

Structural Way of Thinking For Attaining Reliable Software

**Zenya Koono
Behrouz Homayoun Far**

**Reprinted from
1994 IEEE International Conference on
COMMUNICATIONS**

May 1-5, 1994

Structural Way of Thinking For Attaining Reliable Software

Zenya Koono and Behrouz Homayoun Far
Department of Information and Computer Sciences,
Faculty of Engineering, Saitama University
255 Shimo-okubo, Urawa 338, Saitama, JAPAN

Abstract— This paper describes quantitative relations of software errors from the view point of human errors, and gives a design planning method for attaining a high reliability software. The software design process is a huge chain of human mental processing, during which inevitable human errors occur. By separating build-in and check-out errors, clear characteristics may be obtained, and quantitative measurements are possible. Checks and tests are another type of design suffering also from human errors. Various data are shown to establish quantitative relationships. The quantitative quality design of software development process is introduced.

1. Introduction

Telecommunication systems act as the nervous system of present day society and are responsible for quick and reliable operations. Most of them are computer controlled, so their systems reliability depends upon the reliability of software and hardware. Hardware reliability depends mainly on production errors. Their mechanisms and their preventive means have been well developed. Whereas, software reliability depends mainly on design errors, and preventive measures have not been fully developed, so far.

Software design errors arise from human errors during mental processing. A designer checks-out errors as well as building-in errors. Lack of controlling these two factors invites chaos. In Section 2, the nature of errors are analyzed. Then, a quantitative approach for designing a development process for attaining the targeted quality is described in Section 3. (This paper summarizes the authors' past studies on software errors.)

2. Build-in and check-out errors

2.1. Build-in errors

Figure 1 [2] shows the individual design steps of a software clock. It starts from the top left and, after several transformations, source codes are obtained as shown in the right. A conceptual clock is regarded as an information transformation mechanism from an input *real time clock* to output *clock display*, thus defining the function of *clock*. By intersecting the flow by *time now* and *hands*, the function *clock* is hierarchically broken down and further detailed. *Hour hand*, one

of the hands, is hierarchically decomposed to *angle*, *width* and *length*, and *angle* is defined as 30 times of *hour*, and so on. At the end, logical operations such as

$$\text{angle of hour hand} = 30 \times \text{hour of time now}$$

are converted to source codes. Similar transformations are achieved also on data, until reaching various data declarations. This figure shows traces of human mental operations in the process of detailing. A concept is detailed or hierarchically decomposed at each step. Thus repeating hierarchically, the concept becomes more detailed, clear and crisp, and finally reduced to logical or arithmetic operations. During these steps, an error happens to be built-in and the following steps are influenced by the error until source codes. This probabilistic human error is a software error usually conceived on source codes as an effect.

Software development is a huge hammock net [2, 5] of such human mental operations. During such mental operations, a designer errs inevitably at a small rate (Human error probability; H_{ep}) [1, 3]. When such an error remains in a program, it becomes a software error. Most statistics of the cause of software errors shows that 60 - 70% of errors are rather simple careless mistakes. Though other types of errors exist, such as those caused by misunderstanding and misinterpretation, this human error constitutes the majority of software errors. Hereafter, the discussions go on based on this idea. The quantitative analysis from human error view point will give better understanding to other kinds of errors.

Human errors have been studied in *Human Reliability Engineering* [1]. In it, a human operation is broken down to a series of elementary operations, the total error rate is gained by adding all the error rates of these elementary operations. This is due to the fact that each human error rate is small. In [8] various human error rates for elementary operations are shown. All indicate a wide variation range. Generally, the minimum is one tenth of the maximum. Human errors are very susceptible to conditions and environment. Textbooks on Human Reliability Engineering emphasize difficulties in obtaining a correct H_{ep} [8, 1].

Hereafter, various techniques in Human Reliability Engineering are used for measuring software errors. Let us define *Error build-in rate*, E_b , by the following equation:

$$E_b = (e_b/loc) = H_{epd} \times (N_{mo}/loc) \quad (1)$$

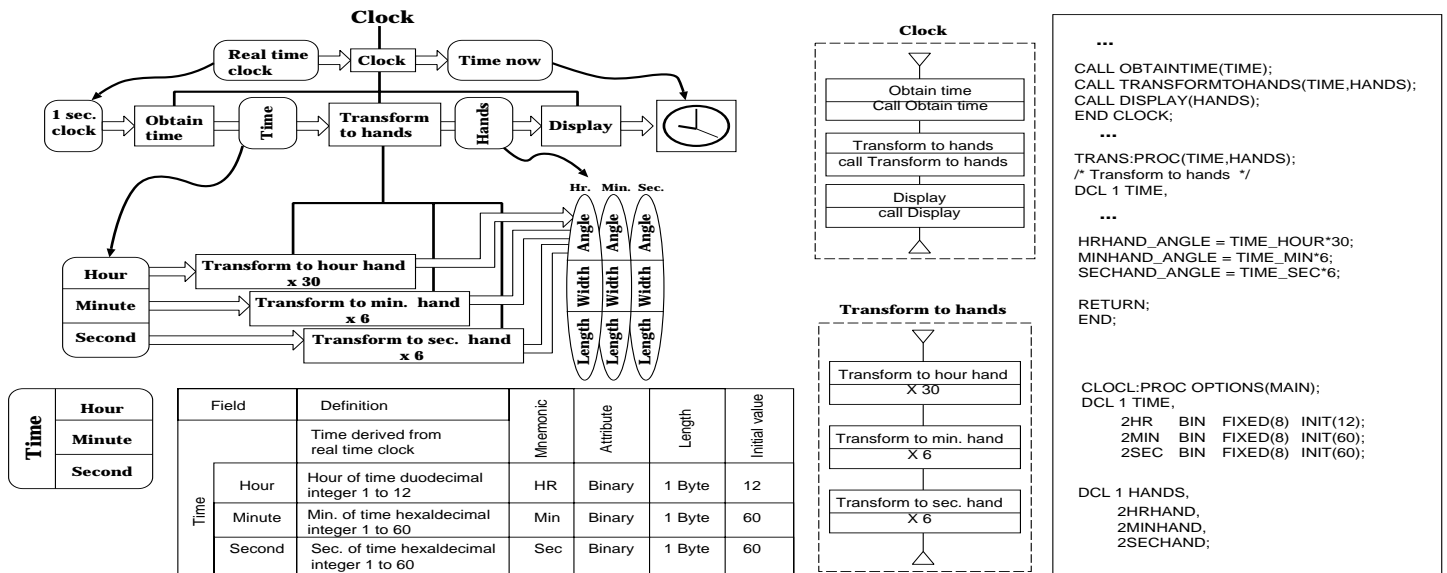


Figure 1: Design steps of a clock

Where H_{epd} is a human error rate during a design step; e_b is number of errors build-in; loc is non-commentary source code line; and N_{mo} is number of mental operations for that design step.

A typical E_b of throughout software design is 10 to 100 errors / 1000 loc.

2.2. Check-out errors

Most designers check their design output unconsciously during the design or are obliged to do after the design. A typical check is to trace the original design, in which the designer makes another design again (in his mind) and compares this result with the previous one. If the designer finds any mismatch between the two results, the designer makes a deep check and gets the correct answer in most cases. When both designs err, errors remain. This means that a check is not free from human errors. The residual error rate is given by the following equation:

$$E_R = H_{epd} \times (N_{mo}/loc) \times H_{epc} \times (N_{moc}/loc) \quad (2)$$

Where E_R is residual error rate after a check; H_{epc} is human error rate during check; and N_{moc} is number of mental operations during the check.

This shows that the original error rate is decreased by:

$$H_{epc} \times (N_{moc}/loc) \quad (3)$$

The *check-out rate*, E_c , is given by the following equation:

$$E_c = 1 - (H_{epc} \times (N_{moc}/loc)) \quad (4)$$

The check-out rate for a whole software development of 50% is attained with any check scheme, but it needs special effort to make it higher than 90%. Building-in errors is made unconsciously at a small rate, while checking-out needs efforts. Without the control or the separation of build-in and

check-out of errors, the error data includes uncontrolled errors and it shows unintended variations¹.

2.3. Evaluations of build-in and check-out

The variations of human error rates, may be avoided by taking a large sample. An averaged data shows a stable value due to the law of the large numbers. Figure 2 [3] shows data for an initial development of a software system, where all the designers experienced it for the first time. Designers finished design after rough checks (i.e. controlled). Then they checked their design (hereafter desk check) and recorded checked-out errors. After all the design was finished, they tested the system with a machine, and recorded machine checked-out errors.

This data is shown in various ways in Figure 2 [3]. In this figure, the horizontal axis is the years of experience, and the vertical axis is the normalized error rate. Figure 2 (a) shows the machine checked-out error rates for each person. (Desk check-out is neglected as other papers do.) Apparently, this figure does not show any meaningful trends. In Figure 2 (b), circles shows their build-in error rates, where build-in error rate is a sum of machine and desk checked-out error rates. This figure clearly shows the decreasing trend of build-in error rates as experience is gained, and good designers show lower rates than bad designers. Figure 2 (c) shows desk check-out rates for each person. They vary wildly. This variation made plots in Figure 2 (a) meaningless.

The fact that design errors are first built-in then the rest of checked-out is a unpublished common knowledge in quality concerned organizations, due to the cause analysis of errors. Experienced leaders know these relations from their experi-

¹ Note that past papers outside from Japanese industries do not take this check-out into account. A considerable part of what past papers on software errors reported contradict with each other. This is caused first by neglecting this and second by various factors affecting human error probability.

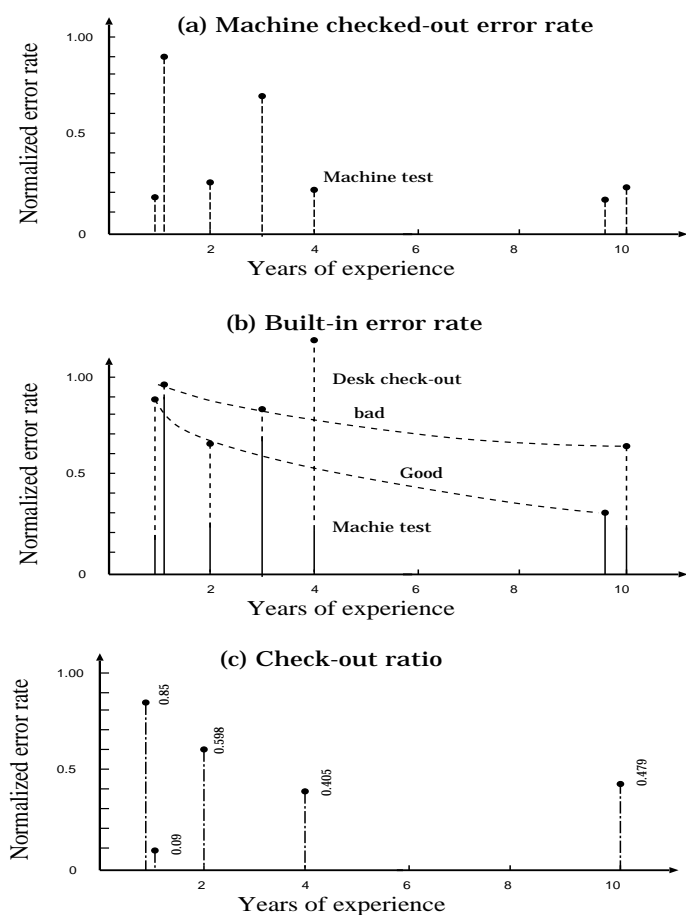


Figure 2: Various error rates

ences thus they can manage their work².

A software development is usually partitioned into many sections, or work processes. During each section, errors are built-in. Figure 3 [4] shows several profiles of built-in errors during design processes. Though various data of programming language, system sizes and applications are taken, they show a trend. (Uncontrolled data neglecting check-out shows a random nature.) Around 1/10 is built-in during the initial phase, specification and architecture design, around 1/2 is built-in during the last phase, detailed design and coding, and the rest in between these two. As human errors are a product of human error rate and the amount of mental operations, these 1/10 and 1/2 trends reflect the ratio of man-

² Much care must be taken in human error related data. The following are several examples:

1. In an ordinary situation, experienced designers show a better performance than Figure (b). The experienced designers are experienced not only in programming but also in the system.
2. In some cases as following, built-in error rates become random:
 - Some people build-in errors unexpectedly due to the his bad condition.
 - When a leader assigns a difficult part to experts.
 - The educational leader assigns new and therefore unfamiliar areas to each maturing designer as a job rotation.

hour allocations. When the man-hour allocation changes, the profile becomes different. If the usual uncontrolled data is shown, they do not show any meaningful trends as before.

To summarize, the meaningful and rational characteristics of software errors can be devised. Errors are built-in and checked-out during human operations named design. They depend on the process in the wide meaning of the word. Quality conscious, data oriented experienced leaders know these relations including their subordinates performances quite well. So far as the conditions are kept the same, the same error related performance reappears where the law of the large numbers exists.

2.4. Evaluation of machine testing

A machine test consists of test design, a run of the designed program, a comparison of the run result with test design, and the repair of errors if errors are found. Therefore, it is essentially a comparison between initial design and test that both may include inevitable human errors. What is free from human errors is only the run of the program, if there is no errors in hardware that the soft runs on it. Therefore, the situation is quite the same as a desk-check, which is also a comparison of error prone several human designs. The fact that a test is also error prone is usually forgotten. It becomes clear when above error rates during tests are measured.

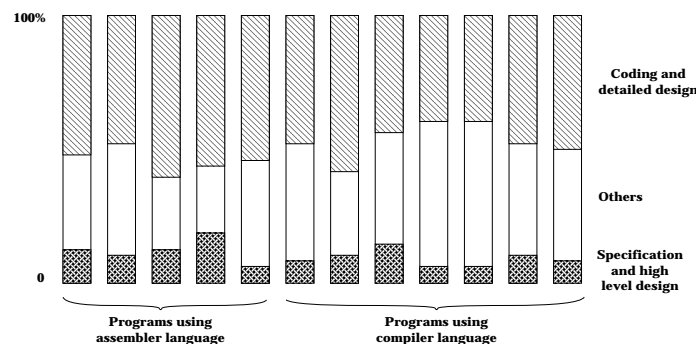


Figure 3: Profiles of errors built-in.

Hereafter, the evaluation of testing is discussed on actual data. Figure 4 [3] shows the plot of a typical progress of a test. The vertical axis is an accumulated number of checked-out errors, and the horizontal axis is the date. Generally, this plot is divided to several regions. In each region there is a rapid increase of checked-out errors followed by a saturation effect, that can be modeled by $B\{1 - \exp(-kX)\}$. Where B is the step value; and X is the elapsed time. Each region corresponds to a certain design and test activity. The first one is for testing by designers, the second is for testing by Quality Assurance people, and the last is errors found in the field. The step values decrease in the later stages.

The authors found the following normalized value for the step:

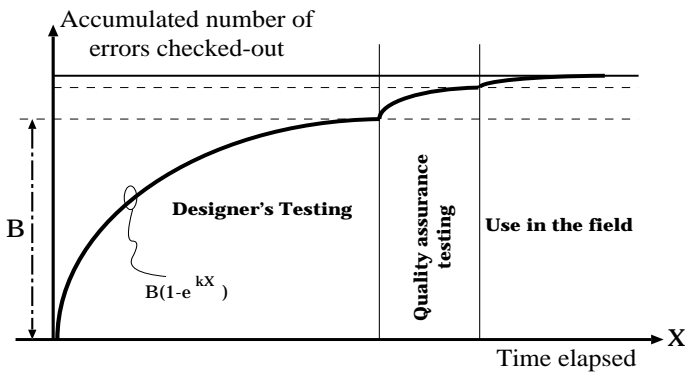


Figure 4: Typical progress of a test

Number of errors checked-out during designers' test	Number of errors checked-out during quality assurance test	Number of errors checked-out by users in field
10	:	1
	:	0.1

These show that residual errors follow similar to Eq.2 – Eq.4. Testing is not the means to check-out errors completely as is supposed to be, because tests all include human errors and the error rate is affected by the design process.

Taking field found errors, the residual errors shown in Figure 4 may be obtained. This is shown in the upper part of Figure 5 [2]. The lower part of Figure 5 depicts the residual errors when the number of tests are taken into account. The vertical axis is a normalized residual (found in field plus during test) error rate and the horizontal axis shows the accumulated test intensity (number of tests/size of the software). Except for environmental tests shown by dotted line, where the number of tests were not controlled, each plot shows a linear line. It shows that the curve is $B(1 - \exp(-kX))$, where residual errors are expressed by $B \exp(-kX)$. The intersection of the vertical axis is the residual error rate before testing, the final plot is the field ship out quality, and the gradient of the line shows the effectiveness of tests. It is found that these correlations are very strong within one group where they share the same standards, working practices, and environment [10].

Figure 6 [2] is another example. In this case, the horizontal axis is the date, and the vertical axis is the interval between troubles. This may be used for the prediction of the date when the system reaches the required stability.

These facts show that the residual error rate decreases at a nearly constant rate as Eq.3, and each curve may be expressed by $B \exp(-kX)$.

3. Improving software reliability

Several factors concerning system stoppages are first described, then the way to decrease software error rate is discussed. Details are given for major relationships.

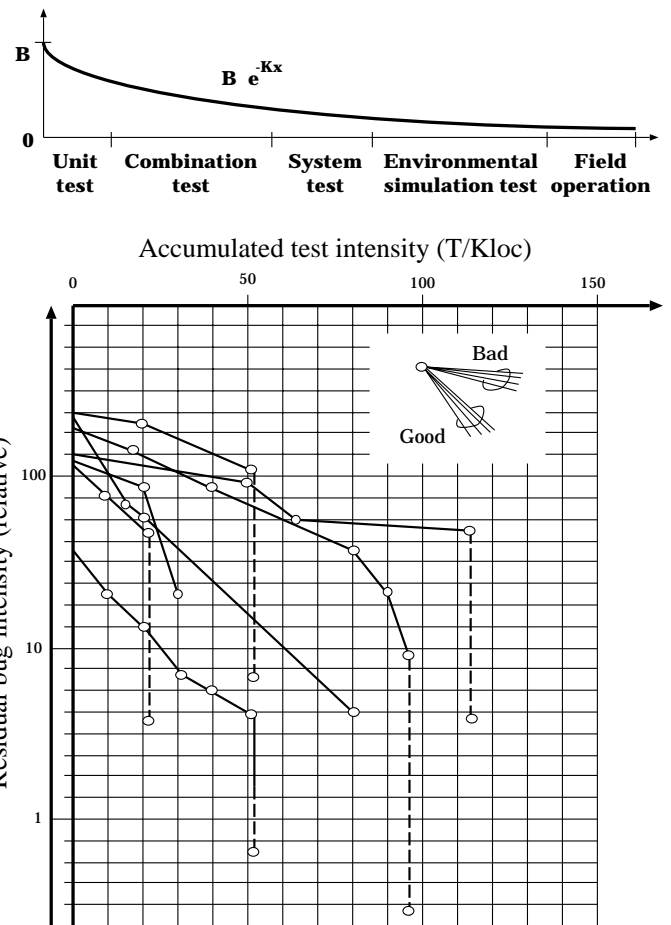


Figure 5: Effectiveness of testing

3.1. Reliability and error rate

Most software errors provoke functional defects. Some of them invite system breakdown. The ratio of system stoppage and software errors shows a constant, as a fraction of errors cause them. This ratio may be measured during environmental system simulation tests prior to the field use. Let us name this ratio the major trouble ratio. It ranges from 1/3 for systems (without so much precautions) to less than 1/10 (for a well prepared system).

In order to make a system reliable, it is necessary to decrease both the error rate and the major trouble ratio. The major trouble ratio is system dependent. It relates to many factors such as software (e.g. programming practices and coding rules) and architecture (countermeasures for troubles), hardware (e.g. various protection systems). These factors may be estimated [9] by applying a similar technique as Fault Tree Analysis.

A practical countermeasure is making a system robust in case of major troubles. System re-initialization is an example. (When a system faces any difficulties in continuing a normal operation, a part of the data, then all the data, and finally data and hardware are reset, and the system configuration is changed and the system is restarted after programs are re-loaded.) Still system stoppages occur where they are

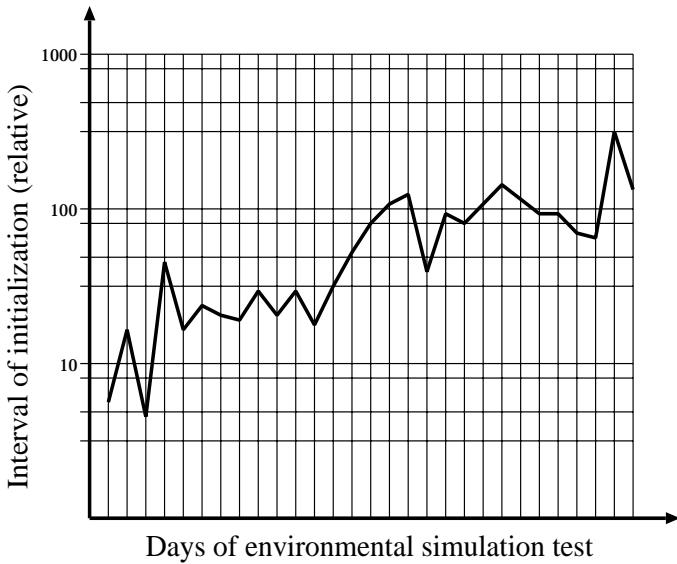


Figure 6: Increase of interval between troubles

not expected, though at a small rate. Let us name it the 'robustness ratio'.

Past experiences have shown that the robustness ratio is greatly affected by the human design errors. Factors missed during system design of normal and abnormal cases, and unintended abnormalities of software and hardware. Fault processing, reconfiguration and re-initialization software have been generally unreliable as they are less tested by abnormal situations and less used actually compared with other normally acting software. Their defects invites total system stoppage which they are expected to cope with. It is important to know that a robust design can not be free from inevitable human errors.

3.2. Software development process overview

A high quality software necessitates a high quality process. Software errors reflect the development process. An overview is given by the past status of a development team. The overall status is visualized by an error built-in profile as shown in Figure 7. The way to get this profile is obvious for the detail of the figure. The residual error rates are visualized in the upper part of an error level diagram, shown in Figure 8. The total sum ending in the coding process is regarded as the error rate before machine testing. This gives a safe side estimation. The lower part of Figure 7 is obtained from the number of tests and the effectiveness of tests described with reference to Figure 5.

The resultant figures show the present status. If the new target residual error rate in the field is set and shown in the figure, the next step is to device the means to attain it.

3.3. Process strategy

It is well known that an error checking-out cost becomes more expensive as the check-out comes in a later step. The cost effective check-out is to check-out errors during their own build-in processes. In order to do it, a strategy is needed.

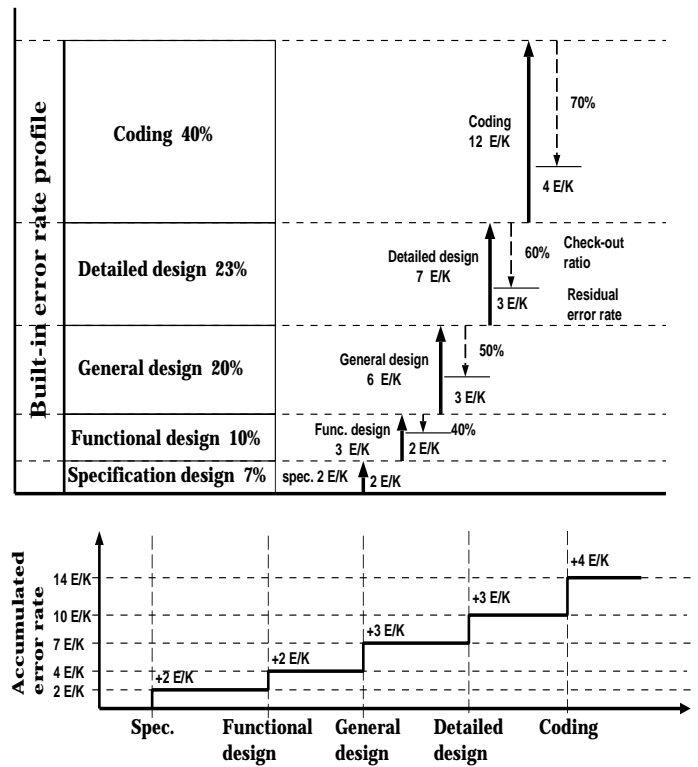


Figure 7: Error built-in profile

The basic hint is:

Repeated (careful and rigorous) checks on documents and making each design divided to smaller sections, each fully documented.

When repeated checks are made, the final error rate, E_f , is given by,

$$E_f = \{H_{epd} \times (N_{mod}/loc)\} \times \{H_{epc1} \times (N_{moc1}/loc)\} \times \{H_{epc2} \times (N_{moc2}/loc)\} \times \{H_{epc3} \times (N_{moc3}/loc)\} \times \dots \quad (5)$$

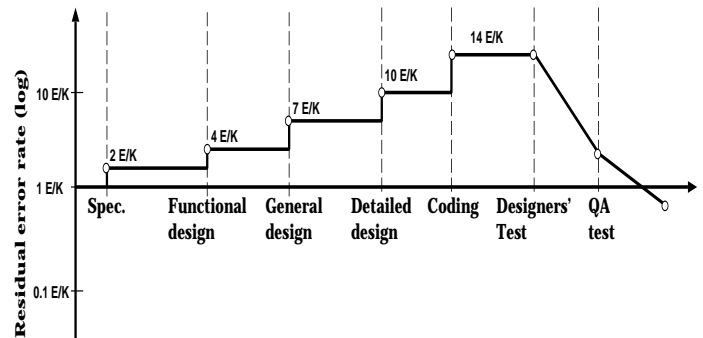


Figure 8: Error level diagram

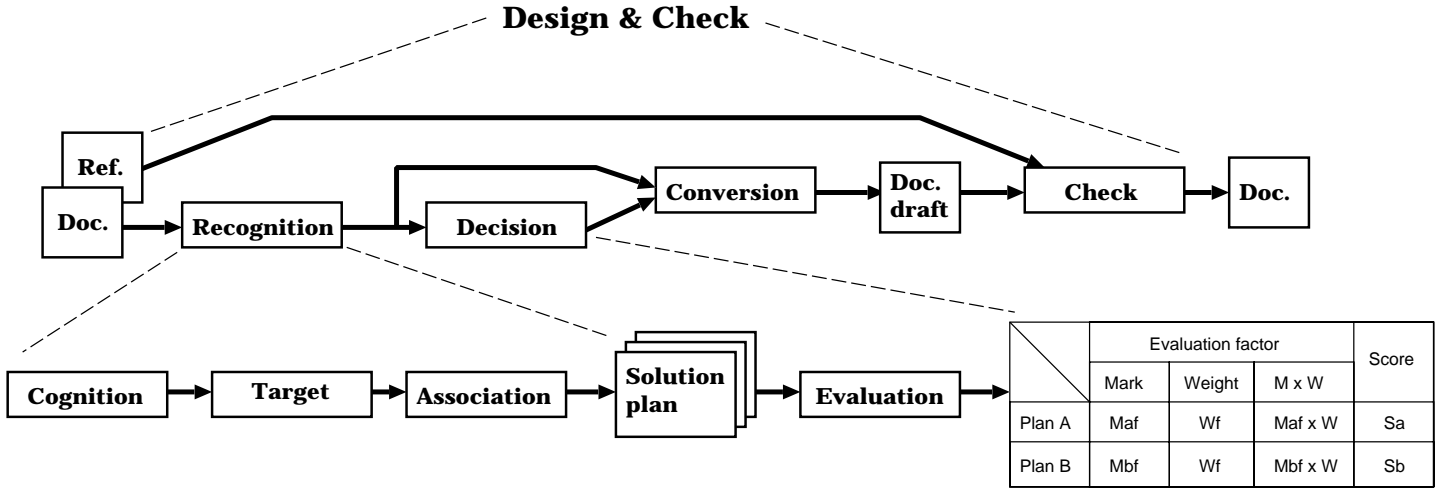


Figure 9: Structural model of design

If one round of checks reduces the residual error rate to 0.2, a double check reduces the residual error rate to 0.04, a triple check reduces the error rate to 0.008. This shows how repeated checks reduce errors drastically.

1. For simple checks such as for codes, the same check may be repeated. The effect is confirmed by accumulating the number of checked-out errors as check progresses. It shows a growth like Figure 4.
2. For higher level design, it is necessary to devise several time checks putting different emphasis each time of the check (e.g. Design reviews from customers view point, maintenance view point, installation view point etc.)
3. If documents are poor, rigorous checks can not be made and the result of checks are sometimes conflicting, resulting in a low check-out rate.

Let us consider a case, where designs are performed by dividing a design section to M equally weighted sections with one check each. The error rates may be obtained as the followings,

$$E_{bp} = \{H_{epd} \times (N_{mod}/M)/loc\} \quad (6)$$

$$E_{rp} = \begin{aligned} & \{H_{epd} \times (N_{mod}/M)/loc\} \\ & \times \{H_{epc} \times (N_{moc}/M)/loc\} \end{aligned} \quad (7)$$

$$\begin{aligned} E_{ro} &= \{H_{epd} \times (N_{mod}/M)/loc\} \\ & \times \{H_{epc} \times (N_{moc}/M)/loc\} \\ & \times M \\ &= E_R/M \end{aligned} \quad (8)$$

E_{bp} is partial error build-in rate; E_{rp} is partial residual error rate; E_{ro} is the overall error rate of whole design; and E_R is the residual error rate of the undivided section.

The resultant error rate decreases to 1/M by dividing to M sections compared with the original. This is achieved by dividing the design section to M sections providing documents at each end of sections and doing checks on them, or sub-dividing a design section to M sub-section of a design providing phased progress of a document at each end of sub-sections and doing checks on them.

The above two cases are discussed for understanding the principles. In an actual work, both of these procedures are used. As both cases show, documentation is crucial. Practically, a build-in error rate higher than 30 errors/KLOC shows lack of proper documentation. Where errors are built-in more, more divisions are needed and respective documents must be prepared.

3.4. Check-out tactics

First, the target for the improvement must be set. It is performed following a Quality Control approach. According to Pareto's principle, the majority of losses come from a few vital causes. Such a few vital, a design section building-in the majority of errors, should be corrected. For effective checks, a correct recognition of the outer side problems and the inner side mechanisms of errors must be made.

Figure 9 [2, 6] shows a structural model for human thinking. The beginning of the design is consecutive decisions, such as what to make and how to make it. The end of design is conversions form data flow to flow charts, then to source codes. In between these two extremes is a mixture of both. Cognition precedes them. As all design should be a checked after a design, check is added at the end. Further decompositions of these operations are possible. The model for decision is taken from Management Science.

Every design process starts from the external input. It is from users, from preceding design stages or from interfacing sections. By fitting this diagram to an error, it becomes easy to understand how errors occur. The counter measure must be provided in *design* as well as *check*, sometimes including testing. The emphasis for those during design should be put

on prevention, and those for checking should be put on earlier check-out. Improvements should be included in the revised work practices and procedures, and check list systems.

3.5. Testing

The decrease of errors depend upon test intensity, and effectiveness of test. In order to increase the test intensity, more details must be written in documents. In order to increase the effectiveness of tests, more clear and correct expressions must be made in documents [2].

1. As the purpose of testing is to check-out errors, strategy and tactics are very important. Such high level test documents are reusable.
2. Each test of a level should be designed to check not only the level (as textbooks say) but also the lower level, as a considerable portion of errors are left unchecked.
3. Test intensity of at least more than 100 test/KLOC is recommended. Proven test data is very valuable, and automatic regression tests should be taken.
4. The reliability of tests should be monitored. The cancel rate of design change triggered by tests, and correction rate of test data are useful measures for this purpose.

4. Conclusion

This paper explains software errors from a human error view point. A software development has a huge net of human mental processing at its background. Software errors are mainly inevitable errors occurring during numerous human operations. By separating built-in and check-out errors, the intrinsic trend of building-in errors becomes obvious and the importance of checks is shown. Such human error view points are explained in many papers.

Each part of development, design, check and testing, is quantitatively evaluated. Quantitative design of software development for attaining the targeted quality (error rate) is introduced. They show that the existing development is already a kind of N-version program, which is much more cost effective than introducing another system.

Several facts about testing have been shown. They revealed that testing is not as perfect as it is thought to be. It is self-contradicting to assume that testing or validation includes no human errors while the design to be tested includes human errors.

References

- [1] Y. Hayashi, “人間信頼性工学—人間エラーの防止技術 (Human Reliability Engineering - Prevention Techniques of Human errors),” (in Japanese), Kaibundo, 1984.
- [2] Z. Koono, K. Ashihara and M. Soga, “Structural Way of Thinking as Applied to Development,” in *IEEE/IEICE Global Telecommunications Conf., GLOBECOM' 87*, Tokyo, Japan, 1987, pp. 1017-1022.

- [3] Z. Koono, Y. Yamato, and M. Soga, “Structural Way of Thinking as Applied to High Quality Design,” in *IEEE International Conf. on Communications, ICC' 88*, 1988, pp. 238-244.
- [4] Z. Koono, “開発 (設計) 作業と誤り発生の構造 (Structures of Building-in Errors During Developments),” (in Japanese), in *9th Software Reliability Symposium, Research Group for High Reliability Software*, MITI, 1989.
- [5] Z. Koono and M. Soga, “Structural Way of Thinking as Applied to Quality Assurance Management,” *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 2, pp. 291-300, 1990.
- [6] Z. Koono, H. Tsuiji, and M. Soga, “Structural Way of Thinking as Applied to Productivity,” in *IEEE/IEICE Global Telecommunications Conf., GLOBECOM' 90*, San Diego, CA, 1990.
- [7] “Structural Way of Thinking As Applied to Good Design (Part 2, Increasing Customer's Satisfaction),” in *Proc. IEEE Int. Conference on Communications, ICC' 92*, Chicago, USA, June 1992, pp. 1665-1672.
- [8] A.D. Swain and H.E. Guttman, “Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications,” NUREG/CR - 1278, SAND 80-0200, 1983.
- [9] T. Tokushima and Z. Koono, “電子交換機中央処理系におけるシステム異常評価の一考察 (An Evaluation of Systems Failures of Central Processing Systems of Electronic Switching systems),” (in Japanese), Technical Report of the ICE, Japan, SE70-8, 1970.
- [10] J. Watanabe, H. Ogata, and E. Kobayashi, “ソフトウェアの品質および生産性予測法の一事例について (An Example of Estimating Software Quality and productivity),” (in Japanese), in *Second Quality Control Symposium in Software production*, Japan Union of Scientists and Engineers, 1992.