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 1)

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. Repetitive decomposing of the objective concept hierarchically, developed for design involves a human intentional activity, ranging from management to physical works. This model proved the empirical rules in Industrial Engineering. In addition to these "design" process characteristics, the "test" process is found to be a defect attenuating process with the attenuation rate of its error rate of the second kind. As these are applicable to any software process, they are very useful.

Keywords. Design, Test, Process, Productivity, Man-hours, Defect, Defect intensity, Industrial Engineering

Introduction

After the end of the 19th Century the industrial production of hardware began, and Industrial Engineering (IE)[1] was also born, and overcome many problems and built up the Production Control of hardware. Later at the beginning of the 20th Century, Quality Control made rapid progress. IE achieved the industrialized production.

Japan introduced IE almost 50 years later than advanced countries. Due to this delay, the rapid deployment of IE began. In the middle of the 1960's, the IE's standard time system was introduced in hardware factories and penetrated many industries. In the 1970's to 1980's, Japanese Total Quality Control/Management (TQC/M) grew and it enabled economical production and quality products from Japan poured into the world.

IE's quantitative, rational and scientific technologies achieved these. The purpose of Part 1 is to introduce them to software development. This is the "product" aspect, and Part 2[2] discusses the "process" aspect[3]. Section 1 introduces design by repeated decomposing of the objective concept hierarchically to mean objects; it is common to mental work and to physical work. Section 2 proves the empirical rules of IE. These mean that the basics of IE may be used also in software. Section 3 explains that a test is a defect attenuator, and the decreasing rate is equal to the second kind of error rate of inspection. Various field data are explained to enable a quantitative evaluation of these.

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

1. Human intentional activity

"Stepwise detailing" by Wilth, N. in 1971 is regarded as the first proposal for a design method in software. Later, it was developed into various structured methods from the 1970's to 1980's. They claim that functions have a hierarchical structure. The hierarchical decomposition is a common structure in various design methodologies. Figure 1[4, 5] shows the authors' design that repeats the hierarchical decomposition of an elementary data flow.

On the left side of Figure 1, data flows show the design record of a Clock program. The specification "clock" is defined by input and output data to form an elementary data flow. It is regarded as a parent and is decomposed to the detailed data flow of the children in the next level. This consists of three *serial* elementary data flows "Obtain time," "Obtain hands" and "Display", following Myers' STS division[6].

As it is a program, an execution sequence of each function is needed. A flowchart starts from a compressed barrel symbol above the first function and goes along the bold line to reach the end mark of a compressed barrel on the last function.

Next, three elementary data flows are decomposed hierarchically. The figure shows that of "Obtain hands." The input and output data of the elementary data flow are hierarchically decomposed, and using the pattern of Jackson's program design[7], the elementary data flow "Obtain hands" is hierarchically decomposed to three *parallel* elementary data flows.

On a lower level a hierarchical decomposition of an elementary data flow "Obtain minute hand" is shown. The detailed result shows that the degree is 6 times sixty minutes, and the next hierarchical decompositions are to convert this from natural language expressions to programming language expressions as the implementation means.

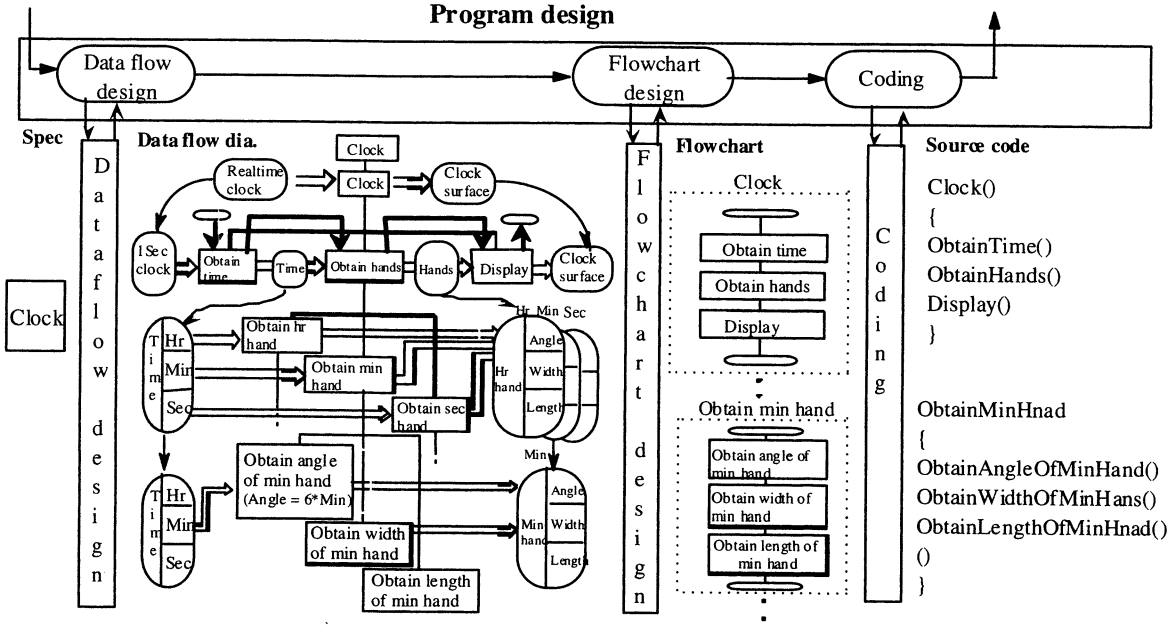

Figure 1. Design records of "Clock Program"

A design is repetitive hierarchical decompositions. As it is repeated, it becomes clearer, more detailed and minute[8]. At the final stage, they are converted to the implementation means (c. f. source code).

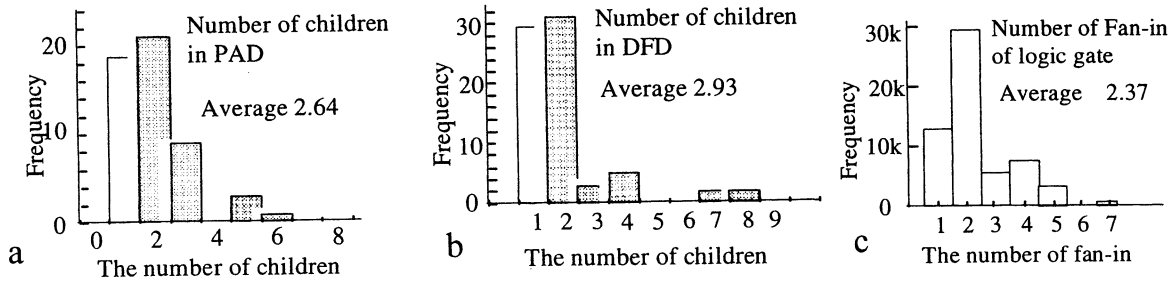

Figure 2. Various expansion rates

As the authors were studying the automatic design learning from human design[20], the aforementioned hierarchical decomposition was standardized. As the experience is accumulated, it was found that it brings about a good design (ease of understanding) and constancy of the expansion rate. Figure 2.a[9] and 2.b[9] are taken from actual designs. Figure 2.c[10] is from a logic circuit design. In logic circuit design, a designer synthesizes a logic circuit using lower level logic signals, which is the same as the hierarchical decomposition. These three show a similar expansion distribution.

The average is a little smaller than 3. It is lower than the usual number of chunks. Surveys in cognitive science found two experiments. The first one is that the expansion rate in a short-term memory is around 2. Another is that larger chunks appear in cases where a long time is allowed for remembering. From these, the average rate seems to appear as a result of a speed neck. A theoretical study shows that e = 2.71828.. is the optimum, but not yet perfect.

Let us examine other cases. Figure 3.a [11] shows the human intention for a physical action, "take a picture." It is hierarchically decomposed to the three implementation means as shown in the figure. Each of these may be further hierarchically decomposed repetitively until they are reduced to nerve signals to drive a muscle as intended.

In military science, "Hierarchy of Object[12]" is an important empirical law in planning a war. It is a repetitive hierarchical decomposition to the details of "occupy island X" as shown in Figure 3.b[11]. This law is explained as follows:

> The highest executive of a nation assigns the final objective of the war to the supreme commander. This person decomposes it to several means to attain it, and assigns each to the respective subordinates. Then these people do similar actions. Through repeating thus the plan of the war may be detailed.

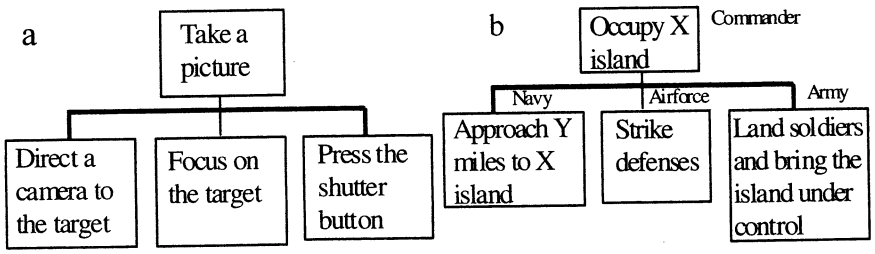

Figure 3. Human intentional activities

Thus continuing, the final objective is broken down to each move of the soldiers involved.

As shown by these three examples of human intentional activity, this repetitive hierarchical decomposition may involve a management objective (at management level), a function (at design level) and then on down to human physical action. All kinds of designs are included in these. The central core of software design is to perform the hierarchical decompositions of concepts in the problem area. The program technique is the implementation means used at the final stage.

2. Constancies of productivity and defect intensity

The axioms in hardware production control and quality control in IE are following simple *empirical rules*. If they are applicable also in software, almost all of the techniques in the IE may be used also in software.

- Linear nature
  Hierarchical decomposition and integration
  (Simple arithmetic operations are possible)
- Productivity
  = (Man-hours)/(number of items)
  = constant
- Defect (build-in) intensity
  = (Number of defects)/(number of items)
  = constant

Although more than 100 years had passed since they began to be used, these empirical rules have not yet been theoretically proven. The object of this section is to prove these apply to human intentional activities in general and also in software.

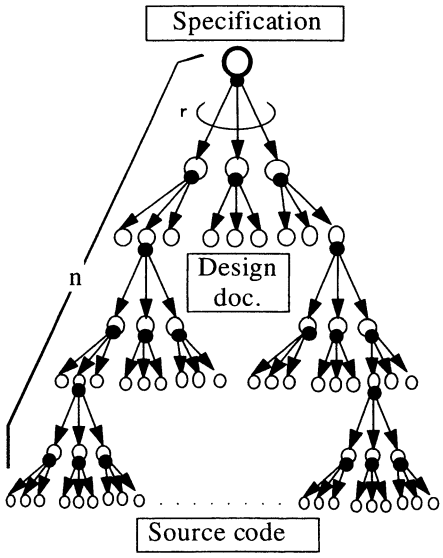

Figure 4. Hierarchically expanding network model

As has been mentioned before, "producing" and "designing" belong to human intentional activity. The central core of the activity is repetitive hierarchical decomposition. Figure 4[13, 5] shows a model of this activity. A white circle is information; a black dot is an elementary processing node. r is the expansion rate, and that is a constant.

In a design, an elementary decomposition node at the top decomposes hierarchically a specification at the top to r intermediate outputs. Next r elementary decomposition nodes operate sequentially. All nodes operate sequentially like the single stroke of a brush, and the final outputs are generated.

Let us assume that a small time t elapses while an elementary decomposition node operates. The man-hours consumed in this network may be evaluated by multiplying them by t·(the total number of elementary decomposition nodes). Let us assume that the elementary decomposition nodes err at a small error rate of e, and thus the erred output propagates through to the final output. By e·(the total number of elementary decomposition nodes), the total number of errors, namely defects, in the final output may be calculated. Thus through these the total number of outputs, both productivity and defect intensity, may be obtained. If these results proved both constancies, the

empirical rules are proved by the aforementioned mechanism, 100 years after they came into use.

When r is constant, the characteristics of the hierarchically expanding network model shown in Figure 4 may be calculated using the formula of a constant-rate increasing series. Let us assume that the network consists of n levels from the top to the bottom. Let N_p be the total number of nodes in the network, and N_0 the total number of the output information. These are expressed as follows:

$$N_p = 1 + r^1 + r^2 + r^3 + + r^{n-1} = (r^n-1)/(r-1) \quad \text{Eq. (1)}$$

$$N_O = r^{n-1} \quad \text{Eq. (2)}$$

$N_p/N_O = 1/r$ = constant, provided that n>1. Eq. (3)

Let assume that man-hours for the processing of the network are the sum of small time *t* consumed at each node for decomposition. The productivity P may be evaluated by (the total number of the output, divided by the man-hours).

$$P = N_O/(t \cdot N_P) = r/t = \text{constant} \quad \text{Eq. (4)}$$

The following characteristics are obtained:

- The productivity is constant
- The man-hours are proportional to $(\text{size})^1$
- The system is a linear system

These are applicable to both mental work such as design and also physical work in hardware production.

Let assume that each node errs at the small rate of *e* during the processing of the network and the error propagates to the output. The defect (build-in) intensity E may be evaluated by the total number of the nodes multiplied by *e* and divided by the total number of the output.

$$E = e\,(N_P/N_O) = e/r = \text{constant} \quad \text{Eq. (5)}$$

The following characteristics are obtained:

- The defect build-in intensity is constant
- The total number of defects is proportional to $(\text{size})^1$
- The system is a linear system

Empirically used constancies are theoretically proved, and the inner mechanisms are also clarified. The linearity guarantees hierarchical decomposing and integration as well as simple (arithmetic) operations for their characteristics. They are useful for practical use. Also, it is noteworthy that errors during intentional activity arise from human mistakes. The actual field data of software developments verify these relationships.

It is desirable that they are many, not biased, from various applications, ranging from small size to large size and implemented by various languages. The materials chosen are from the 1970's and the 1980's, when the initial difficulties of large software developments were overcome but most developments were new and almost without reuse. Figure 5 is man-hour data and Figure 6 is defect number data, both plotted on both-logarithmic scales, where the horizontal axis is the software size and the vertical axis is man-hours or the number of defects (errors) and the plots show a belt-like distribution.

Figure 5.a[14] is re-plotted from Nelson's RADC (Rome Air Development Center, USA) data, adding two sub-trend lines, being equal distance from the center trend line, one located at 1/3 times (standard deviation) of the center (mean) line and one located at 3 times (standard deviation) of the center (mean) line. Nelson reported that the center

trend line equation, gained statistically, is $X^{0.975873}$. (The difference of the exponent of 0.024127/1.0 is not so significant as to deny the linear nature.)

Figure 5.b[15] is re-plotted from Boehm's COCOMO data, and Figure 5.c[16] is re-plotted from Yoshida's Fujitsu data in linear scales. Both plots, as a whole, show a trend of $Y \propto X^1$. They are processed graphically in following way:

- A center trend line of $Y \propto X^1$ is drawn to pass through the center of the plot group. The position of the interceptor of the Y-axis is adjusted as follows:
- Two sub-trend lines are drawn up and down along and equal distance from the center trend line. The distance and the interceptor are adjusted so that a few plots are outside of the belt-like zone. The distance is represented as N, which is the normalized number of the distance by the center (mean) value[2].

These three figures show that, plots in the belt-like zone show a trend of $Y \propto X^1$ or the productivity is constant, or man-hours are productivity multiplied by the software size. Also from these three data the linear nature of the man-hours is verified.

Defect data, Figure 6[17], is re-plotted from Thayers' third project data in linear scales, and the same graphical treatment as above is made. The main trend line shows that $Y \propto X^1$ or defect (built-in) intensity is constant, and the linear nature of the defect intensity is verified.

Thus both constancies of productivity and defect intensity are verified and the linear nature is also verified. As both constancies of productivity and defect intensity share the same body, the node, these are due to the

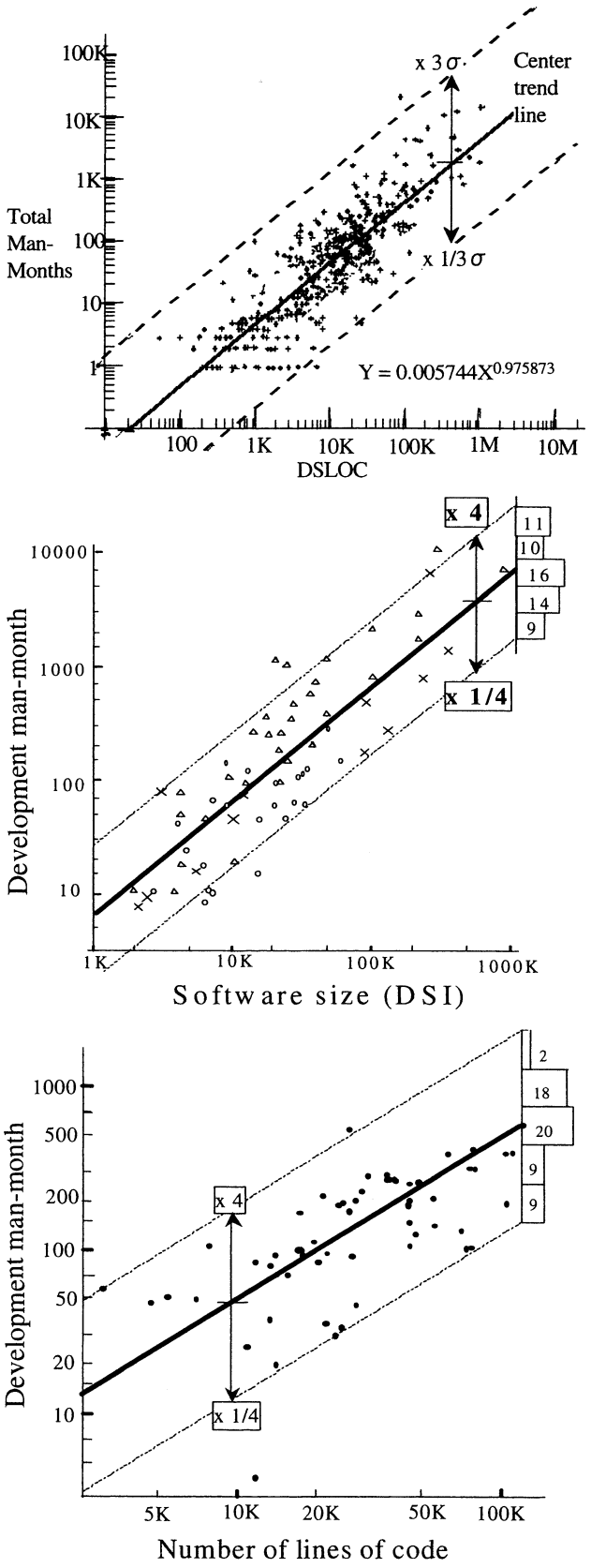

Figure 5. Software size vs. man-month (Both logarithmic scales)

[2] It follows the practice of the Human Reliability Engineering (HRE)[21]. It recommends use of 3 time (N = 3) of the Human Error Probability (average) value for systems evaluation. If it is critical, use N = 5.

conclusion. But, the fact that N = 5 only in this case shows that there must be something particular here. This will be discussed later.

The next problem is to clarify the belt-like zone. In Figure 5.b, 5.c and 6, the belt-like zones, are partitioned to five parallel sub-belts of the same width, and the number of plots in each sub-belt are shown in bar charts at the top right of each figure. They show a bell-like shape, or the facts show a lognormal nature.

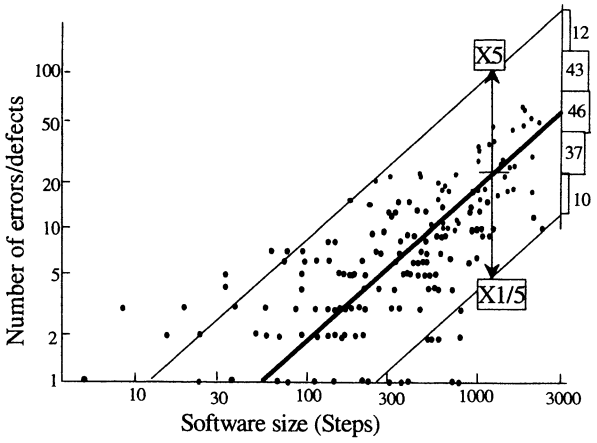

Figure 6. Software size vs. defect (log)

Statistics say lognormal distribution arises as the multiplicative product of many small independent factors. (I.e. epidemiological data and sociological data) In an idealistic lognormal distribution, the plots shown on a log-scale form a normal distribution, and 99.74% of the plots are contained in a range +/- 3·(standard deviation) centered at the average. The standard deviation is proportional to the mean. This is the mechanism that makes the plots show a belt-like zone. Distribution curves in Figure 5 and 6 show their near lognormal distribution nature. Considering the background of lognormal distribution, there are many other similar characteristics.

Figure 7.a[4] shows the growth of "run time." The horizontal axis shows days, and the vertical axis shows the run time in logarithmic scale. It is a "run time" data during an environmental simulation test in the last phase of the system test. A random but heavy load is applied to an online system, and the "run time" is the time from the start to system stoppage by some reasons. When it stops, the cause (usually a software bug) is sought. After it is fixed, the test starts again. The plots show a linear growing trend line, caused by a negative exponential decay of the residual defects. Along the main trend line, two sub-trend lines are drawn, and the normalized number N is written (in the same way as before). From this figure, it is understood that the "run time" shows a lognormal-like distribution multiplied by a negative exponential decay[3], where N is 3.2.

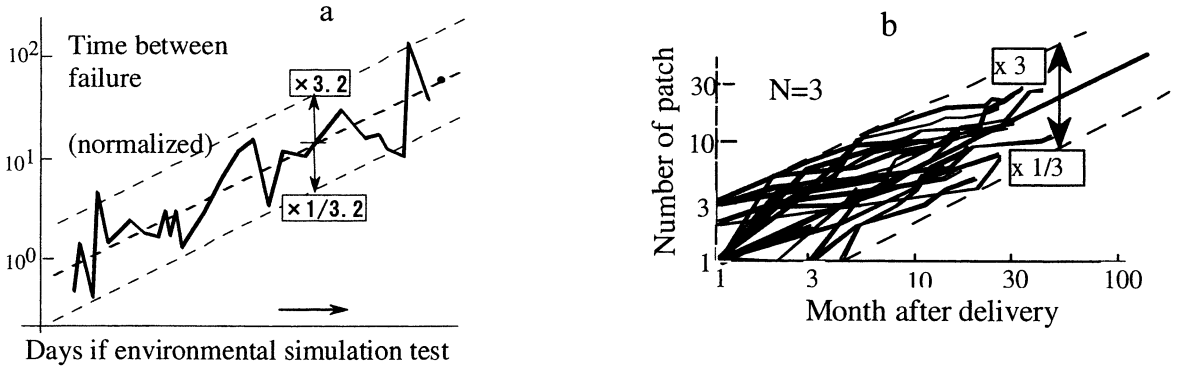

Figure 7. Other lognormal characteristics

[3] The central trend line is named the Kudoh line from the name of the finder. Extending trend lines is possible when the system reaches the desired "run time". Brettschneider proposed a way to calculate the number (day) of the residual defects[19].

Figure 7.b[18] shows the growth of the accumulated number of defects of online systems sharing a mother file. The horizontal axis is "days after the system are put into service," and the vertical axis is the accumulated number of the patches, both in log scales. A system operates with a common mother file and the patches. When some trouble is found, it is analyzed and if necessary, the corresponding patch is prepared. Then, the troubled system is stopped, and all yet un-installed patch(es) including the new ones are installed. The Curves are the loci of each system, and they show a belt-like zone. Also this figure shows the similar trend as Figure 7.a, and N is almost 3.

Figure 5.a (N = 3) case seems to be an all in-house development. Also Figure 7.b (N = 3) is the purest case and Figure 7.a (N = 3.2) suffers from the noise (repairing time). In Figure 5.b (N = 4) (COCOMO data case), there are three plot groups. Exponents of each plot group are as follows; 1.05 for in-house small development group and 1.20 for large development by outside people group and N = 1.12 for the intermediate group. Thus, Figure 5.b is estimated to suffer from the noise by (non-hierarchical design). Figure 6, where N = 5, is the worst case. It will be explained later.

The external characteristics of human-related process should show lognormal distribution, caused by many independent factors influencing them as the multiplicative product. When some other causes operate, as the standard deviation increases the distribution becomes broader, resulting in a larger N. Conversely, when some factors in the process are stabilized, the distribution becomes narrow, resulting in a smaller N. Therefore for their intrinsic nature, aforementioned discussion is correct.

3. Quantitative characteristics of defect removal

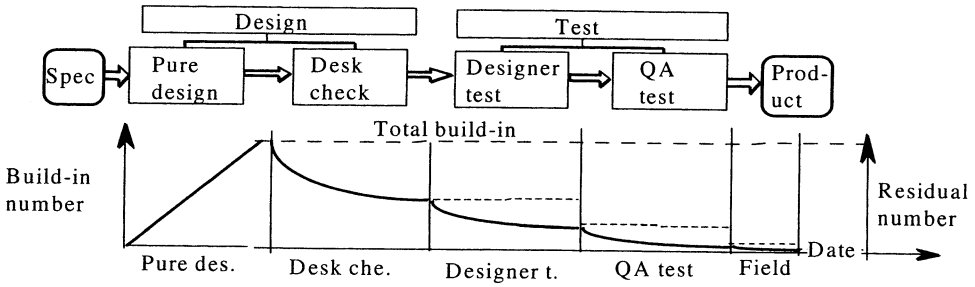

Figure 8. Built-in and decrease of defects

Figure 8[9] in the upper diagram shows a development process. The process flow for design is "pure design" and "desk check," and for the test is "designer's test" and "QA test." (QA is Quality assurance, and some do not have it.) Pure design does not include any checks, and instead desk checks may include any kind of check.

Figure 8 in the lower diagram shows the build-in and removal of defects. In pure design, defects are build-in linearly with time. The total build-in number is E_d and is shown by a broken line. The following desk checks and two cascaded tests are the defect removing process, with negative exponential decay. The number (Ed) is equal to the total number of defects found in following stages, as shown below.

E_d = desk check removed number
+ designer's test removed number
+ QA's test removed number
+ number found after delivery

Eq. (6)

Among the three defects removing processes, the desk check is the most effective.

The defect intensity, namely (the build-in number of defects)/(software size), are the intrinsic external characteristics of a design process. Thus the build-in number of the defects must be measured. This requires that all defects be recorded and removed. Among them those removed during the desk checks are the most important. But many people neglect the importance of the number removed during desk checks. Also Thayers' case neglected these.

Excellent designers remove 80% of built-in defects. However some remove only 10%. In this case, the apparent average defect intensity is decreased from 1 to 0.45, {(0.8 + 0.1)/2 = 0.45}, of the original. In addition to this, a new variation of the amplitude 0.35, {(0.8 - 0.1)/2 = 0.35}, is added on the existing one, which is apparently 0.45. This variation amplifying mechanism is the major cause of the larger value of N = 5 in Thayers' case[4]. Thus, the idealistic "design" builds in defects of the logarithmic distribution[5].

In order to count the removed number of defects, a designer declares the closure of a design, and beyond this point every defect found is recorded. Using thus recorded defects; the exact defect intensity may be obtained after a project. A record of desk checks may be simpler than those for the tests. Surely it needs additional man-hours to take all the records, but without this we might loose sight of the quality.

In the bottom level of the diagrams of Figure 8, both the overall decaying curve and the decaying curve for each section may be regarded as a negative exponential decay. When they are plotted on a logarithmic scale chart, the negative exponential decay will appear as a decreasing linear trend line.

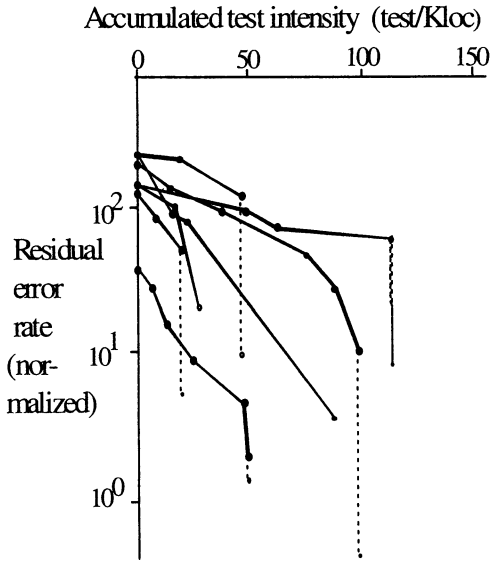

Figure 9. Attenuation by test

Figure 9[4] shows several data that show decreasing trend lines. The gradient of a trend line is named as the "effectiveness of tests" to show the attenuation of defect/test. In the figure, the horizontal axis shows the normalized test intensity (number of test/software size), and the vertical axis shows the normalized residual defect intensity.

The bottom right-most point shows (X = total accumulated test intensity, Y = the residual defect intensity after delivery). The top left most point on the vertical axis is the residual defect intensity after desk checks, or at the start of the test. As the point goes to the left, the number of defects removed is accumulated. As the point goes to the right, the number of tests finished is accumulated[6].

A test is a comparison test of the following two:

---

[4] Thayers' data included other randomness. The data is for routines of one project, and the distribution will have a center axis along the central trend line starting from (X = the total size, Y = the total number of defects) and the shape seems to be like an umbrella without the top part. This explains the spread of the distribution when X is small.

[5] Shiomi in HRE made an elaborate study on 'typing of KB', and proved both the errors and man-hours show a logarithmic distribution by the distribution graph[22].

[6] In Figure 9, vertical dotted lines appear. They are for environmental simulation tests, in the early stage. At that time, various tests of the environmental simulation test were recorded as one test. Later, it was partitioned into many tests.

- program output as the result of so-called design
- expected output derived from a specification.

This test suffers from two kinds of statistical errors. The first kind of error (probability) is to mistake a good item as bad, and the second kind of error (probability) is to mistake a bad item as good. The latter problem is related to the purpose of the tests.

Let us assume the defect intensity of the program is E_D. Then let us assume the second kind of error of the test is E_T. The defect rate after the test/check (the comparison) is $E_D \cdot E_T$. Namely, the initial defect intensity E_D is decreased to $E_D \cdot E_T$. Therefore, a test (desk check) is equal to a small attenuator of defects. It is the reason for the decreasing trend line. But, it should be remembered that these are based on the random nature of defects and tests.

1. If the initial defects are decreased, the curves shift downward.
2. If the number of tests is increased, the trend line is extended to the right by the amount of the increase, and then the final residual defect rate is decreased.
3. If the second kind of error probability is decreased (by making the tests more accurate) the curves go down more sharply, thus the residual defect intensity is decreased.

As these show, following quantitative, rational and scientific treatments are possible in desk check and test.

- Quantitative measurements with quantitative relationships
- Quantitative planning, execution, evaluation and improvement
- Quality control and Total Quality Management

4. Concluding remarks

Part 1 explained the basic structures and the fundamental characteristics of human intentional activity putting emphasis on software, excluding the learning effect. Due to repetitive hierarchical decompositions of human concept, in any human intentional activities, ranging from management (concept level mental operation), design (function level mental operation) and physical works, following fundamental relationships exit.

- Productivity = (The number of output)/(man-hours) = constant
- Man-hours = (The number of output)/productivity
- Time consumed in mental operations causes consumption of resource.
- Defect (built-in) intensity = (The number of defect)/(the number of output)
- (The number of defect) = {Defect (built-in) intensity}•(the number of output)
- Error of a mental operation causes a defect.
- Design is a linear system, where decomposition and integration is possible.
- Defect removing (desk check and test) process is an attenuator of defect, and the attenuation rate is the second kind of the error probability.

These apply in any kind of "process" of all of human intentional activities. This wide applicability based on the scientific foundation is the first salient feature of this system.

Based on aforementioned relationships, Industrial Engineering has brought up various engineering systems, such as production engineering, production control, management engineering, quality control and so on. By them, the present day world has been enjoying prosperity. Software industry has been long staying primitive labor-intensive industry up to now. This system indicates that those engineering systems may

also support software industry. Software works may be managed quantitatively, rationally and scientifically. Thus software industry may become industrialization.

Acknowledgements

One of the authors, Koono wishes to express his deep gratitude to superiors, colleagues and forefront people in Totsuka Works, Hitachi, Ltd. for their kind cooperation. This study is the result of their generous help. The authors wish to express their thanks also to those who contributed to the Software Creation Project. They are also very thankful to Mr. Daniel Horgan for his very careful corrections and valuable advice on their English.

References

[1] G. Salvendi, ed., Industrial Engineering Handbook 1982, John Wiley &Sons.
[2] Z. Koono, H. Chen and H. Abolhassani, An Introduction to Quantitative, Rational and Scientific Process of Software Development (Part 2), SOMET 07, pp. 372-390, November, 2007.
[3] Z. Koono and H. Chen, Toward Quantitative, Rational and Scientific Software Process, Software Process Workshop 2005, pp. 454-458, May 2005.
[4] Z. Koono, K. Ashihara and M. Soga, Structural Way of Thinking as Applied to Development, IEEE/IEICE GLOBECOM 1987, pp. 26. 6. 1-6. 6, 1987.
[5] Z. Koono, H. Chen and B.H. Far B, Expert's Knowledge Structure Explains Software Engineering, Proc. of Joint Conference on Knowledge-Based Software Engineering 1996, pp. 193-197, Sept. 1996.
[6] G. J. Myers, Reliable Software through Composite/Structured Design, John Wiley and Sons, 1979.
[7] M.A. Jackson, Principles of Program Design, Academic Press, 1975.
[8] H. Chen, B.H. Far and Z. Koono, A Systematic Construction Method of an Expert System Used forAutomatic Software Design, JIJSAI, Vol. 12, No. 4, pp.616-626. 1987.
[9] Z. Koono, H. Chen and H. Abolhassani, Knowledge Structure of Human Intentional Activities and Its Work, IEICE TR KBSE2006-56 (2007-01).
[10] A. Kawamata, Electronic Circuits of the DEX-1, Development Report of NTT, Vol. 16, N0. 11, pp. 2275-2306, 1967.
[11] Z. Koono, H. Chen and H. Abolhassani, Knowledge Structure of Human Intentional Activity and its Work, IEICE TR KBSE2006-56, 2007.
[12] Carl von Clausewitz, Vom Kriege, 1832.
[13] Void
[14] R. Nelson, Software Data Collection and Analysis at RADC, Rome Air Development Center, Rome, NY, 1978.
[15] B.W. Boehm, Software Engineering Economics, Prentice Hall, 1981.
[16] M. Kataoka, S. Hanada, M. Teramoto, S. Yoshida, M. Ohba, and K. Fujino, Panel Discussion, State of The Art and Issues in Software Metrics, JIPSJ, Vol. 26, No. 1, pp. 42-52, Jan. 1985.
[17] Thayers et al., Software Reliability Study, Final Technical Report, RADC-TR-76-238, Rome Air Development Center, 1976.
[18] Z. Koono, H. Chen and H. Abolhassani, Quantitative Evaluation of Process and Product and Their Use, IPSJ SIG TR, 2005-SE-150, 2005.
[19] R. Brettschneider, Is Your Software Ready for The Release?, IEEE Software, Vol. 6, No.4, pp. 100, 102, 108, July/August, 1989.
[20] 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.
[21] A.D. Swain and H.E. Guttman, Handbook of Human Reliability Analysis With The Emphasis on Nuclear Power Plant Application, NUREG/CR-1278, SAND 80-0200 (1983).
[22] H. Shiomi, On Analysis and Summarization of Human Reliability Data for Simple VDT Operation, BICRMS 92, pp. 372-377, 1992.