,4]", which is a commonly used technology, for any "product". It is discussed from the viewpoint of knowledge, which is common to all of the physical activities and mental operations in design and management.

Eventually, the "process" in this paper is the same as that in Industrial Engineering (IE) [5]. This means what has been established in IE on an empirical basis agrees with what is concluded here from the aforementioned theoretical viewpoint. Thus, "process" here is the same as that in industry in general. "Process" is the method for achieving manufacture, commonly used by "products," from the viewpoint of management. Thus, it is different from the so-called Software Process or Work Breakdown Structure.

Section 1 explains what is process, and Section 2 discusses how "divide and conquer" is used with "process", or how management uses it. Section 3 explains how the characteristics are used in a process, and Section 4 discusses the various important points for developments, including management.

[1] Corresponding Author: Representative, Creation Project, Honfujiswa 2-13-5, Fujisawa, Kanagawa, 251-0875, Japan; E-mail:koono@vesta.ocn.ne.jp.

1. The structure of process

In the early age of software, a software development process used the same names as in hardware. As hardware production had preceded software development by at least half a century, the process had penetrated deeply into society. Furthermore, software design, and not only hardware design but also hardware production work, shared the same structure of human intentional activity, discussed in Part 1. It was only natural for software development to stand upon the process technology developed in hardware, as its basic form.

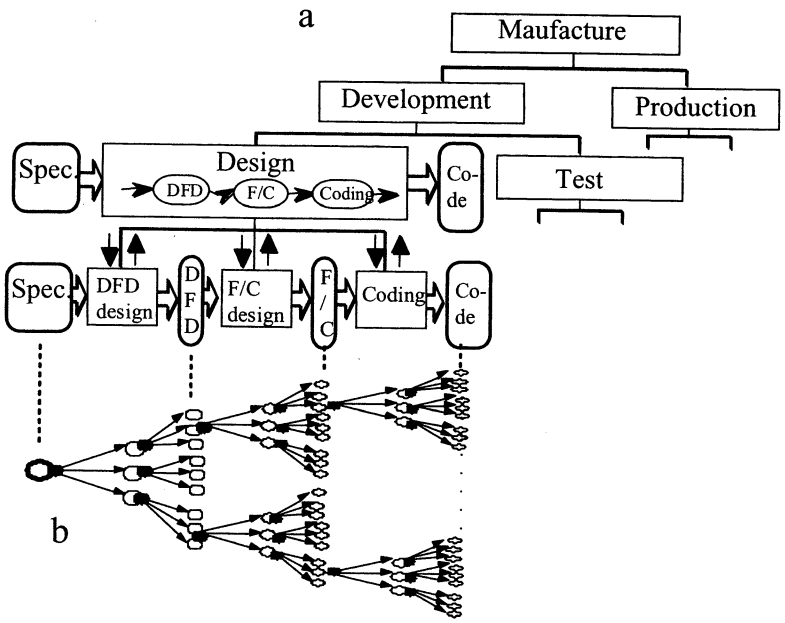


Figure 1. Structure of process

The upper part of Figure 1[3] is the upper part of a "development" process, which is hierarchically structured in the customary manner from IE. "Manufacture" at the top of Figure 1, is decomposed. "Development" is a flow of work to complete design documents[6] (including source code list) for the specification, while *"production" is a repetitive work making identical products*. Further downward, the "design" process is broken down to "data flow design" process, "flowchart design" process and "coding" process, where each interface is chosen to be abstracted and simplified, as in the case of a hierarchically structured software system. *Both interfaces of a process are defined by documents.* A process includes people who are responsible directly for the work.

The hierarchically expanding network at the bottom of Figure 1 shows the design from the specification to the data flow design, flowchart design and coding, correspondingly to the lower process below the "design". (It represents the "product".) The "design" commands "start" at a lower process, and on receiving the "finished" command advances to the next one. By a sequence of clear commands and their responses, the work advances reliably.

The hierarchical processes in Figure 1 constitute a hierarchically expanding network, starting from a final object "manufacture", which agrees with the empirical principle, "Hierarchy of Object[7]" in military science. Similar to a military organization, the usual organization structure for production[2] is a hierarchical one adapted to the process structure. The aforementioned conclusion is supported also by

[2] In the early days of IE, there was a dispute about what is the optimum team organization, whether it is hierarchical or not. Actual work was conducted in both ways, and the comparison proved that a hierarchical team is better. From industry to public society, this spread so that the hierarchical organization structure is a basic one now, and the aforementioned high reliability of work, ease of communication and speed of decision making show the superiority of the hierarchical organization structure.

this empirical rule. Thus non-hierarchical organization structures, such as a surgical team system, are special solutions not fitted for general use.

Based on the authors' studies for the automatic design of software[5], "manufacture" in Figure 1 is a human intentional activity. As the decomposition goes on, the work becomes clearer, more solid, more concrete, more detailed and finally it is reduced to the movements of the hands of a worker. The hierarchical decomposition chain starting from the final object "manufacture" is the knowledge of the "manufacture" process. *The hierarchical decomposing network is the knowledge of the "process".* Most of the working knowledge in human beings is a composite of hierarchical knowledge, as pointed out by Minsky in his book, "The Society of Mind". Cognitive science says that around 85% of human knowledge is of a hierarchical-type, and episode-type knowledge is few. Episode-type knowledge in "process" knowledge appears only in the state transition control sequence, as shown in the "design".

The following are keys for the "process" to work together with various "products." IE[3] has established the process concept:

- "Process" is defined at both of its ends (i.e. programming language and documents). It cannot be defined by procedures using natural language.
- "Process" consumes management resources as it progresses. Real time also elapses.
- "Process" does not get into any inside of a 'product'. It understands the inside of a product through measurements of the external characteristics of the process.
- "Process" is a means of control. It is attained through "divide and conquer".

As has already been discussed, "process" in this paper means activities of the people who have rights and responsibility for the work. Thus it is different from the Software Process or Work breakdown Structure, which is a mixture of "product" and "process".

2. Process technology using "divide and conquer"

The conversion from an input to an output belongs to the technology of a "product". The "process" technology is the control technology for better attaining the conversion, and the orthogonal design of the "product". In principle, this control may be applicable not only to physical works in hardware production but also to chemical processes such as in ceramics and biological processes such as brewing. The key point of managing any "process" is said to be "divide and conquer". Figure 2[2, 3, 4] shows the principle involved.

In the right side vibrating string-like figure, the amplitude shows the magnitude of the variation. A design process shown in the second level is constrained at two edges, and the variation becomes greatest at the center. In order to constrain these more severely, two intermediate documents, the data flow diagram and the flowchart, divide it hierarchically into three processes (data flow design, flowchart design and coding). As two intersecting points are added to constrain the vibration, the variation is decreased, as shown in the right side figure. If the variation is not yet sufficient,

---

[3] IE is also called Management Engineering, and systemizes the process technologies applied to the production control of hardware. It started as a cost reduction method in hardware work, and then Quality Control, using statistical methods, started in the early 20th century. It is a compilation of technologies for maximizing the profit of a business. The areas involved are production engineering as the measurement/method/ control/ management/planning of work, quality assurance and various optimization engineering. After 1970's, there was an important progress of Total Quality Control/Management in Japan.

another lower level process is divided into M (in the picture M=3.) processes by intersecting documents, resulting in a reduction to 1/M. If this is not enough, the process is further sub-divided. (It should be remembered that this is effective for internal variations.)

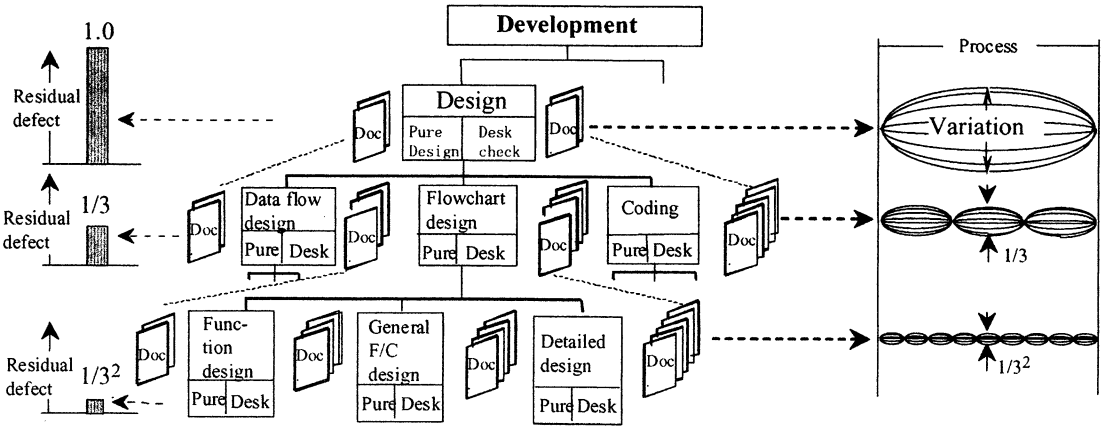
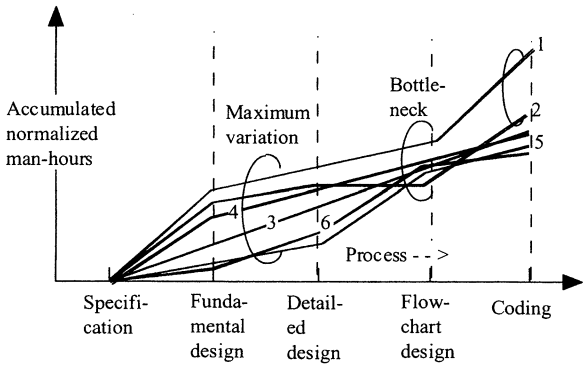


Figure 2. Divide and conquer

Although the principle is simple, it requires an effort to customize it to fit a project. Figure 3[5] shows a record of a first project using both complier language and structured design in the end of the assembler language age. The horizontal axis shows the progress of the processes until coding. The vertical axis shows the accumulated productivity (normalized man-hours/lines of code) of 6 teams as the design progressed. Although the number of processes was increased, the variation was still large. As the contour of curves show, intermediate constraining points do not work.



After studies, it was found that the large variation was caused by loose process definitions, which were just indices of each document. The new definitions were not only the indices but also each description standard and sample documents. These regulated everyone's documentation, and the variation was much decreased.

Currently, the best practices are eagerly discussed. It is feared that people might misunderstand that introduction immediately solves problems. If a new practice is introduced, usually it results in some pluses with a certain minus also. More important than the introduction is *the ability to study the problem, use quantitative measures as needed, and localize the cause and then to create some improvements. Namely, what is important is not to imitate, but to apply the principle in own environment.* Otherwise the introduction of this new method does not bring substantial improvements.

Next, regarding the desk checks or quality improvement using "divide and conquer" shown in Figure 2, the manner to remove defects more effectively by desk checks, performed after design, is explained. In Part 1, it was pointed out that any defect removing methods attenuate defects.

In a design on the second level of the figure, after the "pure design", a desk check is made. Let us assume that the defect is random and the defect intensity is D (defects). Let us assume that the probability of the second kind of error is C. The residual defect intensity of the checked program is D·C (defects). In the third level, the program is divided to three of thirds of the entire program size, and each third contains defects of D/3. The errors of each desk check are decreased to 1/3 or the probability is decreased to C/3. Thus three time checks of 1/3 programs are 3 x D/3 x C/3 = D·C/3. If it is divided to M sections, then it is D·C/M. For further decreases, one of the divided processes is further divided to N pair of pure design and desk check. The defects are decreased to D·C/M·N. Bar charts from the second level show this.

The desk checks are effective as the dividing number is large. When thus divided sections are small, or the conversion from the input to the output of the section (or conversion distance) is small, the change is small. In actual desk checks, the check is very easy, and the result is reliable. The following are two examples, each with a different conversion distance.

The first example is that of a telephone switching system. This kind of system is currently called an embedded system. It may be modeled using a Finite State Machine, FSM. A FSM is implemented by memory for remembering states and combinatorial logic for advancing states and for generating necessary signals. When an event driven OS is used including state memory, a state transition route program corresponds to the combinatorial logic. As this system is complex, it is divided into two mutually orthogonal subsystems FSM_A and FSM_B. Both were developed using the same architecture. They were developed by team A and B respectively.

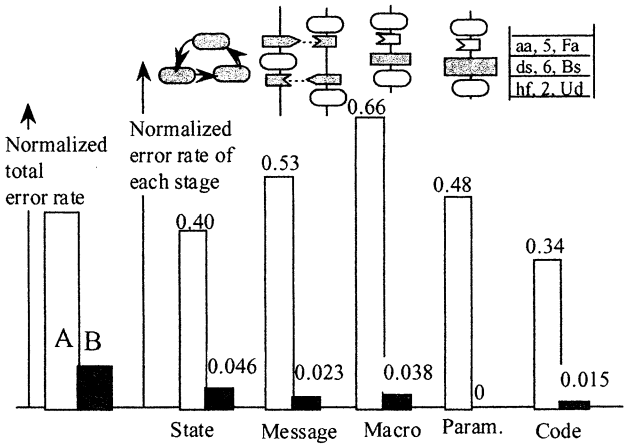


Figure 4. Test detected defects

After the programs were designed, machine tests began. In team A, many defects were found. However, in team B, a few defects were found. But, due to so large a difference, some people worried that the tests would not be done at all. More people were added to team A, and they worked a long time in order to keep to the delivery date. Fortunately, the system was completed, and delivered as scheduled. After all this, team B found a small number of defects.

The bar chart on the left most side of Figure 4[8] shows the overall removed defect intensity found during the test of each switching program. A is around 4 times larger than B. The study focused on the state transition route programs, which is the central core of the switching program. The design process was partitioned into the following five lower level sub-processes, and defects were analyzed from each defect report and they were classified according to each built-in sub- process, thus partitioned. They were in the following order:

1. Partitioning of FSM
2. Message between decomposed FSM's
3. Macro of state transition route programs
4. Parameters of macros and messages
5. Machine code

The right side bar charts show the thus analyzed defect intensity (normalized by defect/state). Very large differences between A and B are clear. During the study, it was revealed that team a desk-checked just a little. On the other hand, in team B as it was their first experience of designing using a state transition diagram, first the leader of B designed and checked by him in a step-by-step manner, and his subordinate checked again in a similar manner. Gradually the leader shifted the workload to the subordinates. If A had done such desk checks in the design, much overtime work and heavy budget overruns could have been avoided.

The next report is of an idealistic desk check performed during the small steps of a design. The program was for pasting a detailed data flow diagram next to the parental elementary data flow diagram for the automatic design system[5], and the size was 147 lines of C code. The design is repetitive hierarchical decompositions of elementary data flows, as shown by the design of the "clock" program in Figure 1 of Part 1.

IPO (Input Process Output) forms are arranged in hierarchical layers to form HIPO (Hierarchical IPO)[9]. The design is recorded on IPO form, and not the IBM way but using data flow diagrams in a sequence of data-function (processing)-data accurately. When the records increase to over one A4 sheet form, the continuing details are recorded in the next page to form HIPO.

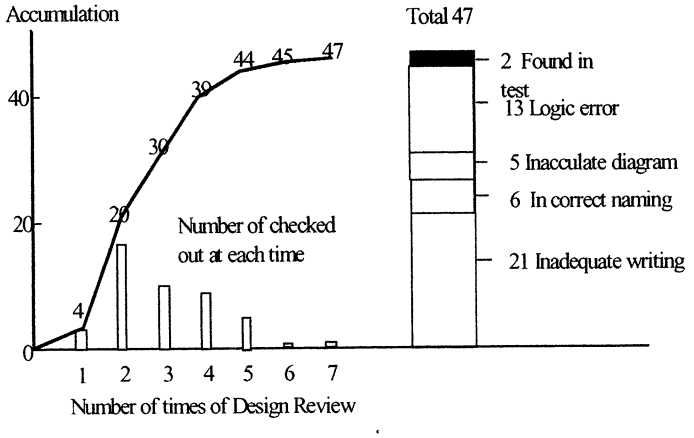


Figure 5. Check in small steps

As the design progresses, review was made at each step of design. A designer explained the design aloud reading data flows on the IPO, and his boss checked the documents while listening to what the designer explained. The left of Figure 5[10] shows the accumulation curve of the number of review times in the horizontal axis and number of check reports in the vertical axis. The right most side of Figure 5 shows a breakdown of the checks.

At first, most checks were mainly related to the drawings not in the design. Next, checks reached natural language expressions as in the diagram. After further progress, checks begin to include processing logics. At the 6th time, all the checks and the checks of amendments were finished, and the review ended. The designer then advanced to the coding, targeting 'no errors'. Actually, two coding errors were found in the test. Among the checked out items, which may relate to program errors, were 13. Adding 2 bugs, the built-in defect density was (13+2)/147 = 102 E/KLOC. (Beginners usually build-in around 100 E/KLOC.) The desk check rate remained 86.7%.

3. Quantitative process characteristics

In Part 1, the quantitative characteristics of human intentional activity in a development process were discussed. The external characteristics are obtained from the actual data. In this section, the "process" is explained using an excellent development taken as the example, with various diagrams and the work records. In an immature team, their

process characteristics show a large variation. In an excellent team, however, as the variation is very small, it is thus clear what they are doing.

This is a development of PBX (private branch exchange) software; the development was made in the end of 1980's in GTE (General Telephone and Electronics), which was the largest of the "independent" (non Bell, which had a market share of around 80% of the USA) US telephone companies at that time. Figure 6[3, original data in 11] is reproduced[4] under one of the authors, (Koono)'s responsibility.

These excellent leaders managed this development very well. People eagerly absorbed technical papers, discussed each element, performed trial experiments to gain the data they needed, and thus developed their method for the project. It was impressive to know that there were no disagreements among team members, and their answers were made quantitatively. The situation was like visiting an excellent hardware/components laboratory. During discussion we understood that quantitative measurements and discussion on the results made their opinions agree.

This project adopted various new technologies of that time. They took a compiler language and structured design. The hierarchical products layer was class, sub-program, module and segment. These layers were also used to name the design process. Hierarchical documents in structured design manner were used. They first made a pure design, and then desk checked at each process.

Figure 6 shows the accumulated defect intensity as the development progresses. The vertical axis shows the defect intensity, and the horizontal axis shows the layer name as design process. In "test", it shows the sequence of tests. Below layer names, each desk check method is shown. The final defect built-in intensity was 21.0 defect/kLOC. It was around 1/4 of the previous generation Assembler programs. Desk checks prior to tests removed 82.5% of built-in defects, and the defect intensity removed during tests was only 3.1 defect/kLOC. These are the top-level records at that time. Generally it is difficult for a team to remove more than 80% of defects by desk checks.

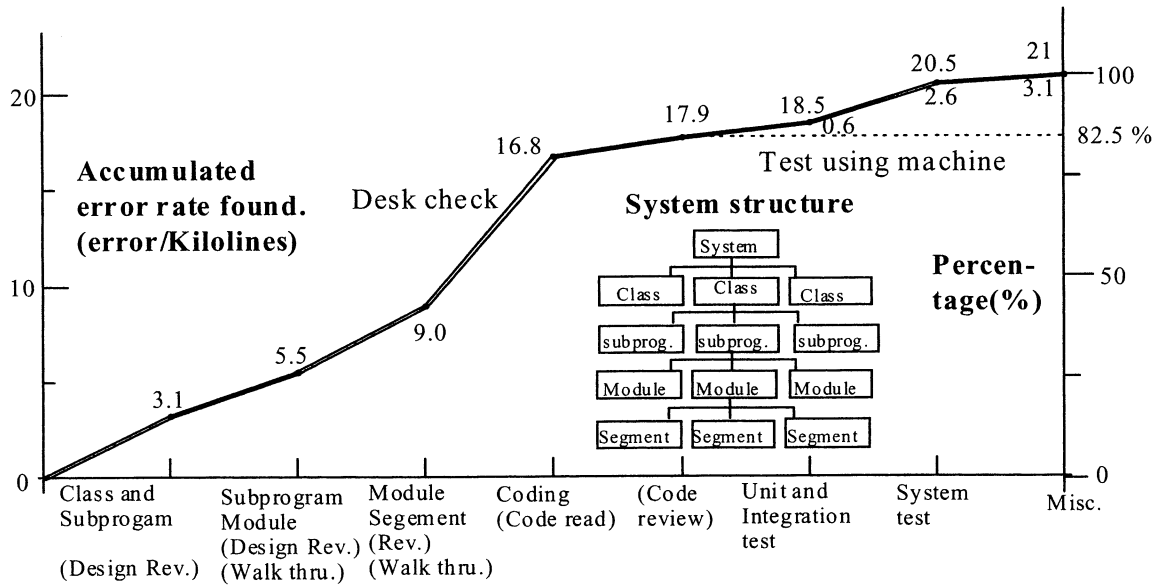


Figure 6. Accumulated defect intensity

[4] This excellent development team seemed to be born of GTE's special expectation. The top manager was not only MSEE but also MBA and a member of Tau Beta Pi and so on. The leader was an engineer with a scientific and rational way of thinking based on quantitative measurements, admired by members, and the team members were eager to absorb new techniques.

There were several keys to this result in that project.

(1) Abundant design documents.
(2) Emphasis on desk checks prior to tests, encouraged by the quantitative evaluations.
(3) Desk checks so as to check and correct defects by the preceding pure design.
(4) Rational man-hours allocated for desk checks.

In the early phase of the checks, Design Review (DR) was done in the originally proposed way[5]. The man-hours consumed by DR were the highest percentage (5.9% of total man-hours) among design checks. But the fact of their very small defect intensity found during tests proved the success of this method.

In the next phase, the weight of DR was decreased but a walkthrough for checking functionalities was added. As their design progressed, the weight of the walkthrough was increased. After coding, they checked the operation of each statement, thus simulating their minds and tracing the entire operation.

Figure 7.a[4] are evaluations after the project, and show the growth of defect intensity in designs. As shown by the upward-pointing arrow, defects are built in during pure design, and as the downward-pointing arrow shows, defects are removed in the following desk check, thus the residual (built-in minus removal) arises. (In this system, the exact built-in intensities of each design process were not known; thus all desk removal ratios are assumed to be equal to 82.5% of built-in defects.)

Figure 7.b[4] shows the decrease of the residual defect intensity as the test progresses. When shown on a logarithmic scale, the total decreasing rate is shown by adding each decreasing rate of a test on a logarithmic scale. (As the defect intensity found after delivery was not available, this is not shown in the figure.)

All defect reports should record each built-in process. (This procedure will be explained 4.2.6.) After a development terminated, and after the growth of defects in the field had been decreased sufficiently, all defect reports are summarized, and the accurate defect build-in intensities of each pure design process and defect removal rates of each desk check process are known.

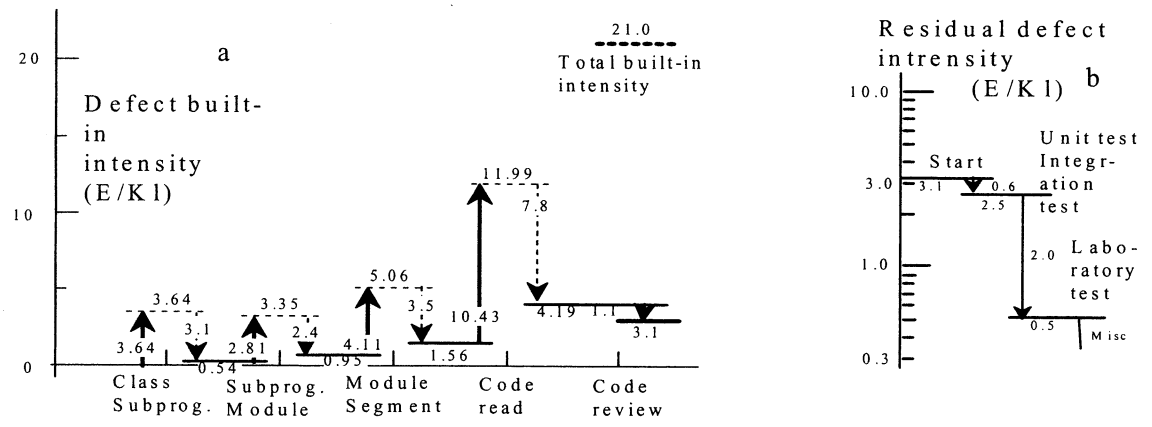


Figure 7. Defect intensity level diagrams

[5] Experts were chosen from the fields of planning, installation, operation, service and maintenance. They examined external specifications and performances, with which most designers were not familiar, through documents prepared by designers. Most designers do not touch such aspects, and defects are built-in. A designer knows, through the review item list, what are the problems, and learns a lot about the technology and the system performances, through preparing the description documents. Thus the designer's workload is very heavy. Nominated experts have to read and check thick volumes of documents, prepare each comment, and explain them at the DR meeting. In such cases, GTE telephone company's co-operation was obtained.

Figure 8 shows man-hour statistics. Figure 8.a shows the accumulated man-hours, where the horizontal axis shows the progress of development, using the name of the layer in the design, and the vertical axis shows the accumulated normalized man-hours. Figure 8.b shows the productivities of each process (normalized man-hours)/(source code lines), where source code lines are normalized to 1.

- The hatched parts in the bar chart show the defects attenuating or removing man-hours. The percentage ratio of (man-hours of de check)/(man-hours of pure design) shows the intensity of the desk checks. They were as follows: Class and sub prog. process: 59%, Module and segment process: 35.1%, Coding process: 33.1%

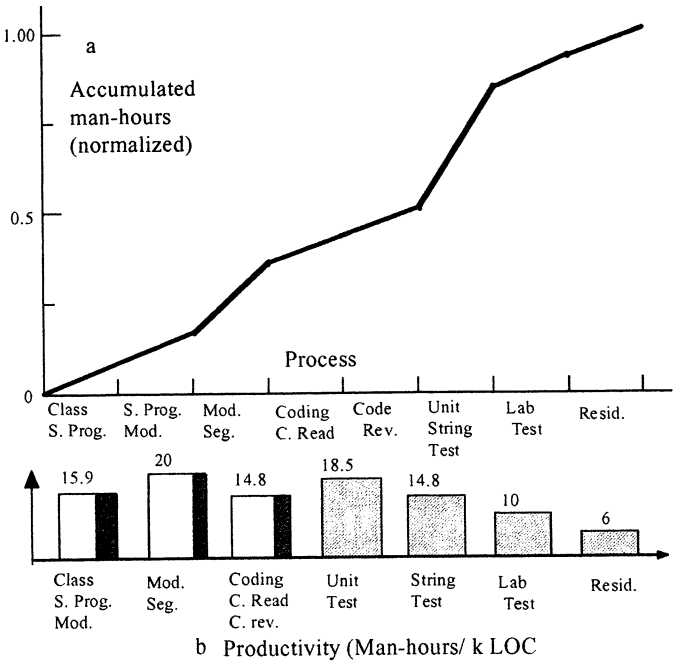


Figure 8. Man-hour data

As these show, in order to encourage people to do their best desk checks, the corresponding man-hours for them must be allowed in the budget. Based on past data, this estimation is possible as shown in model diagrams in Figure 9[12]. The upper one shows the time of the pure design and desk checks when the number of people is constant; it shows also man-hours. In the bottom figure, the horizontal axis shows time, normalized by the pure design. The vertical axis shows the residual defect intensity, which grows linearly during pure design (namely until time = 1), and decreases negative exponentially, after desk checks start.

The residual defect intensity is shown by e^{-aC}, where C is normalized time beginning at the start of desk checks and desk check removal rate $D = (1 - e^{-aC})$.

Let us denote the past data as normalized time C0 and desk removal rate D0, and the new targeting desk removal rate is D1 and the normalized time is C1. From two equations $(1 - e^{-aC0}) = D0$ and $(1 - e^{-aC1}) = D1$, C1 is given by $C1 = C0 \cdot Ln(1 - D1)/Ln(1 - D0)$.

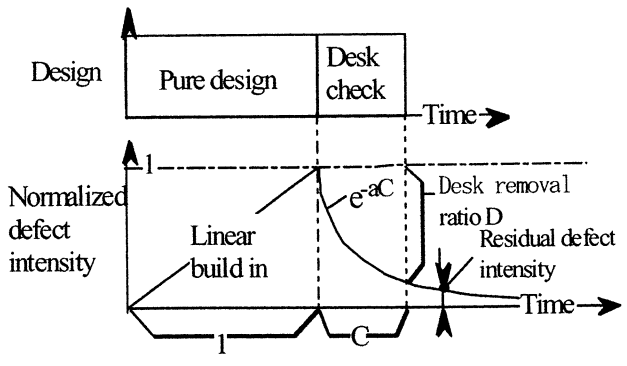


Figure 9. Desk check man-hours

In accordance with increasing allotted man-hours to C1, the method of the desk checks must be improved as to reach the targeted removal rate.

Another example is to allocate reasonable man-hours for tests. Man-hours of test process include works of:

1. test design,
2. pure test operation (with no defects),
3. defect fixing and amending (repair).

Pure test consumes (number of test) x (man-hours for a test), and fixing and amending consumes (number of defect) x (man-hours for a repair).

Each characteristic may be measured or estimated through calculations[5].

If quantitative measurements are made in various cases, the standard model becomes clear. If measurements of each component are made, and each contribution as well as various losses becomes apparent. By correcting the cause of each loss, the productivity is increased and the quality is also increased. Present state without monitoring quantitatively, the situation is like a blind person drives on a highway. When people reached this level, they are approaching hardware production processes.

4 Progresses

4.1 Learning Effects

When a person learns a new sport (physical work) or a new game (mental work), the person shows rapid progress at first. Then the growth rate decreases gradually, but it continues to grow. This is called a Learning Effect. As hardware production work is an N-time repetitive work, the Learning Effect has been known from early times, and since 1936 it (Logarithmic learning efect) has been included in industrial planning.

Figure 10.a[13] was published by one of Hitachi's software factories. The horizontal axis shows the number of designs, the vertical axis shows productivity (man-hours/kLOC) and the two curves are the data for each year. The curves show a rapid decrease at first and the improvement gradually decreases, as mentioned earlier. These are the Learning Effects.

Figure 10.b [3] is re-plotted in both-logarithmic scales from Figure 11.a. It is assumed on the author's responsibility that each software size is constant and their average value. Linear trend lines appear. This type of Learning is known as a Logarithmic Learning Effect[5]. In logarithmic Learning Effect, the work time for Xth is expressed by the following equation:

$Y = K \cdot X^{-A}$

Where X is the number of repetitions, K is the first work time, and A is a characteristic index of Learning Effects.

The gradient of the trend line is proportional to the improvement efforts, and it grows linearly as long as the same efforts are made.

The Learning Effect is supposed to be caused by the accumulation of the memory of the experience. The authors found that as experiences are repeated, the new

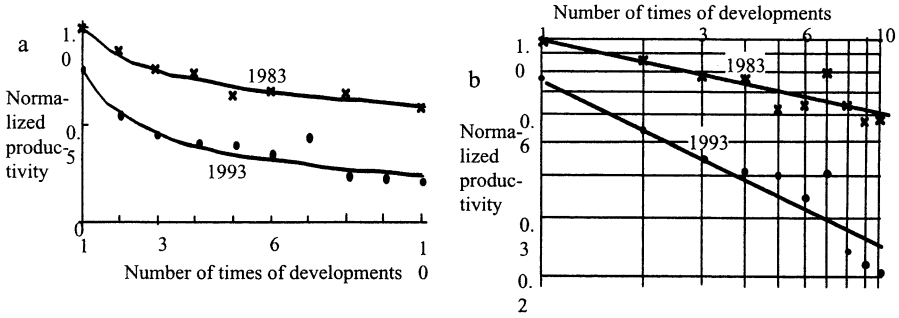


Figure 10. Logarithmic learning curve

knowledge (a pair of a parent and children) grows in logarithmic way[5]. The increase brings the increase/decrease of the characteristic figures. This means that *an improvement is a reflection of the knowledge accumulated.*

Most external characteristics of human related processes (as well as the product) show a Logarithmic Learning Effect. As the future value may be easily predicted, it is used for planning of the productivity of future projects, evaluation of the effectiveness of tools and future trend estimations such as the cost of new devises, etc.

Figure 11.a[14] shows the learning curves of productivity of several categories of software for pure new designs. The horizontal axis shows the accumulated normalized software size. Figure 11.b[15] shows the learning curves of defect build-in intensity for hardware production work and software design. These two graphs were the world's first reports of the Learning Effect of productivity and quality of software by Koono.

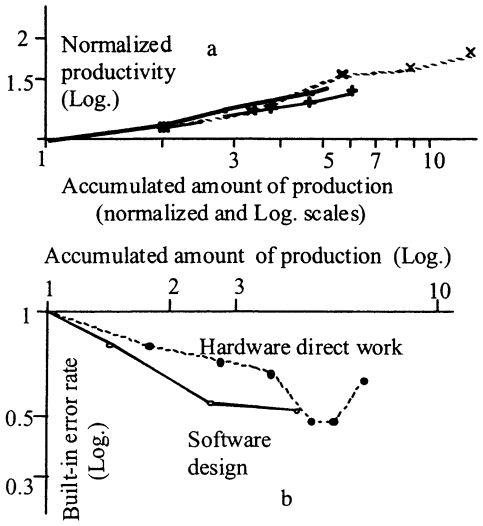


Figure 11. Learning curves

4.2. Business improvements

4.2.1 Independent quality organization

The top manager must be at the head of the improvement movement and has to lead by creating an improvement strategy that cannot be attained by other people. The following is a proposal to provide a quality organization responsible for all quality problems in a company, but independent of the line of design and production.

The Inspection or Quality Assurance department is under the top person in an organization and directly reports to that person and follows their instructions. They direct the whole organization on quality problems, educate people about quality, and, based on the tested results, have a right even to stop the delivery of products, and thus reflect top management's quality policy on all the products and services involved. Formerly, they applied a sampling test based on a statistical quality control, but as the overall quality of products has been improved, they changed to applying tests to all products. Besides such direct work, such quality controllers contact users as representatives of the company. Top management allocates excellent people to such work, and invests in quality related devices and systems. The number of these can reach from several to more then ten percent of the total number of people in a company.

In software, the Quality Assurance (QA) test is applied after the designers' test. After the designers' test has proved to show better results so then at the specified level, QA testing begins. But if the quality level is below the QA's specification, QA turns the system around to the designers. The improvements are shown in Figure 12. The top Figure 12.a[16] and 12.b[16], are from the first study. In Figure 12.a, the horizontal axis shows the number of defects found during the designers' tests and the vertical axis shows those found during QA test, both on a logarithmic scale. Figure 12.b shows similar data of QA in numbers vs. number of defects found after delivery for some

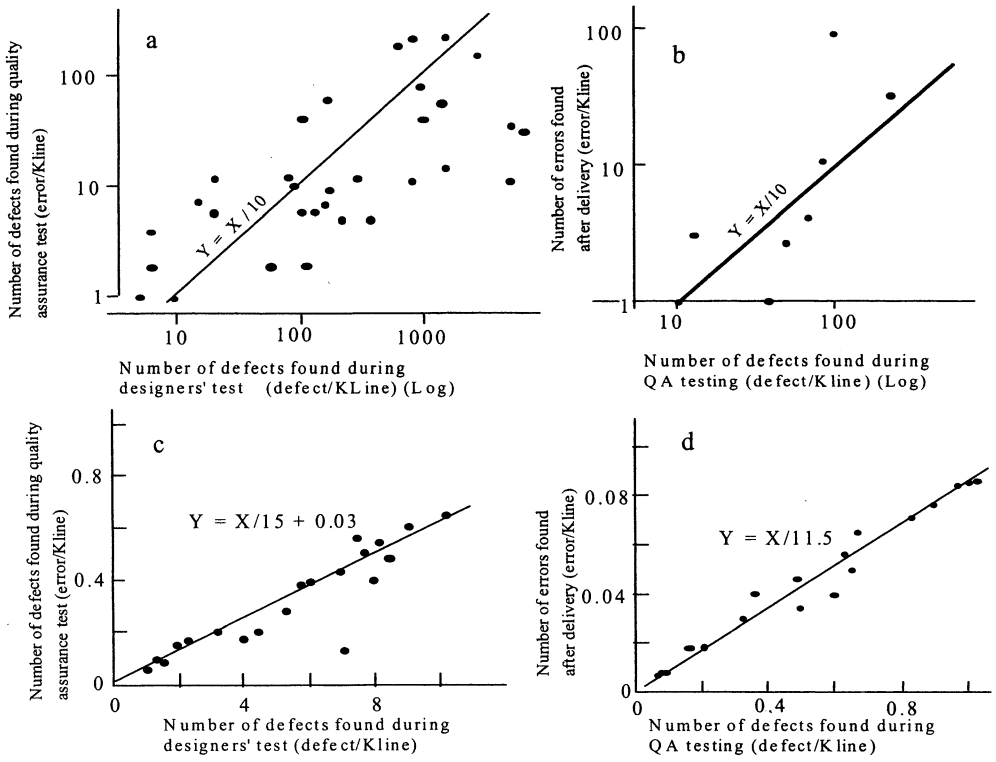


Figure 12. Correlations between the first test and the second test

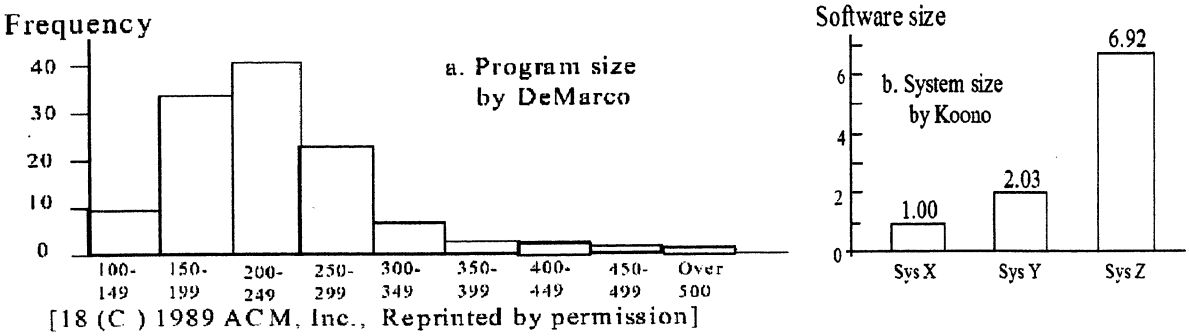
period until the appearance of the defects decreased sufficiently. The two graphs show the period until the appearance of the defects decreased sufficiently. The two graphs show a correlation of roughly 1:10, namely a series of test defects decreasing to 1/10.

Based on this result, an elaborate study was made. The results[17] are shown in Figure 12.c and 12.d (both shown in linear scales) on the lower level. The variation is much smaller and the correlation is very clear. The attenuation is around 1/10. Although the data used in Figure 12.a and 12.b were gathered from all the past data in a factory, Figure 12.c and 12.d were taken from a group of products. Constraining the process strongly decreases the variation.

The QA's test is black box test, while the designers' test is white box test, and numbers of both tests are equal. QA's tests are designed to check the weak points of the designers' group, the specialties of the market and customers as well as the respective weak points of inexperienced designers. In short, QA people are on the same level as designers and in some fields they are superior to designers. These are the secret keys for an organization that improves the quality level 10 times.

4.2.2 Common problems of both 'product' and 'process'

In software, quantitative measurements and their use reveal many problems. The first theme is the development cost. Figure 13 shows examples of software size. Figure 13.a is from DeMarco's paper[18, © 1989 ACM, Inc. Reprinted by permission]. Programmers wrote the same program to the same specification, and programs were tested. He was impressed by the fact that the ratio of the largest/smallest is 8. As the distribution has a fast rise and slow decay, it may be regarded as a lognormal distribution, where the ratio of the upper limit and the lower limit is equal to $3^2 = 9$.



[18 (C) 1989 ACM, Inc., Reprinted by permission]

Figure 13 Software size

Figure 13.b[10] shows the size of switching system software of roughly the same specification. If the central system Y is taken with a normalized size of 1, the size range is from {1/2 - 1 - 3.4}. It is near to the {1/3 - 1 - 3} range and of lognormal distribution, which is supposed to be in accord with the statistics. These two examples mean that the software size ranges from the least, 1/3 times to the largest, 3 times of the average, and the ratio of largest to smallest reaches 32 = 9. As these two samples show, not only the characteristic of a human related "process", but that of a human created "product" also shows lognormal distribution.

The development cost of a system is expressed by the software size multiplied by the productivity. The productivity shows lognormal distribution as shown in Figure 5 of Part 1. Using the power sum law ($3^2+3^2 = 4.26^2$), the largest /the smallest ratio for development cost (measured by the software size) is 18. If the development cost varies to such an extent, is the management of a software development project possible?

The software size problem affects other aspects. Most software, after its initial development, increases the total size approx. 10%/year[6]. The software size relates not only to the initial development cost but also to annual upgrading (so-called maintenance) cost. In addition to these, the software size relates to the real-time characteristics (i.e. processing power and response time) and the reliability.

In any other engineering, the product's cost is regarded to be shown by (number of items) multiplied by (cost/item). In software, the former is software size and the latter is the productivity. In these 10 years, process improvement has been discussed. But, not only the "process" but also the "product" should be managed. Therefore, it is important to measure both "process" and "product".

4.2.3 PDCA management cycle

PDCA are abbreviations of the Plan, Do, Check and Action (cycle) of management. This system was developed originally by Shewhart, W. A., and further developed later by Deming, W. E, and both pioneers in the quality field. The cycles are as follows:

Plan: establish a work plan.

Do: execute the plan.

Check: monitor to ensure all is going as planned.

---

[6] The trend was found in a telephone switching software in the 1970's[~~20~~], and it had been confirmed by various embedded systems at that time. The telephone switching software continued to grow at this trend; most systems' size had reached mega line in the1990's.

Action: take corrective actions to correct the discrepancies from the initial plan.

When one cycle ends, the people involved must have advanced through PDCA. *The next plan must aim at a better target than before*, as people will have advanced. Through thus repeating these cycles the people involved continue to advance. It is like a spiral progress. In other words, the progress shows a logarithmic Learning effect.

This management system is used in the following manner.

PLAN: Before planning, the manager, who is responsible, gives the strategy to the head of the team responsible for it, and shows how the work should be done. This person decomposes the given strategy to several means to attain it. It is important to establish a tactic for the strategy. Then the person discusses the way to attain it with his/her subordinates, and by leading and encouraging the subordinates details it to the plan for implementation. There will be various difficulties in the implementation of the plan. The countermeasures for these are listed, and some margins to allow individual initiatives are included in the plan. (IE inhibits plans without reasonable margins.) Thus all the members commit it themselves to it.

DO: The weekly performance is gathered and displayed on a screen by two curves of the target (budget) and the reality. The leader monitors the curves.

CHECK: When a discrepancy is detected, the leader calls the members to help them understand and clarify what is the problem.

ACTION: Then they discuss how to resolve the discrepancy, and the countermeasures are determined. Hereafter, people must work coordinating the original plan with the countermeasures. If the recovery is not made or might not be made, the leader must immediately report to the manager. The manager gives advice to the people involved. If this is still ineffective, the manager has to report to the director of the company. The director does the same.

A human person has two sides, one is lazy and another is industrious. In order not to be lazy, it is forbidden not to aim at the target, and this enables people to see the reality quantitatively in contrast to the work target. In order to improve themselves of their own accord, let people join in the meeting, encourage them to speak, propose, and challenge them to be successful. Thus feeling the joy of achievement, people improve.

By acting thus, most troubles are solved in the stipulated period, as people have improved their work during that time. In the next planning session, the manager orders leaders under him/her to analyze the major causes of the discrepancies, establish the countermeasure not to repeat them and encourages leaders and others to establish a plan aiming at a higher target. By thus *repeating quantitatively, people advanced, and the achievements in business are improved. Due to the accumulation of knowledge, the learning curve also grows linearly.*

4.2.4 Approach for Improvements

The following is some advice for applying these improvements efficiently.

1. Eliminate "waste", "strain" and "imbalances" first, in order not to undermine the improvement efforts.
   - Top management and managers are expected to eliminate "waste", "strain" and "imbalances" prior to people's efforts at improvements. Their causes may include some beyond the human level, where the forefront people have rights and responsibilities. If top-level people challenge others to improve on these and they succeed in this, that fact will not only encourage people but also stimulate them.

2. Variations on various phases of a development must be decreased enough prior to other improvements. Tightening up by dividing the process improves the variations caused by internal reasons, and these contribute to decreasing other problems.
3. Concentrate improvement efforts on a few targets at a time and overcome them early. Pareto analysis is a good tool for the selection of these.
4. An improvement is to correct some part of a person. As every person makes his/her best effort already, the person may not be changed easily. It is important to help people to identify the seeds of growth so they can improve by themselves.
5. At the beginning of the improvement efforts, do not require too precise/accurate data. But as improvements go on, measurements may be made for precision and accuracy, to learn what is more effective and what is not.
6. Provide an opportunity for people to think by themselves, be creative in using their own knowledge, implement things in front of others, make presentations before people, etc. so that they may thus make others know about their work and enjoy a sense of satisfaction. It is important to stimulate people's minds also.

4.2.5. Quality control

The fundamental cause of defects is human error, which is common also to production works. Quality control has been developed for hardware production, and the techniques can be applied also to software works. The Pareto[7] analysis, for screening problems, is the most popular technique in software. Figure 14[3] is a model Pareto chart. The horizontal axis shows the items concerned sorted in descending order, and the vertical axis shows the amount of loss. The bar charts show the amount of loss for each item, and the Pareto curve is an accumulated graph of loss.

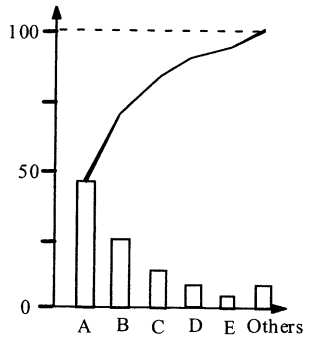


Figure 14. Pareto chart

Juran, J. M. (one of the pioneers in the quality control field) pointed out that *a few vital* elements constitute the majority of the losses. (The rule is also known as the 20%/80% law.) It means that improvements of a few (20%) parts decrease the majority of losses (80%). Therefore, this is used in selecting improvement themes for solving quality problems. If the vital 20% constituting loss of 80% are not found, repeat to change the viewpoint and re-calculate the curve until the a *few vital* is found.

4.2.6 How not to repeat the same errors

A defect arises as a result of a human error. In order to decrease defects, there are two ways, namely to attenuate defects and not to build-in defects. Here, the latter way is discussed. It is impossible not to err in every respect. The only thing possible is to provide some countermeasures ***through feedback*** to ***prevent people repeating errors***. Rather than read the 100 best practices, it is better not to repeat the same error/mistake again. This was found in Total Quality Control/Management (TQC or TQM), and has

[7] This law was originally found by Pareto, V., (an Italian economist), where the reality was that a few rich people possessed the majority of the wealth.

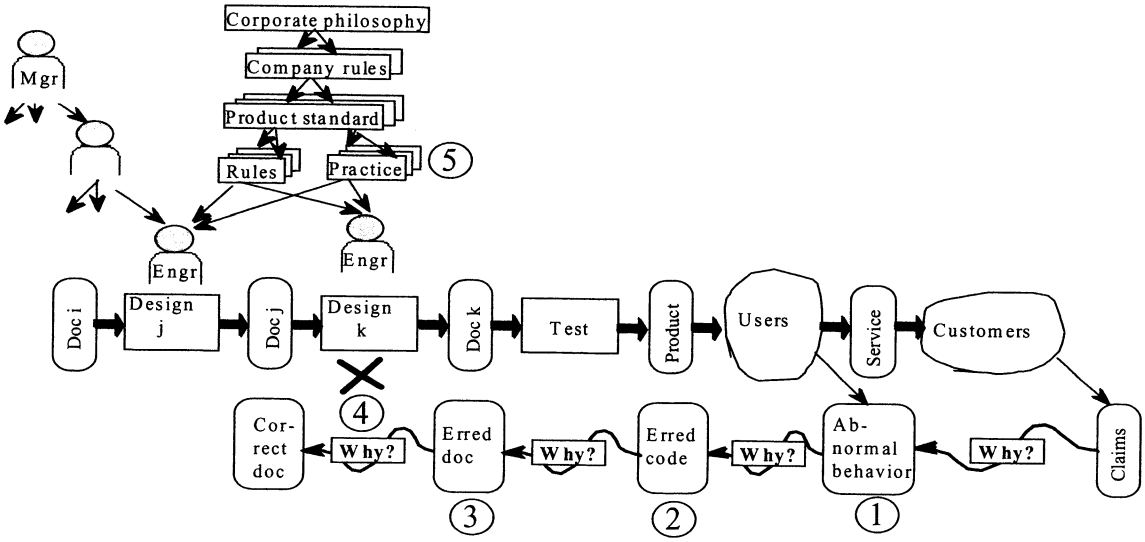


Figure 15. Cause and effect network

resulted in substantial improvements.

Figure 15[3,20] shows a method from the viewpoint of "process." (Originally, it was said to repeat "why and why". But, outside people cannot understand this.) The middle level shows the process flow of a development. After finishing design then tests, a product is delivered to a user and then served to a customer on the right. The customer had trouble and made a claim at the maintenance center.

(1) Responding to a claim on the right side, a service person asks the customer "what happened, when, how, where and what were you doing?" For the records, the problematic behavior is noted.

(2) Based on the record, the faulty mechanism is analyzed, and the mistaken code is identified. For repairs, the correct code is designed, tested and after the QA's approval, the product is repaired. But, the search continues.

(3) Trace upward from the erred code to where the error is built-in. Trace backward from the result to the cause: code -> flowchart -> data flow and so on.

(4) The erred process is where the output documents include error related information but the input document does not include any error related information. In the figure, it is "design k process".

If a person understands the flow of the process, that person can identify the build-in process easily. (It should be noted that the process is divided into many finer processes and enables the identification of the detail of the erred process.)

Each hierarchical decomposing has another side. Any cross-section of a design consists of the following processes:

"Cognition" – "decision" – "conversion".

"Decision" is the selection of the manner of the conversion (algorithm).

"Conversion" is to convert the input information to the output information. Prior to these, there is 'cognition', such as of procedural errors like misreading or mistyping, misunderstanding of the problem, etc. Referring to the input document and the erred document, estimate the erred process from the above three.

An error is the result of a cause at an earlier phase. Trace-back upward to find the cause. During the process, a vital cause X is found that invited the error. In continuing the tracing further, there is a process Y, where if there were no Y there would be no X. Thus the countermeasure to avoid repeating the same error again is to change Y to Y'.

Let us assume that the problem belongs to "cognition" and that a designer in design process k misread the input document. Further upward cause analysis revealed that this is a special case where more additional detailed requirements must be written prior to the design. Then in the earlier phase of standard design practice, an addition is made "in case ····, use design rule ···," and in that part of the work "yyyyyy" is added.

Based on these updates of the standard design practice, special notice is sent to all design teams and people read about the updates as well as about the cause of the trouble and the problem fact sheet. After all the people involved changed their work methods and use this new way, the countermeasure becomes effective in preventing the same error again.

This prevention method is rational and scientific. Moreover, as it is an improvement method for the process, it may be used not only in hardware production work but also in any human intentional activities where design is involved as well as management. TQC/M was developed originally in hardware production. But as it penetrated into industry it attained various improvements. "Improvement by feedback" is one of them. As it may be applicable to a "process" irrespective of component, hardware or software, the application achieved remarkable progress from the 1970's to 1990's.

5. Discussion

Figure 16[~~21~~] shows a very long-term learning curve (in bold line) with the distribution graph of a characteristic. The horizontal axis shows years, and the vertical axis shows the characteristic value. Each small distribution curve of the characteristic (rotated 90 degrees in counter clockwise direction) is pasted on several points of the horizontal axis. The left most side edge of the learning curve shows an initial phase large average value with large variations. After the end of the 19th Century, hardware people had accumulated much experience, acquired various empirical rules, and new engineering bloomed. Based on such efforts up to the present, they are now near the right most ends, where Y is near 0. The enormous accumulation of the knowledge achieves these.

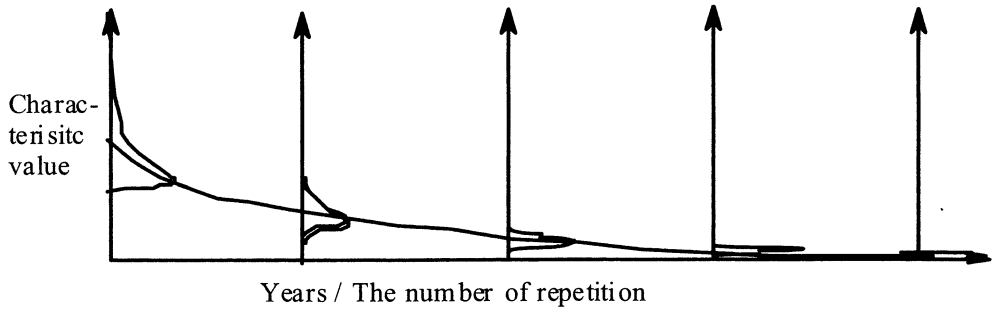


Figure 16. Very long-term learning curve

Where is the software industry located now? In a software development project, after the initial planning and the fundamental design, the number of software people increases rapidly. After the peak has passed, the number decreases, and finally just a few people are left. In some cases, the people come from the outside. As soon as that part of the person's work ends, the person leaves the project and goes away. In the worst case, the person's "product" contribution stays in the software, but what the person achieved in the project (process aspects) is not passed on, or the knowledge is

not accumulated.

In such cases, every project starts from the left most side edge of the learning curve and the project ends at the second line with still large characteristic value with still large variations, and thus the characteristics of the project show a large variation.

- Due to the too large variations of process characteristics, the forefront people miss the information to measure and collect the projects' data.
- Likewise, researchers also might lose confidence that the "scientific approach reveals the secret".

Thus, all quantitative, rational and scientific approaches are discarded. That is the worst-case scenario and the best method is to concentrate on a particular field and to accumulate all the knowledge involved.

In hardware, most companies specialize in a narrow area, and put emphasis on accumulating knowledge. In the early age of PC, it was difficult to co-exist with the "Gulliver". The situation has been much improved and the market of software has widened much. It is important for most software venders to specialize in some area.

Ford's conveyor system is the "key of a factory;" it enables various simple level workers to participate in production. In this case, "the conveyor" is the main body of knowledge. Japan's "software factories[19]" have not disclosed their secrets, but education, work practices and production systems together with reusable components would constitute the "conveyor".

Anyway, it is important for a software company to consider "what is your business model"; in reference to the knowledge they have accumulated.

6. Conclusion

Part 1 discussed the "product". It explained the basic structures and the fundamental characteristics of human intentional activity putting emphasis on software. What was discussed in this may be applied to any human intentional activity, from management to design and physical work. Part 2 discussed the "process". The "process" is for the control of the various "products", thus it is orthogonal to the "product". The basic structures and the fundamental characteristics, including the Learning Effect have also been discussed. These explain the theoretical basis of empirically evolved Industrial Engineering practices and their use in various fields, including software.

As "process" is for control on the aforementioned basis, there are huge quantitative measurements, planning and control techniques involved. Thus, major techniques for management are discussed in the last half of Part 2. Most of them are similar to what has already been done in industry.

Both the first part and the last part show the principles involved. As has been reported in Section 2 (referring to Figure 3), it requires additional efforts to make a "best practice" work as intended. Through the process, one acquires one's own knowledge. Through repeating thus, the amount of knowledge accumulated increases.

Various relationships, methods, rules and principles reported here become valuable when people use them, make efforts to master them and finally achieve success, based upon the structures described earlier. The key that enabled the Japanese hardware industry to grow was the fact that all the people involved, including forefront people, learned introductory IE's and all the people worked on the same basis targeting the same objectives.

What the authors hope is that what they have studied will be used and found effective by many others. Various discussed quantitative, rational and scientific engineering undoubtedly increase peoples' productivity and quality.

Acknowledgements

The author wishes to express his sincere thanks to his superiors, colleagues and forefront software people in the Hitachi, Ltd. Totsuka Works, This study is the result of their kind cooperation. The authors also wish to express their gratitude to those who joined the Software Creation Project in Saitama University. They are also very thankful to Mr. Daniel Horgan for his corrections and valuable advice on their English.

References

[1] Z. Koono, H. Chen and H. Abolhassani, An Introduction to Quantitative, Rational and Scientific Process in Software Development (Part 1), companions paper this conference, 2007.
[2] Z. Koono, H. Chen and B.H. Far: Expert's Knowledge Structure Explains Software Engineering, Joint Conference on Knowledge-Based Software Engineering (1996), 193-197.
[3] Z. Koono and H. Chen: Toward Quantitative, Rational and Scientific Software Process, Proc of Software Process Workshop 2005 (2005), 454-458.
[4] Editor G. Salvendi: Handbook of Industrial Engineering, John Wiley & Sons, Hoboken, 1982.
[5] Z. Koono, H. Abolhassani and H. Chen, A new way of Automatic design of software (Simulating human intentional activity), SOMET 06, pp. 407-420, 2006.
[6] Carl von Clausewitz: Vom Kriege, Dummlers Verlag, Bonn, 1832.
[7] Z. Koono, K. Ashihara and M. Soga: Structural Way of Thinking as Applied to Development, IEEE/IEICE Global Telecommunications Conf. (1987), 26. 6. 1-6.
[8] IBM, HIPO-Design aids and documentation technique, IBM publication GC20-1851-1, 1975.
[9] E. B. Daly and D.A. Mnichowicz, The management of large software development for stored program controlled switching systems, International Switching Symposium 979, pp. 1287-1291, 1979.
[10] Z. Koono and H. Chen, Empirical quantitative model enables quantitative planning of a development, International Symposium on Empirical Software Engineering 2003, Vol. 2, pp13-14, 2003.
[11] Y. Morioka, F. Nagano, O. Ohno, On learning effect of program development works in Eagle/P (CANDO) environment, National Conference of IPSJ 1991 later half, 5-385, 1991.
[12] Z. Koono, H. Tsuji and M. Soga: Structural Way of Thinking as Applied to Productivity, IEEE International Conf. on Communications (1990), 204. 2. 1-7.
[13] Z. Koono, K. Igawa and M. Soga: Structural Way of Thinking as Applied to Improvement Process, IEEE Global Telecommunications Conference, (1988), 40. 1. 1-6.
[14] Z. Koono and T. Ohotsubo, Evaluation of build-in and check-out of software errors, Research report of IPSJ, Software Engineering 95-5, pp. 31-38, 1993.
[15] J. Watanabe, H. Ogata and E. Kobayashi, Prediction methods of software quality and software productivity, 2nd Symposium of Software Production Control, A-2, pp.7-14, 1982.
[16] T. DeMarco and T. Leister, Software development; State of the art vs. state of the practice, 11th International Conference on Software Engineering, pp. 271-275, 1989.
[17] Z, Koono, Kondo, T., Igari, M. and Soga, M., Structural way of thinking as applied to good design (Part 1. Software size), IEEE COMSOC Global Telecommunications Conference 1991, pp.24.3.1-8, 1991.
[18] Z. Koono, M. Toiya, T. Matsuida and M. Soga, In-Service quality improvement activities, IEEE Journal on Selected Areas in Communications, Vol. 6, No. 8, pp. 1299-1304, 1988.
[19] M. A. Cusumano: Japan's Software Factories; A challenge to U.S. Management, Oxford press, 1991.
[20] Koono, Z., Processor systems in High integration age, Joint Conference of Four Electrical institutes 1979, No. 27-3, 1979.