

Proceedings of

JCKBSE'96

Second Joint Conference on Knowledge-Based Software
Engineering

Sozopol, Bulgaria
September 21-22, 1996

ORGANIZED BY:

SIG on Knowledge-Based Software Engineering, Institute of Electronics,
Information and Communication Engineers (EIC), Japan
Russian Association on Artificial Intelligence, Russia
Bulgarian Artificial Intelligence Association, Bulgaria

IN COOPERATION WITH:

Japanese Society for Artificial Intelligence
Research Center for Artificial Intelligence, Institute for Programming
Systems, Russian Academy of Sciences.
Institute of Information Technologies, Bulgarian Academy of Sciences
Faculty of Mathematics and Informatics, Sofia University

Edited by Danail Dochev, Vadim Stefanuk, Haruki Ueno

Sofia, 1996

Experts' Knowledge Structure Explains Software Engineering

Zenya Koono, Hui Chen and Behrouz H. Far,
 Department of Information and Computer Sciences
 Faculty of Engineering, Saitama University
 255 Shimo-okubo, Urawa, Saitama 338, Japan
 koono@cit.ics.saitama-u.ac.jp

Keywords: Design, Model of design, Software Engineering, Productivity, Error rate, Learning effect, Software process.

Abstract. This paper reports on a model of design, and by which various aspects of Software Engineering are explained. Design is modeled by a hierarchically expanding network. Productivity, error rate, Learning effect, and various process problems may be evaluated by this model. Requirements for desirable documentation is also described.

2 Model of design

Figure 1[Koono87] shows design progress of a tiny program, a clock. As is shown at the top, design is an information transformation flow from a specification to dataflow diagram, flow chart and finally to code. A part of it, dataflow design, consists of several stages of hierarchical detailing. A dataflow is then hierarchically expanded likewise. As a whole, design constitutes a hierarchically expanding network. (Design process also constitutes a hierarchical expanding network as is discussed in Section 5.) Hierarchical expansion was pointed out as Stepwise detailing, then the concept had been evolved to Structured Programming, hierarchical program and data in Structured Analysis, from the beginning to the end of 1970's. (Above these levels, "Hierarchy of Object" becomes influential which is again a hierarchical expansion.) In hardware, the situation is the same, and various systems have been made to accept only in hierarchy.

1 Introduction

Some Knowledge Engineering specialists are interested in Software Engineering, while some Software Engineering specialists are interested in Knowledge Engineering. This fact shows that both field share the same origin. The origin, however, has not yet been found.

This paper assumes a human mental model for design, and tries to explain the basic relations in Software Engineering quantitatively. Software Engineering in its early days had been good practices which were accepted by many people. It, however, did not have clear theoretical and quantitative basis. This paper makes it clear.

Thus a hierarchical expanding network as shown in Figure 2 is taken as a model for design irrespective of the hardware or software. In the figure, a circle is a human concept, and a branch under it is an elementary mental operation. Based on this model, quantitative evaluation becomes possible. Let us suppose that a design is a chain of n stage in which an expansion is from 1 to r .

Thus, the following may be calculated;

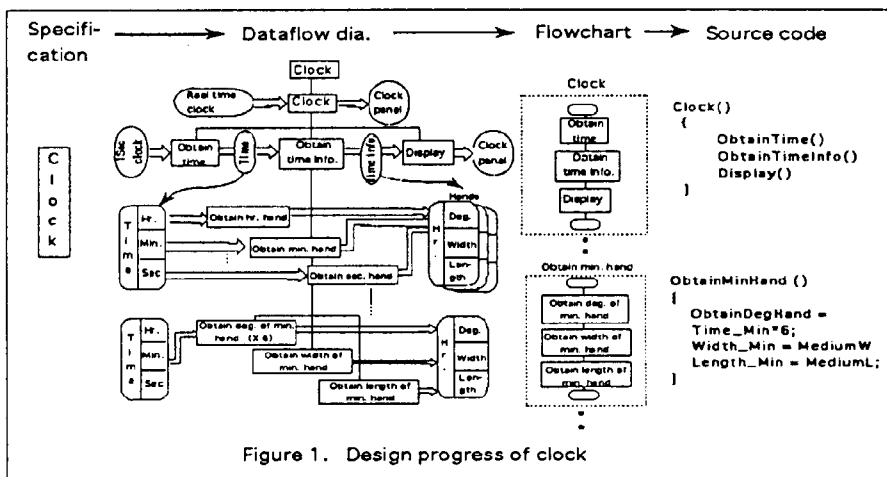


Figure 1. Design progress of clock

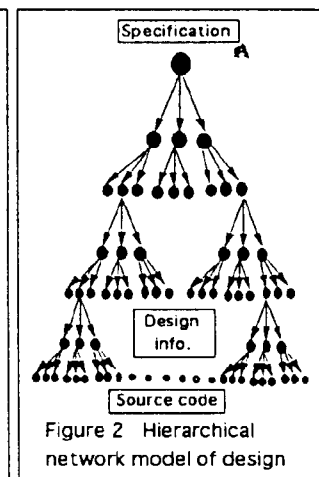


Figure 2 Hierarchical network model of design

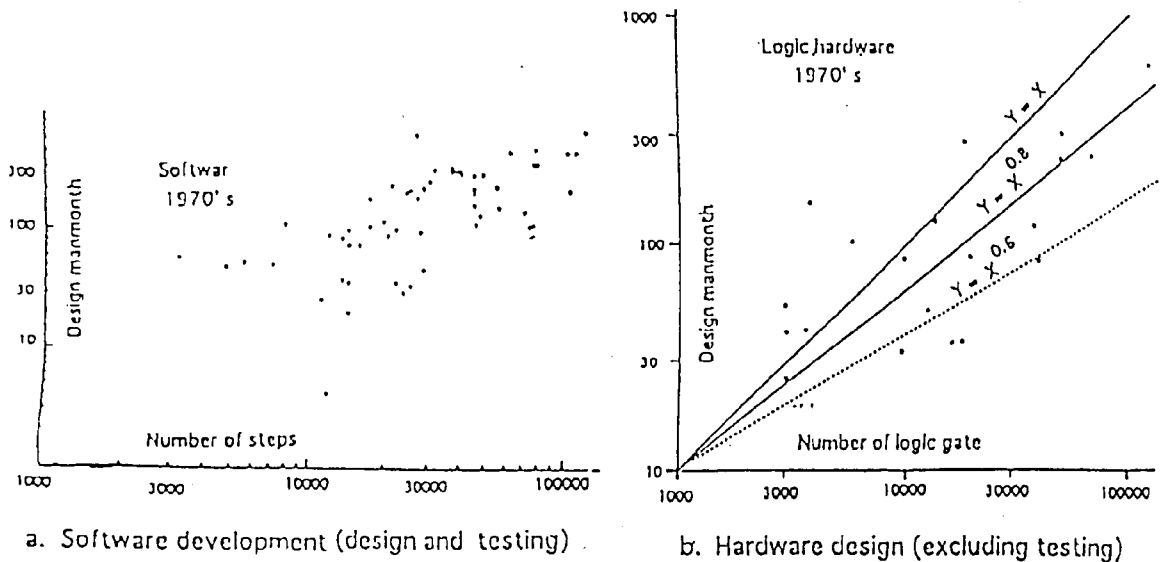


Figure 3 Design complexity vs. manmonth

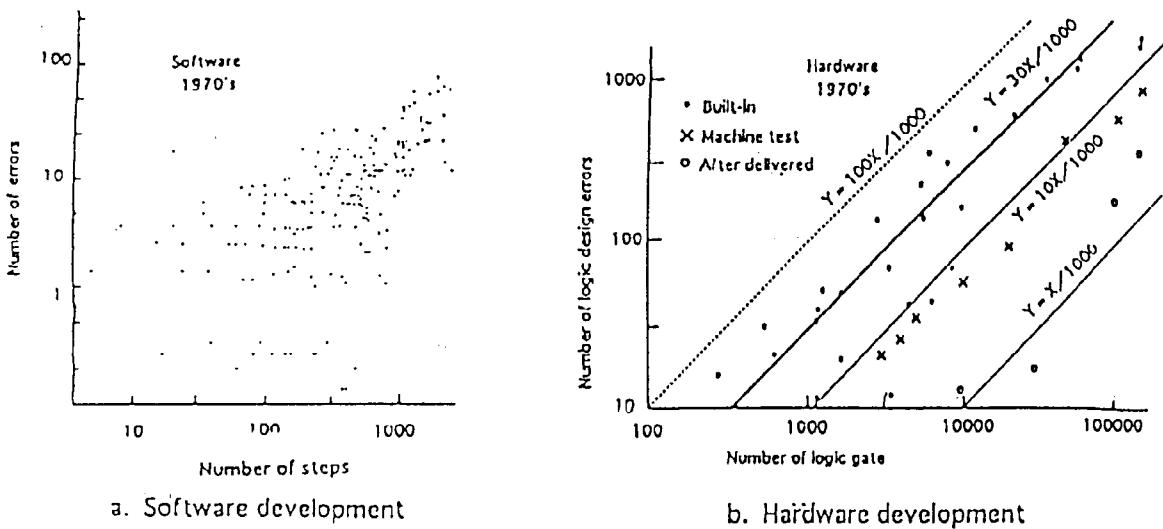


Figure 4 Design complexity vs. error

The number of the output = r^n ,
 The number of mental operations = $(r^n - 1) / (r - 1)$,
 The number of mental operations per output
 = $(1 - r^{-n}) / (r - 1) \approx 1 / (r - 1)$, = a constant.
 (It gives the same approximate equation for a partial series of n stages starting from m -th stage.)
 By multiplying time for a mental operation, it gives;
 "Manhours are proportional to the output".
 By multiplying error probability,
 "Errors are proportional to the output".

Data in Figure 3 for software and 4 for hardware in the front page fit into these relationships. Wide range of data are taken, intentionally from 1970's when initial difficulties for developing a large system had been overcome, most developments were mainly new design and design were made almost by manpower with simple tool of flowchart and logic diagram (later with some primitive Design Automation systems) respectively. Plots of all figures show a wide variation ranging from 1/3 to x3 of the center, which is popular in this kind of problems. It is important to follow a macro view. Figure 3.a {Original data in [Yoshida85]} shows number of design complexity vs. manmonth for development (design plus testing) of software. ([Yoshida85] obtained the trend line is expressed as Manmonth $Y = aX^{0.9}$.) Figure 3.b[Koono93] shows design complexity vs. manmonth for design of hardware. Plots seem to show a trend of exponent around 0.7. At that time, as various machine aids had been advancing and with them a larger development started. Taking this into account, the true trend line's exponent seems higher than 0.7. From these, it is appropriate to understand that the exponent is also roughly 1. From these, data agree roughly with the equation above.

Figure 3.b. and 4.b shows design complexity vs. number of errors. Exponent of trend lines for them seem to be around 1 similarly as before. In Figure 4.b[Koono93] for hardware, x shows the number of built-in errors. The exponent of the trend line is roughly 1, and the rate is around 30 errors per 1000 gates. In Figure 3.b{Original data in [Thayer76]} for software, the number of errors is those found during testing which is built-in minus uncontrolled desk-check-out. The exponent of the trend line is roughly 1, and the rate is around 20 errors per 1000 steps. They agree with the above equation.

By considering the hierarchical expansion model, quantitative evaluation becomes possible, keeping in mind that it is a first order approximation.

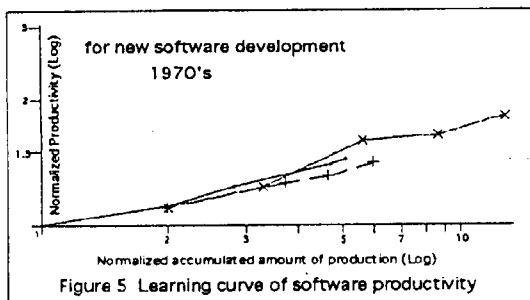


Figure 5 Learning curve of software productivity

3 Learning effect

When a person learns a new game or a sport, that person's records are first rapidly improved, then the gradient of the improvement becomes smaller and smaller, but it continues as that person tries to improve. This trend is called a Learning Effect. In this case, the trend line of accumulated amount of experience vs. record in Log-Log scale becomes linear. This (Logarithmic) Learning effect appears in hardware labor works (Industrial Engineering field) and also in human mental operations such as adding two numbers (Cognitive Science field). Authors found that it appears also in software productivity (Manhours/KLOC) of new (excluding reuse) software development as shown in Figure 5[Koono90a]. Plots are for various software during several years also in 1970's when machine supports had not been used extensively. Nevertheless large differences of each productivity, they follow the same trend line. It must be stemmed from the structure.

When a beginner is assigned to detail general flowchart to detailed flowchart, he/she starts working from basic knowledge of the system as well as programming. This is knowledge based period, requiring much time and many mental operations. As experiences are gained, various commonalities are noticed and abstracted and considerable work is now done by following processing patterns. This is rule based period, with less time and less number of mental operations. By repeating further, on seeing an input the detailed flowchart immediately comes out in brain. Now that person acquired skill, performing jobs speedily without thinking. The number of mental operations has been decreased as experiences are accumulated. It is understood in this model, that the previous lengthy hierarchical mental operation network has been shunted by much shorter one, which is achieved by neural networks in brain. This explains the mechanism of how a Learning effect appears, as well as the structure of knowledge based, rule based and skill based.

4 Error check strategy[Koono93b]

Quality is another essential problem. Quantitative evaluations on strategies for improving quality are explained by the model. A designer inevitably errs at a small rate during design. Let us suppose that a design consist of S mental operations per 1000 LOC and each error probability is H_{ep} , which results in build-in error rate of $E = SH_{ep}$. This build-in rate E ranges from 10 error per 1000 LOC for experts in their expertise area to 100 error per 1000 LOC for beginners for simple programs. (The error rate also shows a Learning Effect[Koono88].) In order to check-out errors, a designers checks design but it inevitably accompanies errors of checks causing residual errors when both err. Let assume as before, the error rate is evaluated as $E' = S'H'_{ep}$, where ' denotes for check. E' is not so good as design resulting from less matured technical level of check. E' is around 0.2 for overall project for

ordinary organizations. From these, following evaluations of residual error rate R after checks are gained;

* design without checks $R = E$,

* design accompanied by checks $R = (E \times E')$,

where design and check is assumed to be independent which may be realized by choosing appropriate means for check.

If E' is 0.2, R is decreased to 1/5. This shows that check during design is a very effective strategy. In a low maturity group where program level errors are dominant, various machine aids are effective for checks. In a highly matured group, human checks such as design reviews and walkthroughs are vital for reducing errors. Fujitsu reported to have attained ship-out quality of 10 errors per million LOC in 1982 by putting emphasis on document reviews [Yoshida85].

There is another strategy. Let us divide a design to M sections of equal number of mental operations for both design and check. The residual error rate R is given by,

* $R = (E \times E')/M$.

When M is 10, R is reduced to 1/10, which is substantial reduction. NEC IC and Microcomputer System has attained ship-out quality of 1 errors per million LOC in 1988 [Kurahara90]. In the company, an embedded software design process consists of 30 types of documents, and M would be more than 20 [Kurahara90]. These elaborate checks during design as well as smaller design processes are vital factors also in Clean Room method.

It should be noted that testing by machine is essentially a comparison of (normally designed) with test designed by run result basis. Extending these, quantitative design of quality [Koono94a] of software design becomes possible.

5 Work process

Next, another hierarchical system, hierarchy of work process, will be discussed. In explaining Figure 2, it has been pointed out that design is divided to dataflow design, flowchart design and coding. These sections are called work process. It is human knowledge to conquer a development process by division. Each of the above process may be divided further to fractional design steps. This hierarchical nature is called work process hierarchy in Industrial Engineering. As a work process is decomposed hierarchically, it becomes clear and concrete, and on continuing expansions finally they are decomposed to human elementary operations.

It is much easier to define the input and the output requirements of a process than to define the necessary procedures themselves. The industrial practice is to organize all the processes in a hierarchical system and define every intersections. In hardware production, interfaces are materials, parts, units and finally an equipment, where processing during a process is a transformation of objects. Whereas in software, being a process a transformation of design information, interfaces are documents. The rule or standard on writing documents act as indirect

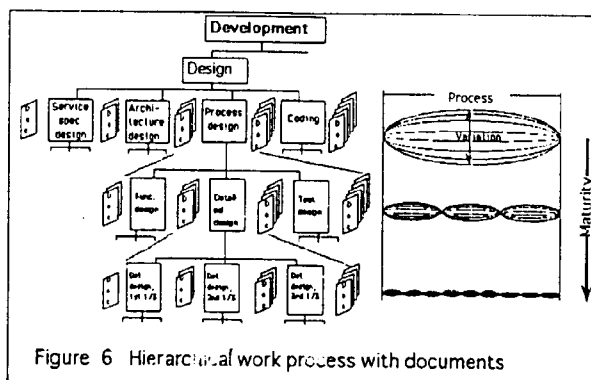


Figure 6 Hierarchical work process with documents

definitions of work processes.

Figure 6 [Koono93b] shows the concept. When a process needs to be specified in more detail, it is hierarchically broken down and to several lower processes and intersections are defined. Therefore, as the hierarchy goes down or document practice is specified in more detail, the process is strictly regulated. The right side picture shows this conceptually.

As a work process is defined in more detail, manhours required for the work show less variation, and by checks at each subdivided process the quality is improved. When operated with hierarchically organized function organization, people are assigned to some part of work process for a particular function or a part of software, and experience repetitive works, then the person shows a rapid progress towards becoming an expert. This is what is done not only in excellent software organizations but also in hardware industry as well as human society in general. For a software development organization to reach such level, standardizations [Koono90b] of functions, products and structures are needed, and it requires improvements for a long period.

Finally in this section, requirements for documentation will be discussed. Figure 7 [Koono95] shows a design information flow between brain and the outside world from a Cognitive Scientific view point. In order for design information flows back and forth freely without loss or distortion, it must be a well organized semantic network of hierarchical structure with expansions less than the limit of magic seven. In order to be understood clearly and easily as well as remembered for a long period, combined use of pictures and texts are recommended. These and preceding requirements will reveal the key for a good documentation.

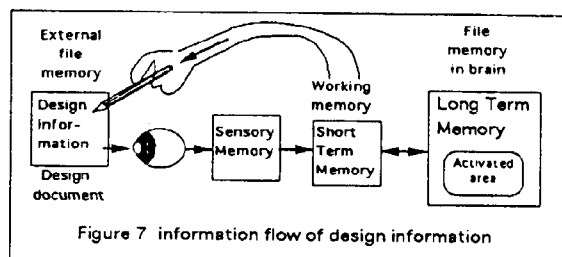


Figure 7 information flow of design information

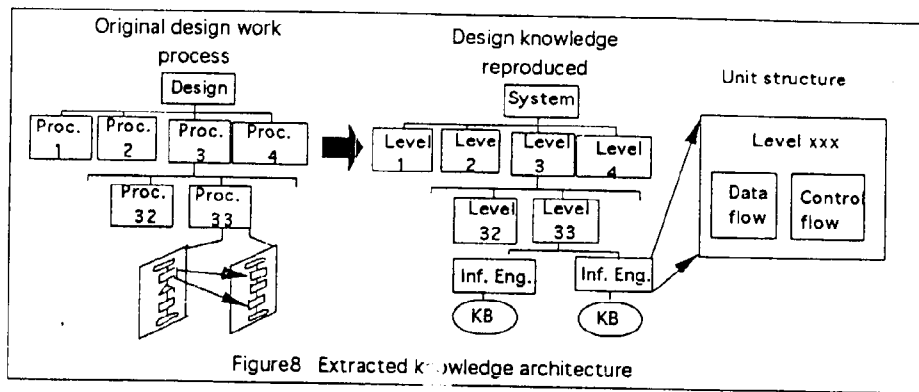


Figure8 Extracted knowledge architecture

6 Future

Finally, outlook for tomorrow in Knowledge Based Software Engineering will be discussed. Automatic software design is a dream for all software people. Figure 8[Koono94b] depicts the idea. Design knowledge at the lowest level of the original work process is acquired as a hierarchical expansion rule from differences of adjacent two documents, and other higher level knowledge may be reproduced in other ways. Thus reproduced knowledge reflects the original work process, and the original design may be reproduced by an expert system[Koono93a]. With accumulating enough design knowledge at the lowest level, low level detailing design may be automated. With studying and implementing high level design knowledge, the system may evolve endlessly. This open ended architecture may be a basis for a future Intelligent CASE tool[Chen96].

7 Conclusion

Software development may be modeled by a hierarchical network of human mental operation. Based on it, various key factors of Software Engineering have been explained and evaluated quantitatively. This supplements what is missed in present day Software Engineering. This also explains any human intentional activities other than design.

Acknowledgements

Authors wish to express their thanks to those participated in this study. This work is supported by scientific research fund by Ministry of Education, Japan.

References

[Chen96]Chen, H. et al.: Software Creation: A proposal of Intelligent CASE tool, Nat'l. Conf. JSAI for 1996, pp. 273-276, 1996. (in Japanese) 陳他: ソフトウェアクリエーション: 知的CASEツールの方式について, 1996人工知能学会全国大会, pp.273-276, 1996.
[Koono87] Koono, Z. and Ashihara, K. and Soga, M.: Structural way of thinking as applied to development. IEEE/IEICE

Global Telecommunications Conference 1987, 1987.

[Koono88] Koono, Z., Igawa, K. and Soga, M.: Structural Way of Thinking as Applied to improvement process, IEEE Global Telecommunications Conference, 1988.

[Koono90a] Koono, Z., Tsuji, H. and Soga, M., "Structural Way of Thinking as Applied to Productivity", IEEE International Conference on Communications 1990.

[Koono90b] Koono Z. and Soga M., Structural way of thinking as applied to quality assurance management, IEEE Journal of Selected Areas in Communications, vol. 8, no.2, 1990.

[Koono93a] Koono Z., Far B. H., Baba T. and Yamasaki Y., Software creation: Towards automatic software design by simulating human designers, The 5th International Conference on Software Engineering and Knowledge Engineering, San Francisco, 1993.

[Koono93b] Koono Z. and Ohtsubo T., Evaluation of build-in and check-out of software errors, Technical report of IPSJ, Software Engineering 95-5, 1993. (in Japanese) 河野, 大坪, ソフトウェアの誤りと除去の評価, 情報処理学会, ソフトウェア工学研究会, 95-5, 1995.

[Koono94a] Koono,Z. et al.: Quantitative design of a software development process, The 4th European Conference on Software Quality, pp. 173-181, 1994.

[Koono94b] Koono,Z. et al.: A systematic approach for design knowledge acquisition from documents, 3rd Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, 1994.

[Koono95]Koono, Z.: A systematic ways of design and the knowledge, The 13th Design Symposium, 1995. (in Japanese). 河野: 系統的な設計の作業方式とその知識, 第13回設計シンポジウム, 1995.

[Kurahara90] Kurahara, H. et al.: Total Quality Control in technical groups, JUSE Press, 1990. (in Japanese)倉原他: 技術集団のTQC, 日科技連, 1990.

[Thayer76]Thayer, T. A. et al.: Software reliability study, TRW System Rep. SS-76-03, 1976.

[Yoshida85] Yoshida, T. et al.: State of the arts and problems of software metrics, Proc. of IPSJ, vol. 26, No. 1, 1985. (in Japanese) 吉田他.: ソフトウェアメトリックスの現状と課題, 情報処理, vol. 26, No. 1, pp. 48-51, 1985.