FACING INFORMATICS VIA THREE LEVEL COMPLEXITY VIEWS

by F. SAEZ VACAS (Spain),

Professor of Computer Science and Cybernetics,
Polytechnical University of Madrid.

FIRST LEVEL COMPLEXITY VIEW

In informatics there is one kind of complexity that is perceived by everyone. It is the complexity of a concrete, isolated object, normally situated completely within one of the branches universally recognized by the scientific and technical community. Examples of this are the complexity of integrated electronic circuits, the complexity of algorithms and the complexity of software. The first complexity deals with the number of circuit components, the second with computation time and the third with the number of necessary mental discriminations. In order to illustrate my point, I will take up the last complexity, which, moreover, is the least well-known.

The complexity of software is a specific subject which has received considerable attention due to the economic impact software has had on the total cost of computer use. In 1977, computer programming costs [9] in the U.S.A. fluctuated around 100,000 million dollars, an amount greater than 3 % of the GNP. Between 40 and 70 percent was consumed in program maintenance [6] .

Actually, it is excessive to talk about the complexity of software. We would be more exact if we were to talk about the complexity of a program since, with some unimportant exceptions, only metrics dealing with the degree of difficulty involved in understanding and working with a program, considered separately from other programs, have been developed.

Among the approaches taken in measuring program complexity characteristics, here I will cite Halstead's Software Science and McCabe's cyclomatic number as a couple of examples.

Halstead, inspired by thermodynamics, proposes formal expressions to be computed with some parameters, whose value is measured by enumera-

ting the following symbols of a specific program : a) unique operators; b) unique operands; c) total of operators; d) total of operands (see Tables 1 and 2).

Table 1 presents a definition of the parameters and Halstead's complete metrics. In Figure 1 and in Table 2 we have a practical application of a specific implementation of a bubble sort program. The numerical results of this program are completed in the right-hand column in Table 1.

Unique operators	n_1	8
Unique operands	n_2	5
Total operators	N_1	30
Total operands	N_2	18
Vocabulary size	$n=n_1+n_2$	13
Program length	$N= N_1+N_2$	48
Calculated length	$\hat{N}=n_1 lg_2 n_1+n_2 lg_2 n_2$	36
Potential vocabulary	$n^* =2+n_2^*$	4
Program volume	$V=N lg_2 n$	182
Potential volume	$V^* =n^* lg_2 n^*$	8
Program level	$L=V^* /V$	0.043
Calculated level	$\hat{L}=2/n_1 \; n_2/N_2$	0.069
Intelligence content	$I=\hat{L}.V$	12.6
Language level	$\lambda =L.v^*$	0.35
Programming effort	$E=V/L$	2640

Table 1. - Halstead's Metrics applied to the bubble sort program presented in Figure 1 [5, 6] .

Measurements such as intelligence content and programming effort positively correlate well with total programming and debugging time and with the effort required to comprehend an implementation.

McCabe's metrics [10] is the first of a series of topologic methods which measures on the program's flow graph how difficult program testing will be. The cyclomatic number V(G) on Graph G with n nodes, e edges and p connected components is

$$V(G) = e - n + 2 \times p$$

which, on a strongly connected G Graph, is the maximum number of linearly independent circuits. This number measures the complexity of a program,

if we accept that this complexity only depends on the program's decision structure.

```
(a)  BEGIN
         DO I = 2 TO N;
             DO J = 1 TO I;
                 IF X(I) < X(J) THEN DO:
                                         SAVE = X(I);
                                         X(I) = X(J);
                                         X(J) = SAVE;
                                     END;
             END;
         END;
     END;

(b)  CALL SORT (X,N)
```

Fig. 1. - Actual (a) and potential (b) implementations of a bubble sort program [6] .

Uniq. Operators	Total Operators	Uniq. Operands	Total Operands
Actual :			
Begin...end	1	I	5
;	11	N	1
Do...end	3	J	4
=	5	X	6
	1	Save	2
To	2		
If...Then	1		
( )	6		
$n_1 = 8$	$N_1 = 30$	$n_2 = 5$	$N_2 = 18$
Potential :			
Call	1	X	1
Sort(...)	1	N	1
$n_1^* = 2$	$N_1^* = 2$	$n_2^* = 2$	$N_2^* = 2$

Table 2. - Halstead's Metrics applied to the bubble sort program presented in Figure 1. [6] .

All programs can be turned into associated directed graphs and, therefore, classified by their cyclomatic number. The greater the value of the cyclomatic number, the more complex the program. The program in Figure 2 has a cyclomatic number of 4.

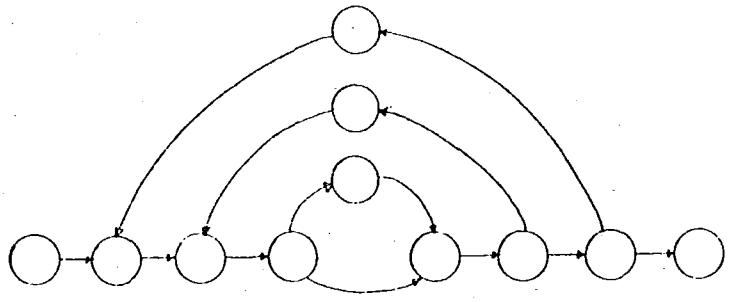

Fig. 2. - Directed graph associated with a program's decision structure.

Both approaches have been combined and improved in new approaches. In my opinion, Halstead's theory, which explicitly includes the descriptive power of language (that is, the observer's language) in order to reduce complexity, is superior. This is not important here since, with all the differences we may find between them, they belong to the same category of complexity studies, where the systemic approach is nowhere to be found.

SECOND LEVEL COMPLEXITY VIEW

Information technology, in general, and informatics, in particular, are never isolated objects, but rather a group of interconnected elements.

Examples : An operating system is a set of programs that interact over a period of time. A computer is a set of functionally different machines. An information system is the result of the interaction between a set of hardware, a set of software and a set of individuals.

In all of these cases, a strong sensation of complexity arises; but is handled in a confused manner. Perhaps this is the result of the meeting between different specialists. In other words, the universe of discourses is divided into specialities.

In my opinion, this is a complexity level (the second level) that will clearly emerge if the system notion is consistently applied. Unfortunately, this notion is rarely known in depth by information technology specialists (you need only take a look at the bibliographic references that these specia-

lists use to see that this is true). As a result, the second complexity level view has yet to be formulated. In other words, I believe that in spite of the creation and dissemination of techniques such as modularization, abstraction, information hiding, hierarchy, virtual machine, abstract data types and others, the unconnection and asystematism of these efforts demonstrate that a complexity methodology has hardly opened up its way into the scientific informatics community.

For our part, we have tried to make a contribution to the formulation process of this level which, in light of the present state of informatics, we consider necessary to the design, construction, understanding and use of this very complex technology (high technology means complex technology).

We have developed [8] a complex systems observation model, whose application to the understanding of informatics is quite promising. It consists in seeing (designing, constructing) a complex system as a multilevel, quasidecomposable system. Here we will summarize this approach as an example of what we consider to be a second level complexity view.

By taking the complexity of a system to be the minimum amount of information necessary to characterize it (definition inspired by [2, 3] , it is possible to study complexity by means of a set of interrelated factors, all of which are sources of complexity. They are the following : a) the parts or components of the system; b) their interactions; c) the environment on which some of their interactions depend; d) the observer.

In principle, it seems that complexity depends heavily on the observer. Cybernetics and Systems Theory tell us that it is the observer who defines the quantities in the system and its level of resolution [1, 7] ; for this reason, the aforementioned factors a), b) and c) are subject to this definition. This is true. But it is also true that in the technology of artificial systems it is more common to find that the observer of the system is not the system's designer but rather the user. In other words, in a certain way the system is turned around : the minimum complexity of which the observer should be capable of handling is imposed up on him by the complexity involved in the set of factors a), b) and c) in the system.

In short, in the field of artificial systems there are two important classes of observers, which we will schematically call the designers and the users. The designers face complexity in their designs and make decisions accordingly. They also have in their hands the power to codify the system in order to control the complexity of its use. For either one of these two roles, the class of designers needs to know the complexity concept and the intellectual tools ad hoc.

Now that these important boundaries have been drawn, we can center ourselves on the structural factors : the system's components and interactions which determine structural complexity.

Figure 3 presents various kinds of structures with (n+1) components. A system will normally be a hybrid of these different classes of structures.

We already know that a system is more and at the same time less than the sum of its parts. As complexity grows, new properties emerge and the properties of the very components are repressed. What is certain is that regardless of the kind of observer involved, an increase in structural complexity makes systems analysis or synthesis more difficult for him.

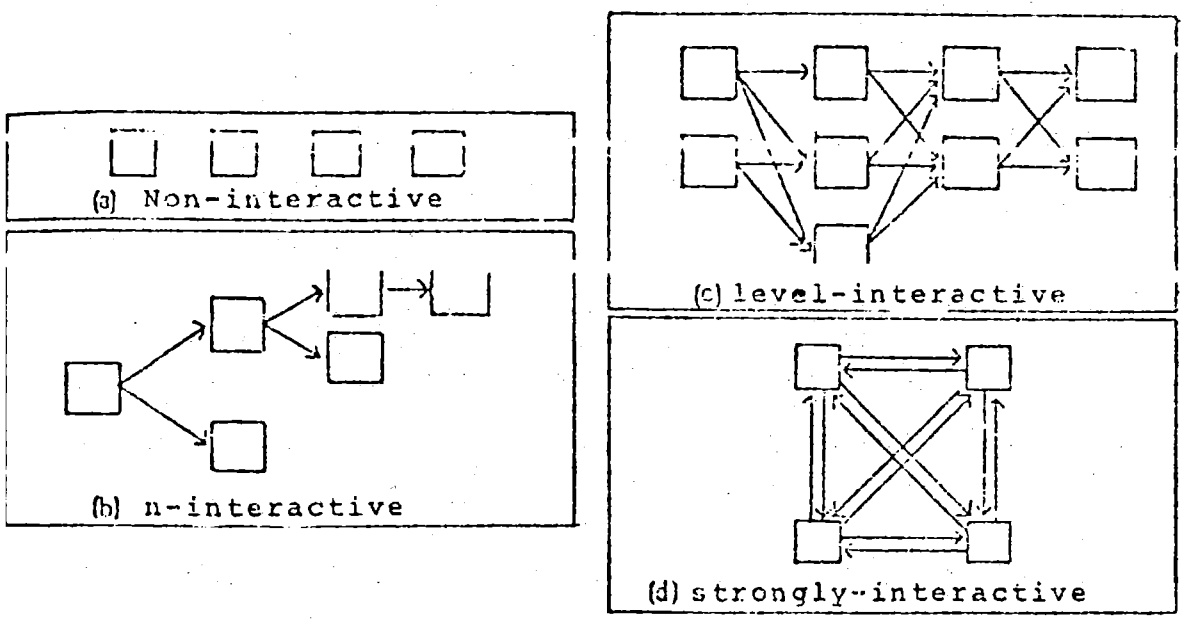

Fig. 3. - Kind of structures with (n+1) components.

We [8] propose that the observer-designer of complex systems use an approach that is both multi-level and quasidecomposable. The first approach is achieved by organizing the structure in a level-interactive fashion with strong interactive relationships within the levels. Thus, we have two kinds of interactions : bidirectional interactions in located areas (certain levels) and other unidirectional interactions between levels, which are simpler because they are based on the static master-slave distinction.

The second and complementary principle consists in trying to measure the strength of the interactions, so that priority is given to those interactions that exceed a specific threshold of intensity. In short, this principle involves adjusting the lense in order to bring the view of the structure closer to a non-interactive situation, and applying this lense to as many components as possible. This extremely important mechanism was pointed out in 1962 by Simon in his classic essay on The Architecture of Complexity [12] and used by Conant [4] , among others.

Complexity is unavoidable. It is intrinsically tied to the technological progress of society. The problem lies in knowing and accepting it so that its growth can be controled, thus making its morphology codify a relational complexity (the complexity perceived by the observer in his relationship with the object or system) that is tailored to the intellectual complexity of the observer with respect ot this object.

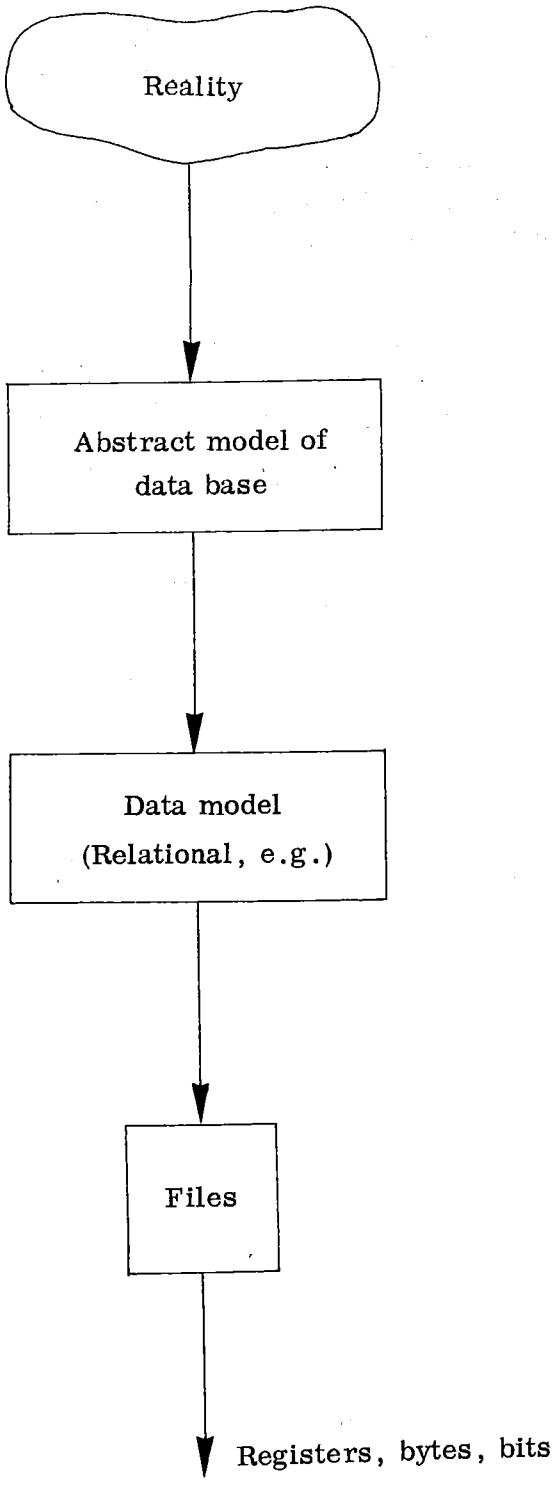

Fig. 4. - Chain of abstractions or levels of a data base.

As I said earlier, a methodology for complexity has not been designed and disseminated in informatics, and I consider this absence negative. Nevertheless, we can see that through intuition and by making approximations over time, solutions that are very similar to the multi-level and quasidecomposable approach proposed in [8] have been reached in the different areas of informatics. To this effect, under my direction, D. Lampaya has written a report, which has yet to be published and which analyzes the evolution of solutions in the specialized fields of Operating Systems, Data Bases, Network Architectures, Design and Development of Information Systems Methodologies, Programming, etc.

By taking up just one of these fields, namely Data Bases, we should note how it tends towards a theoretical scheme of level hierarchization, with a dominant correspondence between levels that goes from greater to lesser abstraction and a strong interaction in the set of elements of each level. Indeed, within each level, in an abstract model, for example, objects (entities) group themselves together, forming classes (entity sets) by considering certain properties (attributes) and passing over others. This could be considered a quasidecomposibility application. The different classes interact among themselves (relationships) (Fig. 4).

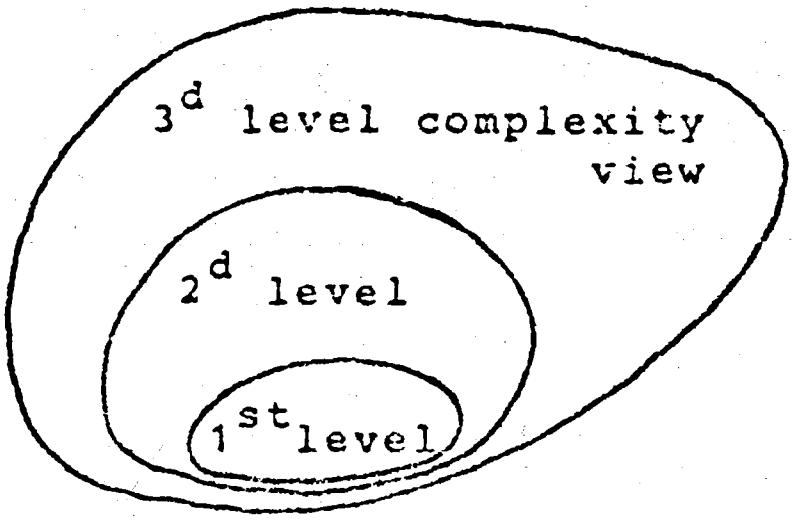

Fig. 5. - Fitting the three levels together.

The last point within the evolution of these systems would be the following : for an observer of the whole, which covers the abstract model down to the physical site where the bit is situated, the system should be multi-level, establishing the greatest degree of power on the upper level. We come much closer to achieving this goal with a relational data model than with a network data model, for example, for the simple reason that the first model allows us to use a greater amount and diversity of interactions on the upper level, in this sense resembling a typical master-slave relationship in a multi-level structure.

This would be, among others, an example of the application of the second level complexity view. Thus, the complexity of the task of designing technology is kept within acceptable limits; it enables and gives form to an increase in the real complexity of technology, with its retinue of interesting properties; and at the same time, it codifies its relational complexity for users. In the case of data bases, each class of user would perhaps face only one system level, which he would see as if it were an element of a non-interactive structure (the Data Base Management System would be designed to make the lower levels transparent).

THIRD LEVEL COMPLEXITY VIEW

When data processing technology artifacts are brought into society, they go from being systems to being elements of an antroposocial system. Sparks fly at all points of contact because the (supposedly) organized complexity of an artifact confronts the disorganized complexity of man in order to adapt themselves to each another, thus giving rise to a new being, in which the first complexity usually carries out the dominant "contra natura" role.

During this encounter, the disorganized aspect of complexity gains strength : that is disorder, uncertainty. It is inevitable. We should not forget that disorder clearly forms part of material even at the subatomic level. It is found everywhere and in the very heart of artificial systems (artifacts). Moreover, a living organization suffers from it and needs it to evolve.

Man makes and uses high technological systems which he impregnates with his own disorder, and this disorder adds itself to the disorder these systems already carry intrinsically. It is time to recognize that a paradigmatic order technology such as informatics implies, associates and even generates a strong dose of disorder. We saw in the section on the second level complexity view how the designer (actually, a set of designers) have not yet learned how to master complexity; nor do they know how to codify complexity for the different users. It is here that we find sources of disorder. To this we should add the multiplicity of users who, at a given moment, form part of an antropotechnic system and whose specific complexities differ from the complexities of the artifacts and interfere with each other. The resulting mismatch is probably, in quantitative terms, the most important cause behind disorder. The risk of disorder grows as the complexity of the system increases.

In summation, in an environment of widespread complexity we should expect to find a certain level of disorder, in one of its many forms, from the breakdown of an artifact to the misuse of its many possibilities. We could provide a great number of examples.

Let me cite the famous problem with base software that occurred years ago before the problem of processes coordination in an indeterminist environment was discovered. Another serious problem : the unreliability of software. Moreover, we should also mention the inconceivably poor use that is made of instruction sets of computer languages. Others : poorly designed configurations; hardware-software architectural imbalances; inconsistent systems designs; informatics crimes; physical problems with electronic (granted that they are becoming increasingly rarer) and mechanical components; incompatibilities of materials, codes, languages, protocols, etc.

In my opinion, we must accept the presence of disorder in our view of informatic technology. This shapes a third level complexity view, in which the system notion must be surpassed . I am of the understanding that systems theory and cybernetics, which should inspire the second level, are insufficient here. They represent order paradigms : match-ups, organization, information, stability, complementarity, command, negentropy, efficiency, hierarchy, logic, exactness...

We must introduce a more complete vision of complexity, one that encompasses that which is organized and that which is disorganized as inseparable aspects. To the order paradigms mentioned above we must at the same time add/contrast : Mismatch, disorganization, noise, instability, antagonism, entropy, perturbation, misuse, anarchy, intuition, fuzziness...

In other words, this third level view of complexity comes closest to the reality of a high technology environment. The driving gendre of thought is found in the reflections of authors such as Morin [11] .

CONCLUSIONS

The author sees three levels of complexity in informatics, with each level requiring one kind of view.

The first view refers to objects such as circuits, algorithms or programs. The kind of complexity formulated in these objects is recognized by all specialists.

When the informatic object is combined with other objects in order to establish systems, another kind of complexity arises which we could call systemic complexity. We do not believe that such complexity has yet to be tacked in a scientifically conscious manner. For this reason, the second level complexity view has yet to be well-defined.

Lastly, when the above-mentioned systems reach society, they give rise to antropotechnic systems. These systems require a broader view, one that accepts and confronts widespread complexity.

The formulation and dissemination of this hierarchy of levels will probably require a tremendous effort in the area of understanding and education. Moreover, they are essential in an evolutionary, high (complex) technology environment.

REFERENCES

[1] ASHBY, W.R., An introduction to Cybernetics, Chapman and Hall, London, 1956.

[2] CHAITIN, G.J., On the Difficulty of Computations, IEEE Trans. on Information Theory, Vol. IT-16, N° 1, pp. 5-9, Jan. 1970.

[3] CHAITIN, G.J., Information Theoretic Computational Complexity, IEEE Trans. on Information Theory, Vol. IT-20, n° 1, pp. 10-15, Jan. 1974.

[4] CONANT, R.C., Detecting Subsystems of a Complex System, IEEE Trans. Syst. Man. Cyb., SMC-2, pp. 550-553, Sep. 1972.

[5] HALSTEAD, M.H., Elements of Software Science, Elsevier North-Holland, Inc., N.Y., 1977.

[6] HARRISON, W. et al., Applying Software Complexity Metrics to Program Maintenance, Computer IEEE, pp. 65-79, Sept. 1982.

[7] KLIR, G.J., An Approach to General Systems Theory, Van Nostrand Reinhold, New York, 1969.

[8] LAMPAYA, D. et SAEZ VACAS, F., Concepcion multinivélica y cuasidescomponible de sistemas complejos. Aplicacion a la Informática, 5° Congreso de Informática y Automática, Madrid, Mayo 1982.

[9] LEHMAN, M.M., Programs, Life Cycles and Laws of Software Evolution, Proceedings of the IEEE, Vol. 68, n° 9, Sept. 1980.

[10] McCABE, T., A Complexity Measure, IEEE Trans. Software Eng., Vol. SE-2, pp. 308-320, Dec. 1976.

[11] MORIN, E., La Méthode I : La Nature de la Nature, Seuil, Paris, 1977.

[12] SIMON, H.A., The Sciences of the Artificial, The M.I.T. Press, Mass., pp. 99-103, 1970.