New Trends in Software Methodologies, Tools and Techniques
H. Fujita and T. Gavrilova (Eds.)
IOS Press, 2011

doi:10.3233/978-1-60750-831-1-38

Principle of Documents for Systems Design Part 1: Non-sequential Case

Zenya KOONO[a,1] and Hui CHEN[b]
[a]*Creation Project*
[b]*Information Science Center, Kokushikan University*

Abstract. This paper discusses the principles of documents for the systems design of a software system without a sequential nature. It reports firstly on the design aspect from "Human intentional activity", and secondly the documentary aspect for human interception. Emphasis is put on diagrams, charts and tables, in essentially the same situation as a hardware.

Keywords. Systems engineering, design, document, human intentional activity.

Introduction

This paper discusses the principle of design documents in the Systems Engineering (SE) phase of a software development, where the input to output relationship of the system is fixed (Non-sequential case). In Software Engineering, the principles of documents have not yet been clarified. Surely documents are for conveying information, but as both sender and receiver are human, human nature must also be considered. This paper intends to clarify the foundation from the two viewpoints of human intentional activity and synergy by drawings and text. The authors focus on the SE phase, which is the early phase of a development, where both hardware and software are equally treated; and documentation must be on an equal level both for software and hardware.

Section 1 discusses the design, which is based on human top down nature, named "Human intentional activity[1]", and applicable irrespective of software or hardware. It constitutes a "process", and is essentially the same as a physical work process, a hardware design process and a top management process. For an actual work, the process must be stabilized enough and error must be few. Documents are tools for attaining these objectives, and thus the requirements of documentation are fixed.

Section 2 discusses the documents. From the readers' viewpoint, an important synergy[2] has been found, which arises when both text and figure/drawing are used for an object. Based on this aspect, an appropriate combination of drawing and text is the pattern of a document. As a result, Data flow (diagram), Data chain (diagram) with Data specification table and finally control flow are seen as important elements in constituting design documents.

[1] Corresponding Author: Representative, Creation Project. Honfujisawa 2-13-5, Fujisawa, 251-0875, Kanagawa, Japan; E-mail: koono@vesta.ocn.ne.jp, URL: http://www.creationproj.org

1. Principle of design

1.1 Structured design and an extension to ESD

A design method effects not only the productivity and quality but also the future extension of a system. As a beginning for this paper, a design method is discussed.

Studies of design methodology started after the 1968 NATO Conference, where the introduction of Engineering was urged. From the first stage "Stepwise detailing" to recent "Object Oriented Design", various methodologies have appeared. However, due to the lack of quantitative evaluations, the comparison has not become clear yet.

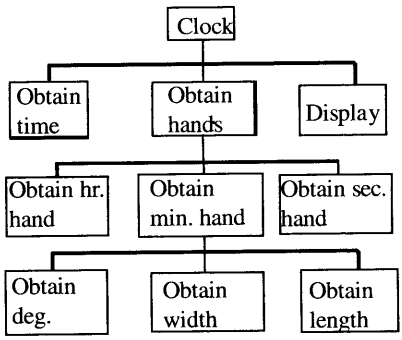


Figure 1. Structured design of "clock"

A survey[3] from JUAS (Japan Users Association of Information Systems) reported that around 1/3 is "Structured Design", 1/3 is "DOA (Data Oriented Approach)" and 1/3 is "Object Oriented Design". "Structured Design" is to organize the target into a hierarchical function system as shown in Figure 1. Many programmers complain of difficulties in decomposing a system hierarchically to form a functional hierarchy. As most of the designers have been trained mainly for programming, their cry is quite natural. Also DOA takes a hierarchical function system by using data flow, and may be regarded essentially as a Structured Design. Thus the problem is how to support the data aspect in design.

The authors have reported[4] a method to make hierarchical design easier. Hereafter it is called an Extended Structured Design and is abbreviated as ESD. As the details have been reported elsewhere, the explanation can be kept to a minimum.

Figure 2[4, 5] shows an example. At the top level, a function "Clock" is a specification to be designed. In ESD, an output data "Clock face" is added, and an input data "Real time clock" is chosen for obtaining enough information for "Clock face". Thus an elementary data flow was formed to be a parent concept, which is used for the following hierarchical detailing. By adding both input and output data, the central function becomes clearer, and also the following detailing becomes easier. Both ESD and

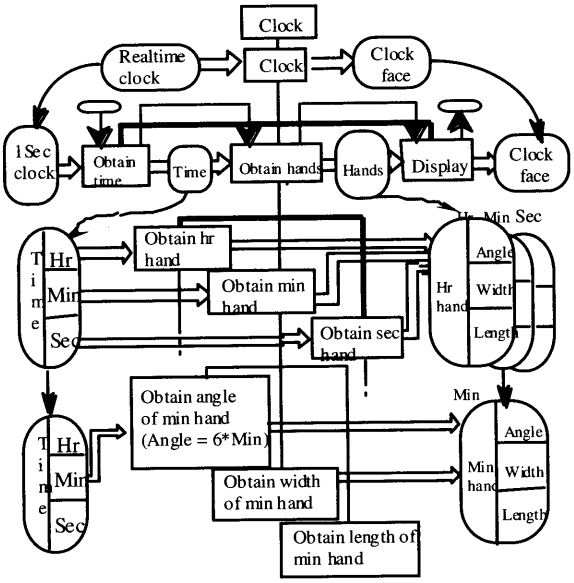


Figure 2. Extended Structured Design

IBM's HIPO[6][2] use an elementary data flow as a parent, and repeats the hierarchical decomposition using data flow. ESD demands that an elementary data flow be mono conceptual, and words or phrases for input, function and output must match the concept, while HIPO is looser to allow even more complex concepts.

An elementary data flow "Clock" in the second-level is hierarchically detailed to a detailed data flow in the third level. It consists of three elementary data flows in a serial manner, "Obtain time", "Obtain hands" and "Display", which are three children concepts from the parent concept "Clock". This decomposition is Myers' STS division[7][3], where two intersecting points "Time" and "Hands" are called *the most abstracted points* from the leftmost input "1 Sec clock" and from the rightmost output "clock face" respectively[4].

This hierarchical decomposition creates a tiny flowchart, which appears in Figure 2. A compressed barrel symbol (starting symbol) above the first children function "Obtain time" is the starting point of the flowchart and the line has an arrow at each end. It passes through the first function, the second function and the third function, and then it terminates at another compressed barrel symbol (terminating symbol) above the third function. This tiny flowchart is created at each detailing to be a part of the parent concept attachment, but not mentioned hereafter.

In the third-level, a child "Obtain hands" is taken for showing the detailing. Both input and output data of "Time" and "Hands" are detailed or hierarchically decomposed to respective children data. Using a corresponding pair of each input data and output data, the detailed data flow consists of three parallel elementary data flows. It is like a Jackson's program development[8].

In the fourth-level, one child "Obtain minute hand" is taken for showing the detailing. As "Obtain minute hand" is detailed or hierarchically decomposed to "Obtain degree of minute hand", "Obtain width of minute hand" and "Obtain length of minute hand". In "Obtain degree of minute hand" it is found that the degree of minute (in a 360 degree system) may be obtained by multiplying 6 times of the minute of time. Thus, this elementary data flow may be converted to a source code block of performing the mathematical operation of degree of minute = 6 × minute of time, in either source code or LSI gate. This is an implementation problem, and the preceding decompositions are common to both software and hardware.

As a result of these investigations, it becomes clear that:

- Design consists of repetitive detailing and one conversion.
- Detailing is performed by hierarchical decompositions of concept using natural language (including scientific and technical terms), thus decomposed concepts becomes more detailed, clearer, and finally minute.
- Conversion is a simple replacement from a detailed concept to a corresponding block of another implementation mean.

---

[2] HIPO[6] by IBM uses a A4 sheet in landscape position of three columns of I (Input), P (process) and O (Output) with the title box in the left side head of the sheet. A function in the title box is hierarchically decomposed to a detailed data flow, who's elementary data flows (input data is written in a square box in I, the processing (function in ESD) in a square box in P and the output data in a square box in O to form an elementary data flow) form a detailed data flow. A function of an elementary data flow is hierarchically decomposed, detailed and written in the following pages. A description is still at macro level and next stage programmers read HIPO and convert to programs.

[3] Myers thought that STS division arises only in case of {external input, function for the processing and the external output}. In ESD, however, a STS division is possible in all cases.

[4] In this example, Time is the farmost but remaining concept from the input data, while Hands is the farmost but remaining concept from the output, respectively.

These apply in other engineering fields, and the above three are universal principles of human nature. But, there are quite different views on design in so-called Software Engineering. The base there is natural language, which plays substantial roll in all human conduct.

Figure 3 shows other hierarchical decompositions. Figure 3.a is a high (management) level example. It was found by Clausewitz empirically and published in 1832[9], and has been regarded as a principle for planning war in Military Science. From the top statement, it is repetitively decomposed hierarchically. It is well known that the hierarchical decomposition continues even down to the elementary actions of a soldier. Figure 3.b is human physical work. An the action is repetitively and hierarchically decomposed to more detailed actions[5]. This decomposition has been empirically used in hardware industry and used as a basic principle of controlling the human process for hardware production. These are achieved by natural language in the human brain for achieving a target. Thus the authors named these actions "Human intentional activity", which govern almost every aspect of human conduct.

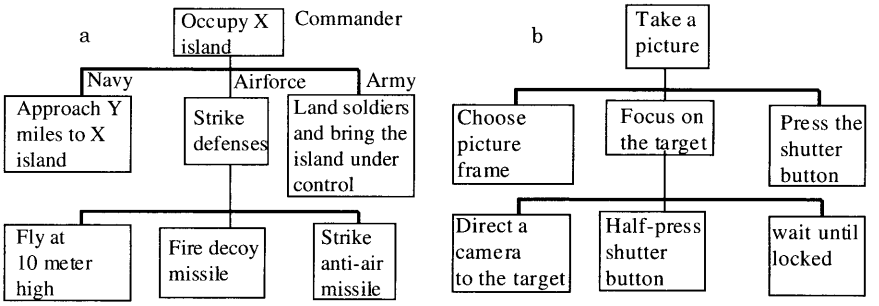


Figure 3. Various intentional activities

1.2 Document enables high quality and stable characteristics

In the previous 2 sections, the structures of ESD have been introduced. This section discusses how documents effect the "design process". Figure 4[1] shows a hierarchical process of a development, and the first half of the development, design, is closed up. In this system, a (design) work, intersected by its input and output is called "(design) process", as is a case of an elementary data flow. Figure 4 shows the "design" at the top. The input is a specification and the output is the resultant source code. It is like so-called Programming.

The right side figure shows the variation. It is like a vibrating string, with both edges constrained. At the top, the largest amplitude shows the largest variation. In the next level, it is hierarchically decomposed to lower processes of data flow design, flowchart design and coding. As this process is partitioned to 3 processes, the

[5] In Figure 2 and 3, all decompositions are hierarchical and the decomposition rate is similar in all three. Past experiences show the average decomposition rate is a little less than three. A theoretical calculation of a simple case shows e = 2.71828... gives the best performance, but the generalization has not yet been made. Considering Cognitive Scientific experiments report the human mental processing time becomes larger rapidly, as the number of decompositions increase. Also the time becomes longer rapidly, as the decomposition rate approaches to zero, due to the increase of the number of stages of mental processing decomposition. By multiplying these two factors, the value, a little smaller than three, is the most optimum.

intermediate 2 points are added to strengthen the constraints. As the diverging power is constant, the variation is decreased to 1/3. If the variation is still large, again the lower processes are hierarchically decomposed to three; the variation is decreased to 1/3 of the previous namely $1/3^2 = 1/9$ of the initial. This is a control method of the variation of the characteristics of design by "divide and conquer" using documents. This manner of controlling a process has been used popularly in hardware production, and a strategic key for "Hi-technology".

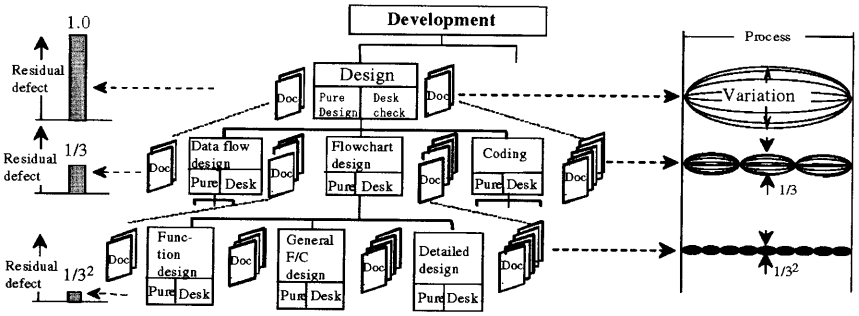


Figure 4. Divide and conquer

Documents contribute to the high quality of a design. In Figure 4, all processes are partitioned to pure and desk at the bottom of a box. "Pure" means plain design without any checks, and "Desk" is check or test after the preceding "Pure" design. The desk check is an "inspection" in Statistics. The design result includes inevitable errors at a rate of Ed. Following desk check errs in two ways: the error rate of the Ist kind of error, mistaking good item as NG, is expressed by Ec1, while the error rate of the IInd kind of error, mistaking NG item as good, is expressed by Ec2. If an error is found during desk checks, the case is examined carefully. If it is OK, the previous NG is cancelled, thus most of the Ist kind of errors are corrected. The IInd kind of error, however, is unidentified, and it passes through the check. The residual defect intensity after the desk check is Ed•Ec2, and the attenuation of defect by the desk check is Ec2. (A series of global test, Ec2 is around 1/10.)

Let us suppose that a process is divided to N sub-processes of equal number of mental operations during a sub-process. After the division, the probability of defects is decreased to 1/N. Similarly, the second error of the check is also decreased to 1/N, and as there are N sub-processes, the total residual defect intensity is reduced to 1/N. This control has contributed to the improvement of quality[6]. The left side bar graphs show this.

These are rather theoretical calculations. Let us review the actual data to show how documents contribute to the high quality. In ESD, a hierarchical decomposition is the adequate small progress of design. Figure 5[1] is a result of this case. The design was made using HIPO, and the number of source code was 147 lines of C code. The desk check was a rigorous check confirming that a parent is correctly decomposed to the children. Checks were made as design advanced, and the checks were made 7 times.

[6] Many people, in their student days, enjoyed the profit of this principle by following teachers' advice "Record your answer step by step. Then check the process carefully and rigorously, one by one."

The vertical axis shows the accumulated number of claims, and the bar graph shows the breakdown of the total claims. The curve is the bug accumulation curve.

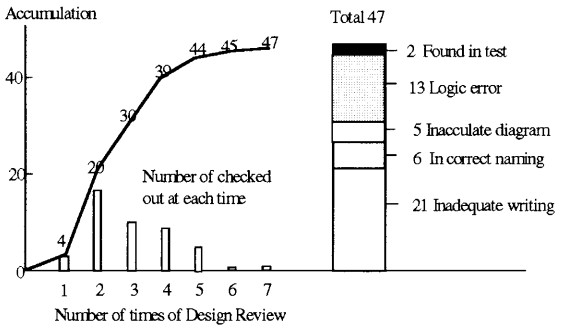


Figure 5. Check in small steps

At first, claims were not about design, but mainly on "how to write HIPO". Next, claims were on natural language expressions. After these, some claims might cause errors in processing logics. At the 6th check, all the checks and the amendments on them were finished, and thus the desk check ended. The designer then advanced to the coding, targeting 'no errors'. Unfortunately, two coding errors were found in the machine test. Among all claims, 13 might cause program errors. Adding 2 bugs found in tests, the built-in defect density was (13+2)/147 = 102 E/KLOC, which is the typical value rate of a junior in university. But, the desk check rate was 86.7%. As the practical upper limit is around 80%, this achievement was remarkably good. The key for achieving this high rate, is the documents and the checks at each small step of progress.

1.3 Documents improves traceability

Traceability in design is to trace backward starting from field claims or some abnormal behavior to discover where the problem was built in. The following explains how it may be done. Figure 6[10,1] shows the development process of a product. Let us suppose a defect was built in process k, but it was not found and corrected in the following tests. The defect appears in the field. Customers experienced the problem, and among them some reported it to the field support.

The claim support people accepted this case and succeeded to recreating the abnormal behavior (1). The designer responsible for this reads the report, and locates the faulty part and repaired the error (2).

The next step is to find out where and what caused the error. Starting from the erred code, the checker traces, or goes up, to the earlier design documents.

The problem is how to identify which process builds-in the error concerned. Examining Figure 6, it is found that

The error built-in process k is a process, having the following specialties:

The output document of process k has some sign of the error, but

The input document of process k has no signs of the error.

Therefore, starting from the erred code (2), trace upward toward the error built-in document (3) in the erred process k (4). Based on these deeper inside the human error is analyzed and the countermeasure to avoid repeating is achieved.

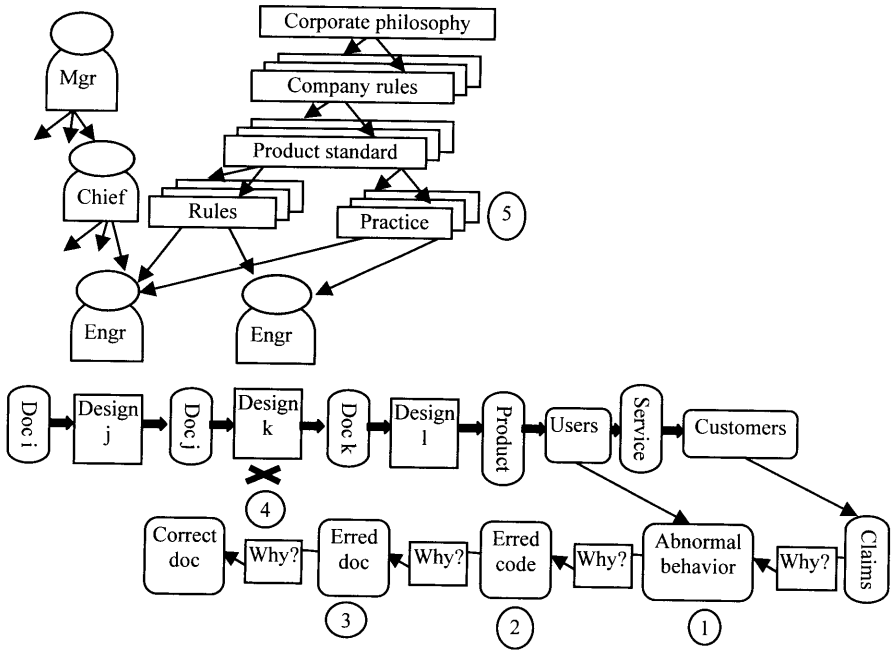


Figure 6. Tracing

2. Design documents

2.1 Structures of development documents

In Section 1, the structures of design have been described. In Section 2, the structures of documents are described. The practical requirement of documents is to compress the volume efficiently.

The first strategy for this is to follow the product' nature. A system document is organized hierarchically following the product's hierarchical structure, mentioned earlier. As all parts part of a system are mutually dependent, this assures the compactness of the document. The second strategy is to use impressive drawings ~ figures, and the third strategy is extensive use of tables and forms. These aim at a substantial decrease of natural language sentences. They are explained later.

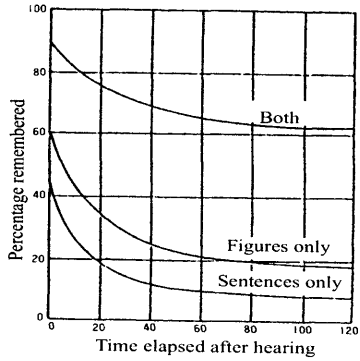


Figure 7. Forgetting curves curves

It is well known that human memory relates deeply to human understanding. Figure 7[2] is from a study in the US Air force. The vertical axis shows the percentage of remembered objects, and the horizontal axis is time after presentation. The presentations were made in three cases.

1. Presentation by sentences only
2. Presentation by figures only
3. Presentation by both sentences and figures

The negative decaying curves are like the famous Ebbinghause's forgetting curves[11]. When both sentences and figures are used, the percentage remembered is much higher than each of both. This shows that synergy[7] arises, when both the figure and the sentence agree well on a target. It is no wonder that movie or video gives best understanding and memory than a book or slides with audio.

A table consists of pairs of some target and its declaration. As each is mutually dependent, only paired relationship exists. Therefore, if each declaration is correctly written, it is OK. A form is similar. In the simplest case various targets are listed, and just writing checks in a space is enough. A general style of a form has several spaces, where, how and what to write is determined. These simplify the answer and the treatment of thus written information.

2.2 Data flow

Data flow is the most important document. Figure 2 shows a sample of a clock from the initial specification of one word of "Clock" to the last one before coding. Reading and examining this figure, a reader understands the design much more deeply than reading a textual explanation of the design, and remembers it better. Various synergies exist in the figure.

1. Elemental figure: graphic symbol with text.
2. Unit concept: easy to remember 3 elementary symbols with text.
3. Human memory-friendly hierarchical relationship: a parent and its three children in both function and data.
4. Multilevel hierarchically structured: from clock to the minutest just before source code. As an example, sophomore students understood this design and remembered it longer Figure 2.

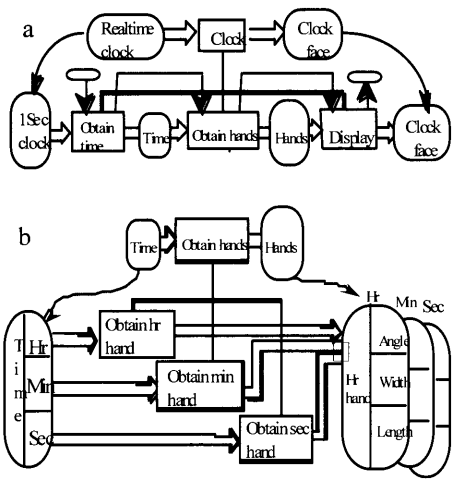


Figure 8. Parent and the children pairs

Athough Figure 2 is fitted for an entire view, a document for each step of design is also necessary. Figure 8.a and b show the unit decomposition of "Clock" and "Obtain time" respectively. Using these, a designer checks if a parent is correctly decomposed, rigorously, and strictly. In it, each parent is shown in duplication. Possible problems may be avoided by marking the parent's information, and a CASE tool administers the entire information.

[7] In a technical context, it means a construct or collection of different elements working together to produce results not obtainable by any of the elements alone. (Wikipedia USA)

2.2 Data specification table

A development starts from a specification (clock), which is defined by an elementary data flow. "Data and algorithm" are two largest parts of the program. In the case of the design of a clock in Figure 2, the whole algorithm is shown by the hierarchical data flows of the figure, but the whole view as well as each data view is also useful.

Figure 9 shows a data chain of "Hands" completed in Figure 2. As is shown in the figure, it has grown up from the initial "Hands" to this chain. This hierarchical data chain impresses the reader more strongly. Another important data in "Clock" is "Time", as both are the most abstracted data of "Clock". If the decomposition continues, further data chains will appear. Temporary and non-relevant data do not need such a data chain.

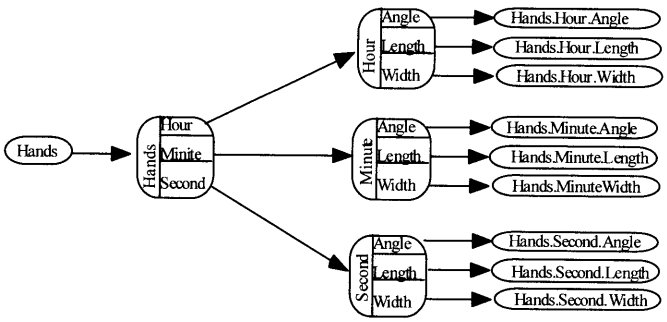


Figure 9. Hierarchical data chain

Table 1 is a data specification table, beginning from "Hands", and it grows as the design advances. (Note the growing steps in the example. Arrow lines of *second* and *third* show the advance, and the last block is the final form of the table.) Figure 9 shows the detailing process. Thus, the pair of both data chain (e.g. Figure 9) and data specification table (e.g. Table 1) are representing documents in a system development document. (For less important data, a data chain is not necessary, and a small structure diagram is enough.)

The table consists of four columns. On the top line of the table, there are the names of each column of lines:

A. Data name and the structure: A small data structure visualizes the structure, useful in cases where a data chain is not shown.

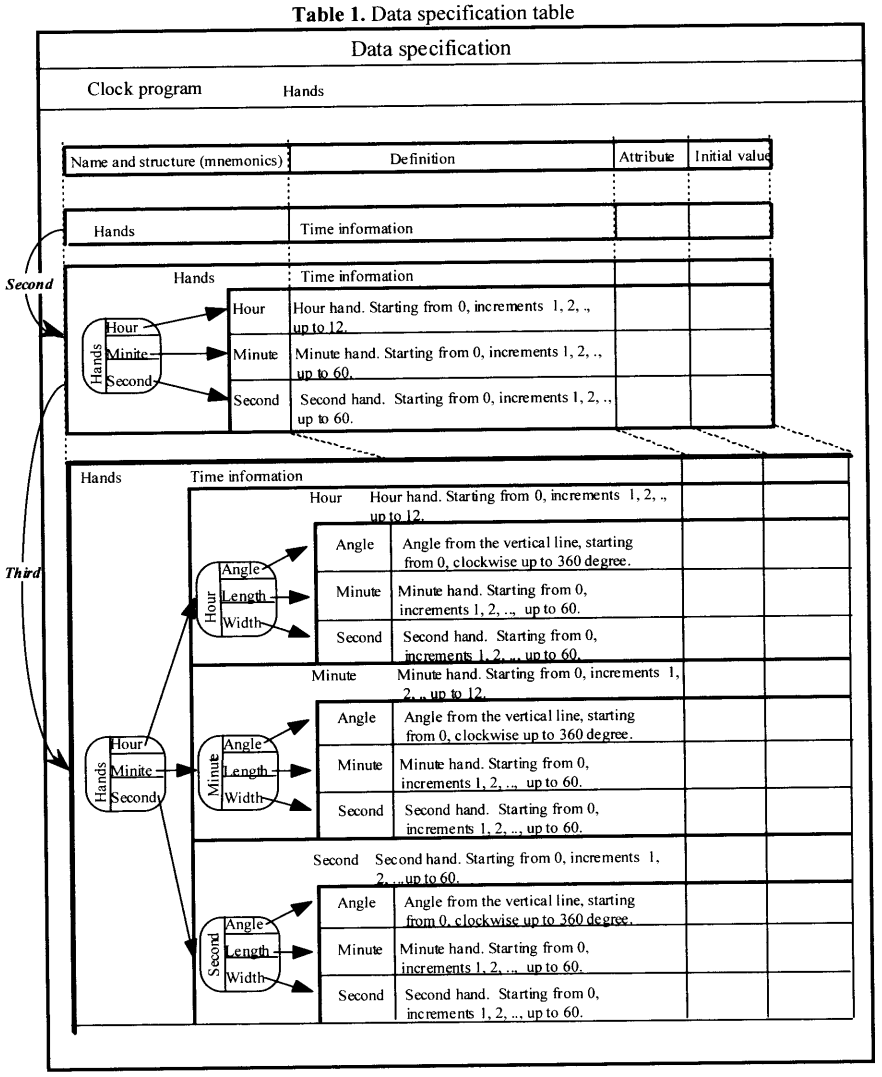
B. Definition: A statement defines the data. A SE must write this directly, clearly and correctly with utmost care.

C. Attribute: Each attribute (e.g. data type, length etc.) has its own area. A column is for each term of the attribute.

D. Initial value: Value at the starting time.

As the design progresses, each column is filled in one by one. In the SE stage, any names must be in natural language, including engineering and technology, but popular abbreviations may be used. By using natural language, people can review any other field, and visitors from other fields can also review and discuss them. Moreover people are obliged to explain their work to their bosses, top management and clients. It is strictlyprohibited to use any local language for software and programs. Based on these SE documents, software people perform the rest of the development work, but they must rigorously follow SE documents, as they are the specifications. If something is wrong or questionable, the person concerned should talk his/her boss first, then discuss the matter with the writer of the document in concern

Table 1. Data specification table



Software people's job is similar to those in hardware production work. Coding may be automated easily using the authors' patent[5,15], which is like production design in a hardware production group. Differing from past software design, they take all the responsibilities as hardware production groups do currently. They have to keep to a specified quality level, specified delivery, and assigned cost. In order to keep these assignments, they are also responsible fnot only for their works' automation but also for corporate level automation, including the various systems concerned. This presents an important opportunity for software people to be treated as an equal member of a company, and to be promoted to top management.

2.3 Control flow

A program consists of components for data processing and a control flow for executing these components as intended. The flowchart has been the earliest tool for the control flow. Although it is simple, it has some defects, such as inefficient expression of loops and a "go anywhere" freedom. After the introduction of a structured principle, various structured charts have been proposed. Their greatest merit is good conformity with the structured principle that makes human thinking easy.

A structured chart is based on a structured principle for using the following three categories of components: concatenation (for a serial execution of functions), selection (one out of many) and repetition (for the loop). Any kind of structured chart is a good tool with natural language for making the control flow easy to understand. Unfortunately, in 1986, ISO tried to standardize structured charts but failed[8], and just a comparison chart[12] was left. Appendix is an abridged table.

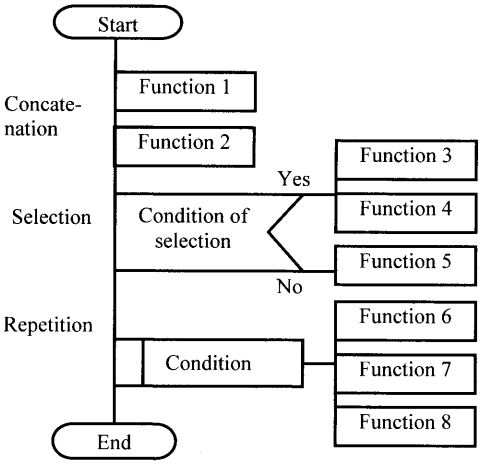


Figure 10. PAD

Figure 10 shows the main symbols of a structured chart, named PAD (Problem Analysis Diagram). In it, the left-side vertical bold line is the route of the control flow to go-downward. The initial two functions are a sample of concatenation. The next wedge shape is a selection. It is the usual practice that an upper line is the YES case. In YES, the control goes to the right and executes two functions, and then the control goes back to the selection and goes down out of the symbol. In PAD, it shows detailing to go to the right side. The next is a repetition, where the first box shows the condition to repeat and the right side extension goes to what should be done. As a result, the shape of a PAD figure looks like the image of a coded source code block.

The problem of control flow is a structural problem in a system. In computer engineering, a central processor (MPU) consists of a "data structure" for data processing and a "control structure" for controlling the data processing in "data structure". There are two technologies of the "control structure". The conventional way is "direct implement" using logic gates, while another is a "micro-program control", in which micro-program orders open and close of gates or mode of operation. In the latter,

[8] Unfortunately, the structured chart had been misunderstood as a tool for programming, and research people have poured their best ideas to it. As a result, all the CASE tools became very susceptible to programming language and the environment, and they suffer from changes by the rapid growth of programming languages. As all the delegates did not want to change own system, there was no hope of standardization. Thus structured charts with the CASE tool had perished, and at present most people still read the list directly. International standardization brings enormous merits, but it is very difficult. The level of standardization must keep to the lowest level so that all delegates agree, the rest must be left as options. As structured charts are one of the influential tools for human thinking, it must be revived as a tool for systems or as an SE level tool in the future, and any programming language problem should be an option level problem.

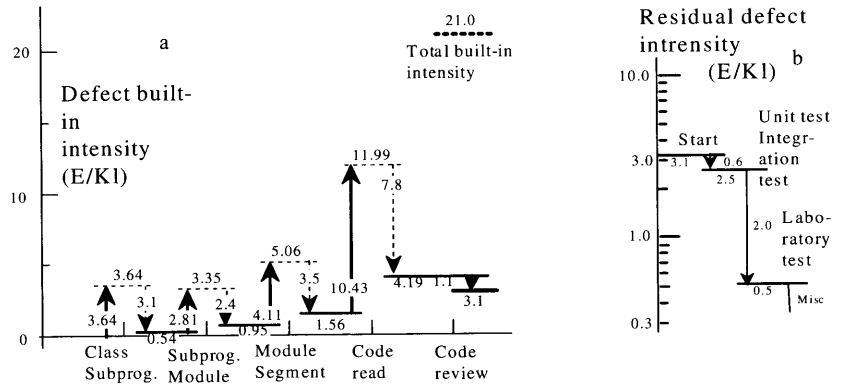


Figure 11. Defect intensity level diagrams

as control becomes easier, a very complicated control may be designed, developed and used widely as is seen in present day processors.

Similarly in software, there are both "data structure" and "control structure". "Data structure" is an assembly of components for processing, as shown in Figure 1. "Control structure" controls these components to perform all the systems functions. In a simple system, a control program controls all components. In the case of a more complex system, the system must be divided into three parts by Myers' STS division. There are input processing subsystems, internal processing subsystems and output processing subsystems. Except for each control, there is a system control over it. As the system is divided, all controls must be easier. Another is to use the Finite state machine for control, namely to design the system as an embedded system. It is the ultimate design of systems[13]. At present most processors are micro-program controlled, likewise FSM control should be the future principle for control.

3. Discussion

What users in industries as well as ordinary people need is *practical improvement of quality, cost and delivery*. The following is a record of a project in the middle of the 1970's, which features a very low defect intensity, achieved with rich documentation and extensive desk checks. Figure11[1].a shows the defect intensity of the design phase, and Figure 11.b shows the defect intensity of test phase.

In Figure 11.a, defects are built in at each pure design, as shown by the upward-pointing arrow. Next defects are removed in the following desk check, as shown by the downward-pointing arrow shows, and thus the residual (built-in minus removal) remains. Thus repeating, the residual defect intensity goes up little by little, but it is only a 3.1 defect / Kline, which is only 14.3 % of built-in defects. They checked out 82.5% of the built-in defects by desk checks after design. Usually, it is very difficult for a team to achieve the desk checkout rate of more than 80%.

The reasons why they achieved such a high desk checkout rate is as follows; firstly rich documentation, secondly desk checks, and thirdly the design progresses in small steps enabling such documentation and desk checks. From the system viewpoint, they took a structured design, and a hierarchical document system, and did rigorous desk

checks after each design. As the progress is small, the range of desk checks is also narrow, and thus the checkout rate became high.

In Figure 11.b, it started at 3.1 defects/Kl. Figure 11.b show the decrease of defect intensity as the test progresses. As a test attenuates, defects at the IInd kind of error, and each test attenuate defect intensity at a certain rate, and the total behavior is shown on Logarithmic scales.

This was a development in a GTE laboratory in the middle of the 1970's. These were reproduced from their paper and their internal documents. One of the world's best records ever published. All the people involved were excellent, and their performance was also excellent. What we can learn now is their excellent process, enabled by hierarchical documents with structured design and extensive desk checks using documents, both prepared in small steps of design. What they did one third of a century ago, we can do the same again.

4. Conclusion

This paper discussed two aspects of a software development in the Systems Engineering phase.

1. Design
   Design is a human intentional activity, and there are many aspects which are common to other intentional activities. In order to enable good performance during design, there is much to be learnt from the example of others.
2. Documents
   In other industries, most documents for a design consist of relatively few using drawing and figures, with least number of natural language sentences. The key is synergy t both drawing and text.
   A similar advance is also possible in software in the System Engineering phase.
3. Learn from other engineering fields
   Fundamentally, there is no reason that programming or software is different from that of other engineering fields.

Acknowledgements

The authors are thankful to those who participated in the Software Creation Project in Saitama University for their contributions. They also express their sincere thanks to the all other people who cooperated, especially the teachers from whom they learnt and experienced so much. They are thankful also to Mr. Daniel Horgan for his careful checks and elaborate collections of their English.

References

[1] Koono, Z. Chen, H. and Abolhassani, H., An Introduction to the Quantitative, Rational and Scientific Process of Software Development (Part 1 &2), Software Methodologies, Tools and Techniques 2007, pp.361-371 and 372-390, H. Fujita and D. Pisanelli (Eds) New Trends in Software Methodologies, Tools and techniques, IOS Press, 2007.

[2] M. A. Broner, Planning and Preparation of Presentation, in R. M. Woelfe (ed), A Guide for Better Technical Presentation, IEEE, 1975.

[3] Japan Users Association of Information Systems and Ministry of Economy of Trade and Industry, Japan, Report of Software Metrics of Information System Users: Year: 2006, Japan Users Association of Information Systems, 2006.

[4] Koono, Z., Abolhassani, H. and Hui Chen, A new way of automatic design of software (Simulating

human intentional activity), pp.407-420, H. Fujita and M. Mejiri (Eds) New Trends in Software Methodologies, Tools and techniques, IOS Press, 2006.

[5] Koono, Z., Ashihara K. and Soga M., Structural way of thinking as applied to development, IEEE/IEICE Global Telecommunications Conf. 1987, pp. 26. 6. 1-6, 1987.

[6] IBM, HIPO-Design Aids and Documentation Technique, GC20-1851-1.

[7] G. J. Myers, Composite/Structured design, Litton Educational, 1978.

[8] M. A. Jackson, Principles of Program Design, Academic Press, 1975.

[9] Carl von Clausewitz, Vom Kriege, 1832.

[10] Koono, Z. and Soga M., Structural Way of Thinking as Applied to Quality Assurance Management, IEEE COMSOC, IEEE Journal on Selected Areas in Communications, Vol. 8, No. 2, pp. 291-300, 1990.

[11] Hermann Ebbinghaus, Über das Gedächtnis, vVerlag von Dunker & Humbolt, 1885.

[12] ISO, Information processing-Program Constructs and Conventions for Their Representation, ISO 8631. (Also Japan Industrial Standard (JIS) X 0128.)

[13] Zenya Koono and Hui Chen, The Ultimate Systems Development Method Based on Finite State Machine, Software Methodologies, Tools and Techniques 2008, pp. 126-145, H. Fujita and Imran Zualkernan (Eds) New Trends in Software Methodologies, Tools and techniques, IOS Press, 2008.

[14] ISO 8631. (Also Japan Industrial Standard (JIS) X 0128.)

[15] Zenya Koono, Hassan Abolhassani, Hui Chen, Priority date, 2002. 5.27, Published in Japan, 2002.12. 5, International publication number WO2002/ 097727. China Paten: ZL 02 8 10859.0, Date: 2006.7.26. U.S. Patent: US 7,480,642 B2, Date Jan.20, 2009.