An application of Cognitive Ergonomics to the Quality Assurance of Software Documentation

Evangelos C. Papakitsos

Secondary Education Directorate of West Attica, Hellenic Ministry of Education.
Department of Linguistics, Faculty of Philology, National and Kapodistrian University of Athens.
School of Electrical and Computer Engineering, National Technical University of Athens.

DOI: 10.6007/IJARCSEE/v1-i1/1308 URL: http://dx.doi.org/10.6007/IJARCSEE/v1-i1/1308

Abstract

The value of documentation is generally recognized as very important for the development and especially for the maintenance of software. It is a prime tool of software quality assurance, which assurance aims at minimizing human errors. The cost of detecting and repairing errors is disproportionally high, in comparison to an initial correct development of software modules, thus proving the significance of qualitative documentation. Standards for writing documentation do exist, addressing subjects like the content, the structuring, the styling and the methodology. What it is not frequently, if ever, addressed is the daily burden of the persons writing or using the documentation, along with the rest of their programming efforts. Cognitive ergonomics is the discipline that deals with minimizing the human effort in accomplishing various tasks. It is argued in this paper that cognitive ergonomics can be used in making qualitative documentation, towards easier to write, understandable and pleasant to use technical manuals for the software developers.

Keywords

Cognitive ergonomics; software quality assurance; documentation standards; systemic modeling; knowledge representation.

1. Introduction.

A typical definition of the term *Software* includes the produced program (algorithms & data structures) in the form of source-files and executable files, and the documentation files (Pressman 1987). The definition can be easily extended to include the accompanying data-files (if any) as well. The discipline of software engineering is the one using a sound methodology for the production of software, according to quality requirements, the available budget and the predefined time limitations (Garidis & Deligiannakis 1993). The quality of software, in conformity with the relevant requirements, is conducted during the activity of Software Quality Assurance (SQA). Consequently, SQA concerns the entire production process (conventionally described by a model like the *Waterfall*), as well as all the various products of this process, including the software documentation.

The importance of documentation is recognized by statements and figures like the following:

- The production of documentation may comprise 20%-30% of the development process (Pressman 1997).
- The quality of user documentation is crucial for the success of a software product (Gialelis et al. 1999).
- Documentation of poor quality can be directly responsible for 30% of human errors (Bailey 1983).
- The volume of documentation is proportional to the volume of programs (Sommerville 1992).

For facilitating the SQA in documentation, various standards have been issued, notably:

- NASA-STD-2100-91 (Software Documentation Standard),
- IEEE std 829-2008 (Standard for Software and System Test Documentation),
- IEEE std.1063-2001 (Software User Documentation)

and numerous others that are used not only for software development (see: Croll 2000) but are also adapted for the training of students in software engineering (e.g. see: SDG 2001; Delaney & Brown 2002). These standards describe:

- what to be included (subjects),
- where to be placed (order),
- how to be presented (formatting).

For example, some best practices that are described include (Papakitsos 2013a):

- how to enumerate chapters and headings,
- how to form the page-numbering per chapter, in order to avoid extensive renumbering if pages are added or deleted, due to future amendments,
- where to place illustrations (photos, tables, figures, charts, etc), namely, near the text that they are referred to.

Nevertheless, unlike the well-known quality metrics (e.g. see: Boehm 1978; Halstead 1977; McCabe 1976) for coding that are presented in the relevant bibliography (e.g. see: Boughton 2012; Leveson 2005; Pressman 2005), for the quality of documentation such kind of metrics do not exist. Because of the reluctance of talented programmers to write qualitative (concise and understandable) documentation, Sommerville (1989) proposes a set of guidelines, consisting of eleven points. Yet, because unlike other engineering disciplines (civil; mechanical; electrical; electronic), in software engineering, these standards are not imposed (Papakitsos 2013b) and because of their vast amount as well, software-producing companies prefer to issue their own guidelines (e.g. see: Berge 2010). It would be very interesting to present this attitude about documentation standards and styles. According to Berge (2010, p.7), much of documentation styles and methods are either redundant or too complicated to be understood by anyone but the creator. Therefore a software-company should use the most common and proven - in various environments - methods that are easily understandable by anyone.

The above statements clearly define the essence of the quest for more quality in software-documentation. Subsequently, we should gather the various aspects and requirements for usable documentation, in order to determine a solution-space.

2. Aspects and requirements.

Humans interact with computers, through software, either as clients or as developers. The User-Documentation standards are focused on the quality of the interface with the client. The rest of the documentation (Technical) concerns the developers, especially regarding the maintenance of the software and the direct preservation of know-how. For a software-producing company, the outcome of software development, besides the product itself, is the commercial profit and the know-how, amongst others (Papakitsos 2013a). The know-how is preserved either in the mind of the staff (indirectly), who may come and go, or in the Technical Documentation (directly). Thus, the latter is a crucial asset for the future of the company, so this paper will be exclusive focused on it.

As it was argued in *Introduction* (1) the features (facts and desirable properties) of technical documentation can be summarized below:

2.i. It comprises 20%-30% of the development process, thus, being a continuous parallel activity of significant effort.
2.ii. Its poor quality can be directly responsible for the 30% of human errors. Considering that the cost of correction per LOC (line of code) can reach up to 40 times that of the development (Pressman 1987), the importance of documentation's quality is huge.
2.iii. It is the main tool for software-maintenance, which accounts for the 60% of a company's activities and of the relevant cost (Pressman 2005). The impact of quality is profound.
2.iv. It has to be understandable, meaning structured, concise and readable (Boehm 1978).
2.v. It has to be simple, by avoiding the redundancy of styles, norms, tools and methods (Berger 2010).

The features 2.iv-v, as requirements, can be dealt with by selecting proper authoring guidelines (Sommerville 1992) and developing/designing techniques. The factors of effort and human error 2.i-iii can be directly attributed to the *mental workload* of the developers. The goal is to minimize this mental workload, which is a subject of Cognitive Ergonomics.

3. Cognitive Ergonomics.

Quoting Cañas et al. (2011):

> "Cognitive Ergonomics is a discipline of ergonomics that studies the cognitive processes at work with an emphasis on an understanding of the situation and on supporting reliable effective and satisfactory performance. This approach addresses problems such as attention distribution, decision making, formation of learning skills, usability of human-computer systems, cognitive aspects of mental load, stress and human error at work".

The above definition is sufficient for the purpose of this article. According to Rasmussen (1983), there are three types of errors that can be caused by a programmer who is very familiar with the developing system:

- Skill-based errors, which are caused by performing poorly but unconsciously some over learned actions of a familiar task.
- Rule-based errors, which are caused by the selection of a wrong condition for a misinterpreted situation.

- Knowledge-based errors, caused in a novel situation where the actions are not correctly planned.

The goal of qualitative documentation should be the avoidance of the first two types of errors. If documentation is perceived as a parallel sub-system to the software-one, then adapting Norman's (1986) definition to this situation, the goal is to create documentation that is pleasant to read.

Either writing or reading documentation requires human effort, which appears as mental workload. When excessive, the mental workload causes undesirable errors. Although there is a debate about the definition of mental workload (e.g. see: Young & Stanton 2002) because of the involved subjectivity, a most generic one is that of JIS (1994), based on the equivalent ISO 10075:1991, for mental stress:

> "The total of all assessable influences impinging upon a human being from external sources and affecting it mentally."

The contribution of Cognitive Ergonomics in establishing guidelines for creating documentation could be summarized in selecting documentation methods that will:

3.i. minimize the mental workload of the writer (authoring effort) and of the reader (using effort) as well,
3.ii. result in simple and understandable description that is pleasant to read,
3.iii. reduce skill-based and rule-based errors.

Certain proposals in specific documentation topics, towards the previous directives, are presented next.

4. Documentation proposals.

Software development can be viewed as two programming process in parallel, one in a computing language (coding) and another in a natural language (documentation). Since these processes are totally related to each other, the same analysis and designing tools can be used for both. Computer Science includes the usage of very powerful systemic modeling techniques, namely SADT, IDEF and OMAS (for an overview see: Papakitsos 2013b), being usable in both stages (analysis and designing) plus the project planning. The latter technique is an attempt to unify the notational techniques, as well, from analysis down to flow-charts. OMAS-III is the latest version (Papakitsos 2013a,b) that will be used in the following proposals.

4.1. Systemic modeling.

The production of documentation is structured along the requirements of the development. The use of modeling techniques aims at providing an outcome of high quality, in an attitude of total quality management.

Guidelines Quality

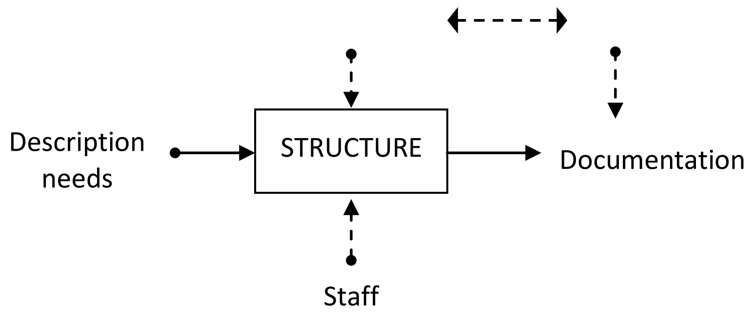

Figure 1: A systemic approach to documentation.

As depicted in Fig.1, *Documentation* is the outcome of specific *Descriptive needs* of software-development. It is *structured* according to the relevant *Guidelines* by the *Staff* of the software-company/organization. There are plenty of standards for structuring and for descriptive techniques but documentation remains badly written (SDG 2001) because many programmers are not capable of writing understandable text (Sommerville 1989, p.577). This particular task requires persons that have a good command of natural language and good knowledge of the programming domain. Such persons will ideally have a first degree in linguistics, philology, communication studies or any of the social/education sciences, then a postgraduate degree (MSc) in Computational Linguistics or Information Systems. He/she will be the *documentalist* of the company/organization, being actually responsible for the proofreading of any documentation (including on-line and interface-dialogues); anything that contains communication in natural language. If the quality of documentation is beneficial for an organization (which is true), then the documentalist is the essential starting point of SQA in this field, as a standard of composition for the development team. This proposal is made from the simple study of the elementary factors affecting documentation, through a systemic modeling technique (Fig.1).

4.2. Notational techniques.

Diagrams constitute a crucial part of documentation. Their clarity is most important for the quality of the description. There are certain sources of ambiguity that could be easily removed, as exemplified below.

4.2.1. Intersections.

Lines intersect frequently in diagrams. It is a source of ambiguity whether an intersection is a branching or a crossing. In the diagrams of circuits (electrical engineering), crossing can be depicted by a bridge (Fig.2a). Branching is very common in hierarchical charts.

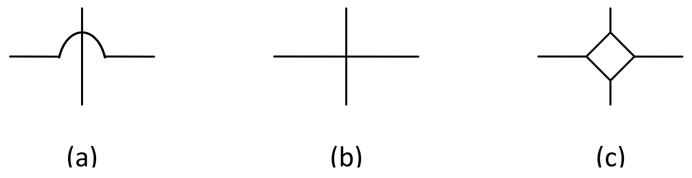

Figure 2: Depicting intersections.

A unified approach of line-styling could be to keep a simple intersection as branching (Fig.2b) and to denote crossing by another symbol, easier than a bridge to draw with a common office-tool, (less effort), like a small diamond (Fig.2c). Wherever possible, intersections should be avoided.

4.2.2. Hierarchies.

The positioning of shapes in the available space of a figure is an art difficult to formalize. Some potential guidelines could be:

i) not to use shapes that occupy more space than necessary, according to the size of labels,
ii) to avoid intersections, wherever possible (*4.2.1. Intersections*).
iii) to clearly denote hierarchical precedence, in a unified manner.

An example is seen in Fig.3 (a modification from: Croll 2000, p.5) & Fig.4.

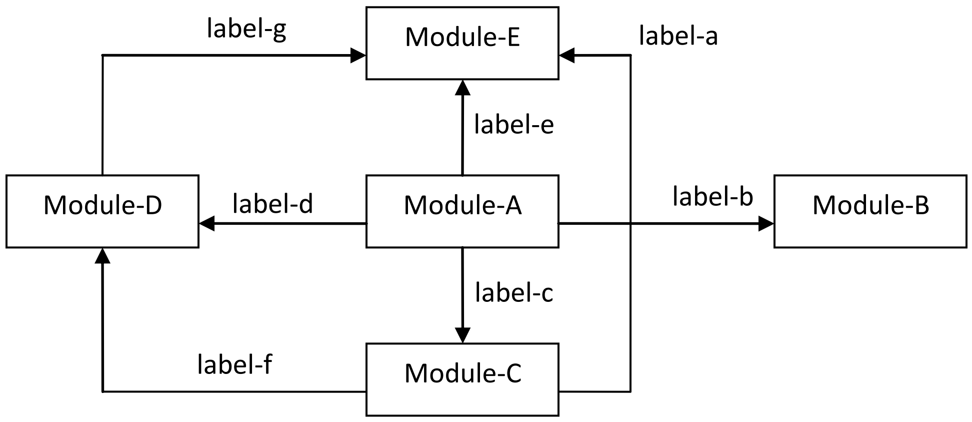

Figure 3: An example of a diagram-arrangement.

We may see an intersection (4.2.2.ii) of the lines labeled *label-a* and *label-b* and we observe that although the rectangle labeled *Module-A* is a starting point, according to the arrows, its hierarchical importance is not clear (4.2.2.iii). Another diagrammatic solution, conforming to the suggested guidelines (4.2.2.ii-iii) would be the one in Fig.4.

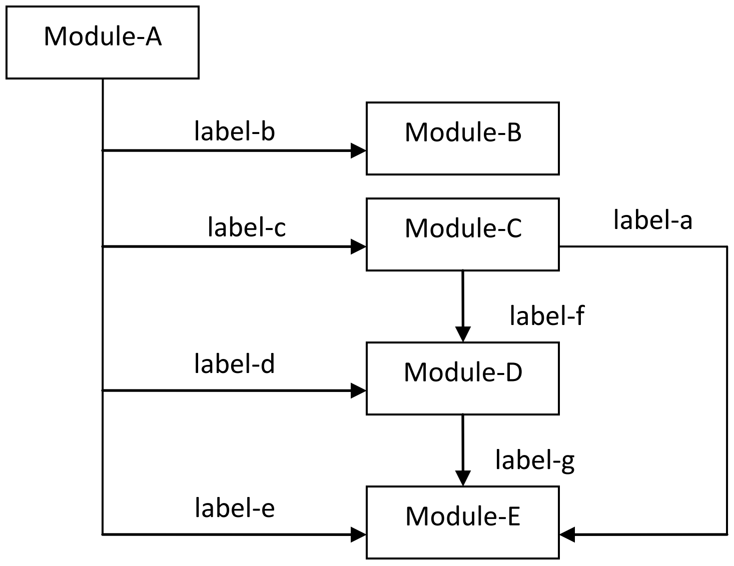

Figure 4: An alternative presentation of Figure 3.

4.2.3. Flow-charts.

In flow-charts the selection (IF-THEN-ELSE) is depicted by a labeled diamond (rhombus), which is probably the worst-possible shape to use, regarding the occupied space according to the size of the label. It is strange that when flow-charts where early introduced, some forty years ago (CMEM 1974) the selection was denoted by a horizontally prolonged hexagon (Fig.5), which is much more efficient in this respect.

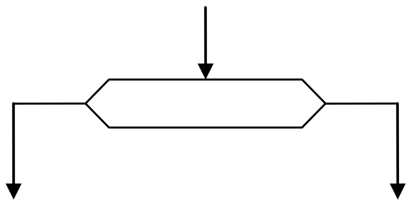

Figure 5: Horizontally prolonged hexagon.

4.3. Data modeling.

A unified depiction for any kind of data structures can be achieved by using tables. In fact, tables are the closest possible representation of knowledge that is directly implemented by variables, arrays, user-defined types (UDT's) and classes. The original notion of *frames* in Artificial Intelligence (Bench-Capon 1990, p.89) is nothing but a table of two columns, one for the attributes and the other for their corresponding values. The rows of the table are as many as the attributes of the represented entity.

Talking about entities, the Entity/Relationship (E/R) model of database-design (Date 1990, pp. 579-593; Chen 1976) is among the most popular and understandable methods of data-modeling. The accompanying diagrammatic technique includes rectangles for entities, diamonds for relationships between entities and ellipses for the properties of entities. Lines of various types connect entities, denoting quantitative and qualitative relations. The notion of entity is equivalent to the frame. It can be depicted by the heading of a single-column table, each row of it being a property. Relationships can be substituted as labels to the connecting lines instead of diamonds. Lines themselves can be dashed or double to denote a different property of the relationship. Arrows of different shape (bullet; angle; triangle; diamond) may also denote certain relationship types, in order to pack more information in the diagram. Special characters (e.g. #, &) in front of labels can be also used for the above purpose. A comparison can be seen in Fig.6, between a standard E/R diagram (Fig.6a) and an OMAS-III one (Fig.6b).

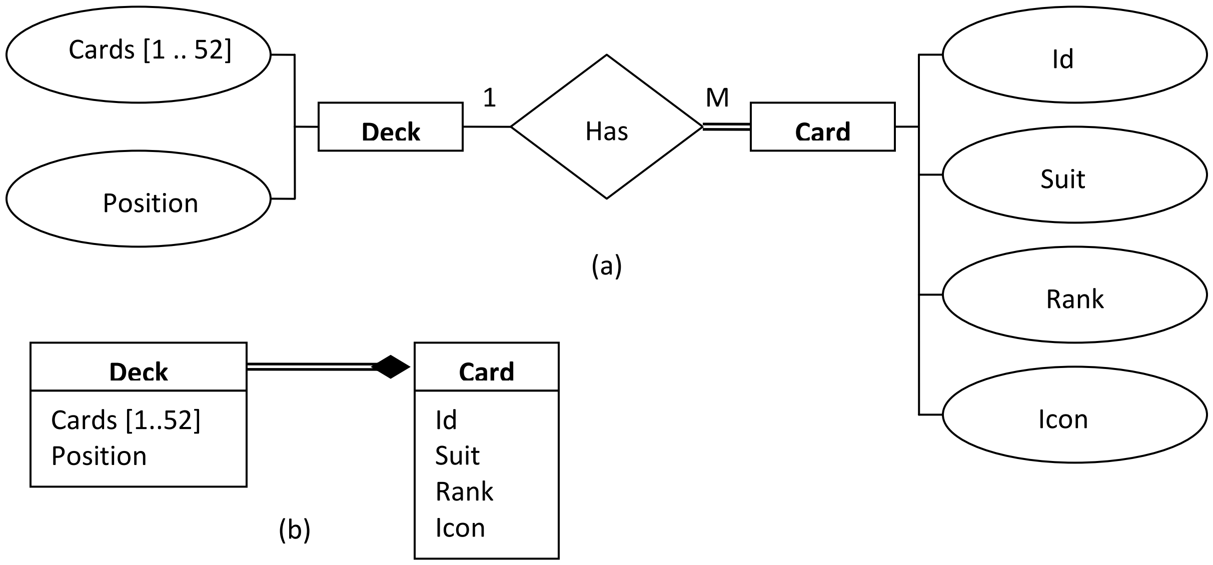

Figure 6: Comparison of different diagrammatic notations.

Classes of Object-Oriented Design (O-OD) are also depicted by tables. Their only difference from *entities* is that they include procedures as well (methods; events), that perform the manipulation of properties' values and the interaction between objects (Fig.7: a modified example from Kytagias & Psaromiligkos 2004). A uniform depiction of *classes* and *entities* may also facilitates data-storage from an application of object-oriented programming to a relational database. The relationships between *classes* can be also depicted by lines of the same semantics to E/R-diagrams. By the way, *classes* could be identified through OMAS-III, at the designing stage. During the decomposition of the overall task in modules, those of them that receive a set of unique input could be identified as a class/sub-class.

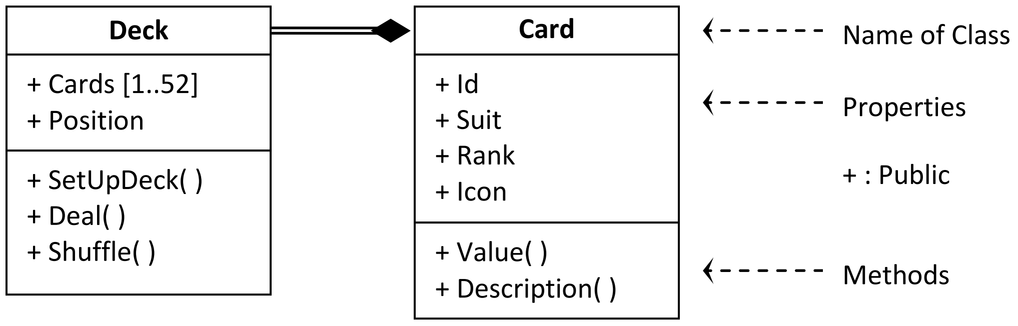

Figure 7: Object-Oriented diagrammatic notations.

4.4. Printing arrangements.

The author, in his early fifties, can hardly stare at a screen for more than two consecutive hours, before a migraine starts. The long-terms solution for a professional is not and should not be pain-killers, eye-drops or even bionic eyes! Printers still achieve better resolution than screens, so his modest advice herein, as an academician, is “more paper-work” for better vision. Consequently, manuals are indispensable; therefore using them should not be a daily burden. The suggestions about their printing arrangements, that briefly follow, are the outcome of twenty-two years of experimentation in writing documentation, in order to deal with specific issues for the ergonomics of reading manuals.

The first issue to be discussed is the reading of diagrams and their descriptions. Often, the description of a complex diagram, occupying a whole page, is expanded in more than one written pages. This expansion obliges the reader either to memorize the diagram or to continuously go back-and-forth. The initial thought was to repeat the diagram (i.e. have it printed) on the even pages, having the corresponding description on the odd-pages (Fig.8).

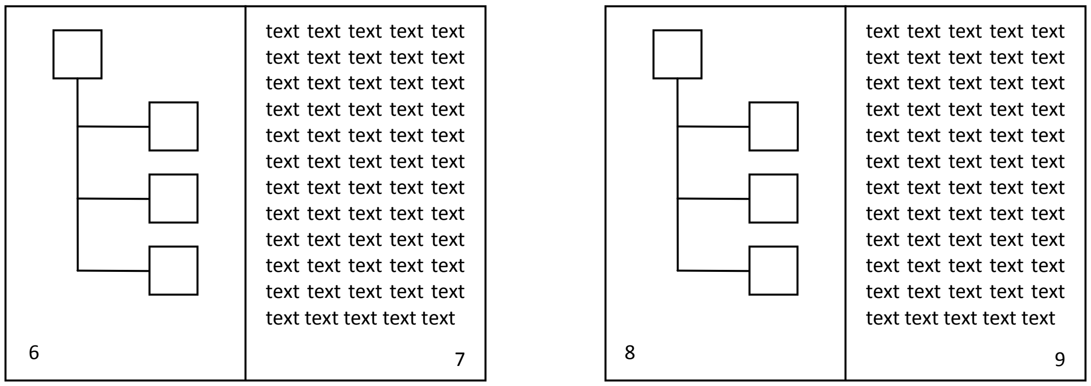

Figure 8: Complex diagram repeated on even pages.

The final suggested solution has been encountered for many years now in the manuals of any house-hold appliance we can think of: the folding page (Fig.9). It can be a page, let’s say double the size of the normal ones. When unfolded, it remains visible while the reader browses the next pages of the description (e.g. see: Papakitsos 2013c).

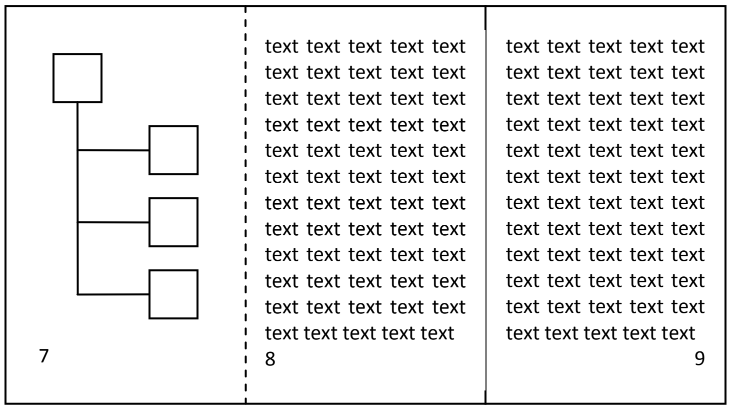

Figure 9: The folding page.

The second issue concerns the arrangement of the printed internal documentation. The indentation of code-lines denotes the structure of the program. In colorless printing, the inserted comments, here-and-there, break the continuity of the code. In the comments, code-lines are frequently referenced but not identified. If a code-line from another subroutine, which is called, is referenced, the identification is even harder. In Papakitsos (2013c, 2009), the printing arrangement of the documentation is in landscape mode (Fig.10). On the left part of the odd page, the code-lines are printed and enumerated like the old-fashion BASIC encoding. Subroutines and units are also identified, like the chapters of a book (e.g. B2#1.0, C1#2.0, etc. the first two characters standing for a unit and the last two for a subroutine). On the right part of the odd page, the commentary is printed. The lines are referenced as #8 for the 8th code-line of the left commented subroutine.

On the even page, above, whenever necessary, the entire commentary about the logic of the subroutine below can be read. In the example of Fig.10, the left part of the even page contains an explanatory flow-chart (or a table, if required) and the right part the text. This simultaneous multi-level description facilitates the literate programming approach (Knuth 1992, 1984), since all the relevant information about a subroutine (code; comments; logic; flow-chart/table) are simultaneously visible to the reader. For increased needs of description, the multi-level landscape printing and the folding page technique can be combined.

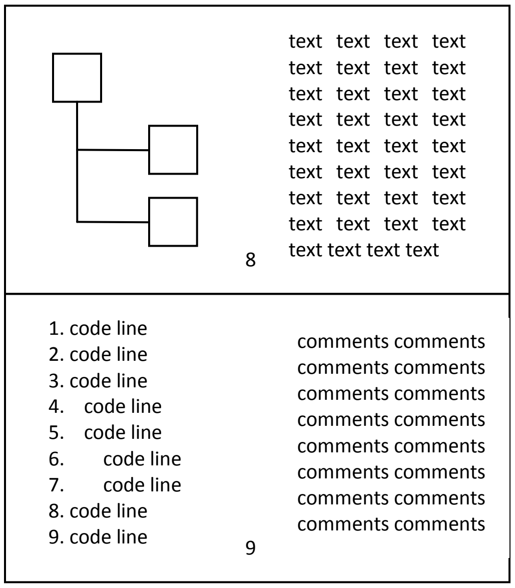

Figure 10: The simultaneous multi-level description.

5. Conclusions.

Cognitive Ergonomics aim at reducing the mental work-load of human tasks. A minimal mental work-load is related to fewer errors, which is a crucial factor in the quality assurance of software development. The quality of documentation has been identified as a very important tool for “errorless” programming. Various suggestions, concerning the quality of documentation, have been exemplified herein, in the direction of minimizing the mental work-load during the usage of software documentation. These suggestions are summarized below:

- The readability of documentation could be improved by the assistance of a *documentalist*, who would perform the proof-reading of text. Such a specialist could be a computational linguist.
- The use of a single systemic technique from analysis to design and down to flow-charts, including the structuring of documentation.
- The readability of diagrams could be improved by avoiding intersection of lines, achieving a uniform presentation of hierarchies and using space-efficient labeled shapes.
- The single representation of knowledge and data-relations through the use of tables, both for the E/R and the O-O modeling.
- The use of printing arrangements like the folding page, the multi-level landscape printing and the numbering of code-lines and modules.

The concept of simplicity can be implemented in various levels, i.e. to the number of tools, the selection of diagrammatic techniques and methods. This, along with the careful arrangement of information and of the specialized persons to perform the relevant task, could significantly improve the quality of software manuals.

References

Bailey, R.W. (1983). Human Errors in Computer Systems. Englewood Cliffs, NJ: Prentice-Hall.

Bench-Capon, T.J.M. (1990). *KNOWLEDGE REPRESENTATION: An Approach to Artificial Intelligence*. The APIC Series No 32, ISBN 0-12-086440-1. London: Academic Press.

Berge, A. (2010). *Technical Documentation Standard for Software development*. TECHNICAL DOCUMENTATION STANDARD V1.3, Specific-Group Software Solutions. Retrieved 9/22/2014 from: www.specific-group.com/files/Technical Documentation Standard for Software development.pdf.

Boehm, B.W. (1978). *Characteristics of Software Quality*. Elsevier North Holland.

Boughton, A. (2012). *Software Metrics*. University of Colorado Boulder, Department of Computer Science. Retrieved 9/22/2014 from: http://www.cs.colorado.edu/~kena/classes/5828/s12/ presentation-materials/boughtonalexandra.pdf.

Cañas, J.J., Velichkovsky, B.B., & Velichkovsky, B.M. (2011). *Human Factors and Ergonomics*. Retrieved 9/23/2014 from: http://tu-dresden.de/die_tu_dresden/fakultaeten/fakultaet_mathematik_und_ naturwissenschaften/fachrichtung_psychologie/i3/applied-cognition/lehre/vorlesungen/Human%20 Factors%20and%20Ergonomics.pdf.

Chen, P.P.-S. (1976). *The Entity-Relationship Model – Toward a Unified View of Data*. ACM TODS 1, No 1. Republished in Stonebraker M. (1988), *Readings in Database Systems*, San Mateo CA: Morgan Kaufmann.

CMEM (1974). *Colored-Methodical Encyclopedia of Mathematics: Theory of Algorithms* [in Greek: Translation of *KLEINE ENZYKLOPÄDIE MATHEMATIK* by VEB VERLAG ENZYKLOPÄDIE LEIPZIG, 1971]. 5th Vol., pp. 327-330. Athens, Greece: Pagoulatos Publications.

Croll, P.R. (2000). *An Overview of IEEE Software Engineering Standards and Knowledge Products*. ASQ Section 509 SSIG Meeting, 8 November 2000. Retrieved 9/22/2014 from: www.asq509.org/ht/a/ GetDocumentAction/i/490/An Overview of IEEE Software Engineering Standards and Knowledge Products.pdf.

Date, C.J. (1990). *An Introduction to Database Systems*. Vol. I, 5th Edition, ISBN 0-201-52878-9. Addison Wesley.

Delaney, D., & Brown S. (2002). *DOCUMENT TEMPLATES FOR STUDENT PROJECTS IN SOFTWARE ENGINEERING*. Technical Report: NUIM-CS-TR2002-05. Department of Computer Science, National University of Ireland, Maynooth.

Garidis, P.K., & Deligiannakis, E.N. (1993). *Dictionary of Computing* [in Greek]. 6th Edition, Athens: Diavlos Publications.

Gialelis, K.N., Dimitriadis, D.P., Kalergis, C.S., Kastania, A.N., Katopodis, I.K., Koulas, P.R., Oikonomou, T.G. (1999). *Software Applications* [in Greek]. Athens, Greece: Hellenic Ministry of Education & Religious Affairs – Pedagogical Institute.

Halstead, M. (1977). *Elements of Software Science*. North Holland.

JIS (1994). *Ergonomic principles related to mental work-load - General terms and definitions*. JAPANESE INDUSTRIAL STANDARD Z 8502-1994 (ISO 10075:1991), Japanese Standards Association.

Knuth, D.E. (1984). *Literate Programming*. The Computer Journal 27 (2): 97–111. British Computer Society, doi:10.1093/comjnl/27.2.97.

Knuth, D.E. (1992). *Literate Programming*. ISBN 978-0-937073-80-3. California: Stanford University Center for the Study of Language and Information.

Kytagias, D., & Psaromiligkos, I. (2004). *Visual Basic: from the theory … to the laboratory* [in Greek]. ISBN 960-8271-09-6. Athens, Greece: Diros Publications.

Leveson, N. (2005). *Software Metrics*. MIT OpenCourseWare: Massachusetts Institute of Technology. Retrieved 9/22/2014 from: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-355j-software-engineering-concepts-fall-2005/lecture-notes/cnotes7.pdf. License: Creative Commons BY-NC-SA.

McCabe, T. (1976). *A Software Complexity Measure*. IEEE Trans. Software Engineering, Vol. 2, December 1976, pp. 308-320.

Norman, D.A. (1986). *Cognitive engineering*. In D.A. Norman and S.W. Draper (Eds.), User centered system design. Hillsdale, NJ: Lawrence Erlbaum Associates.

Papakitsos, E.C. (2013a). *Linguistic Software Engineering: I. Preparation* [in Greek]. Educational Manual, ISBN 978-960-93-5636-7. Athens, Greece: National Library of Greece.

Papakitsos, E. (2013b). *The Systemic Modeling via Military Practice at the Service of any Operational Planning*. International Journal of Academic Research in Business and Social Science (ISSN: 2222-6990), September 2013, Vol.3, No.9, pp.176-190. DOI: 10.6007/IJARBSS/v3-i9/200. Human Resource Management Academic Research Society (www.hrmars.com).

Papakitsos, E.C. (2013c). *Mini Translator: Software of bidirectional machine translation between the artificial languages of Toki Pona and Minimal Extent Free Greek* [in Greek]. Documentation of software in Visual C# computer programming language, ISBN 978-960-93-5454-7. Athens, Greece: National Library of Greece.

Papakitsos, E.C. (2009). *Scansion of Hexameter by Object-Oriented Programming* [in Greek]. Documentation of software in Visual Basic 6.0 computer programming language. PDSA 567: E.K. Thessalou, TRN 062079143, Athens, Greece.

Pressman, R. (2005). *SOFTWARE ENGINEERING: A Practitioner's Approach*. 6th Edition, ISBN 9780072853186. McGraw-Hill.

Pressman, R. (1997). *SOFTWARE ENGINEERING: A Practitioner's Approach*. 4th Edition, ISBN 9780070521827. McGraw-Hill.

Pressman, R. (1987). *SOFTWARE ENGINEERING: A Practitioner's Approach*. 2nd Edition, ISBN 0-07-100232-4. McGraw-Hill.

Rasmussen, J. (1983). *Skills, rules, knowledge: signals, signs and symbols and other distinctions in human performance models*. IEEE Transactions: Systems, Man and Cybernetics, 13, 257-267.

SDG (2001). *Software Documentation Standards*. University of Western Cape, Department of Computer Science. Retrieved 9/22/2014 from: http://www.cs.uwc.ac.za/lecturers_webpages/avahed/www/software_documentation_guideline.htm.

Sommerville, I. (1992). *Software Engineering.* 4th Edition, ISBN 0-201-56529-3. Addison Wesley.

Sommerville, I. (1989). *Software Engineering.* 3rd Edition, ISBN 0-201-17568-1. Addison Wesley.

Young, M. S. & Stanton, N. A. (2002*). It's all relative: defining mental workload in the light of Annett's paper*. Ergonomics, 45(14), 1018-1020.