Jenaer Schriften zur Mathematik und Informatik

Eingang: 2000.02.22 **Nr. Math / Inf / 2000 / 02** **Als Manuskript gedruckt**

CHARACTERISTICS OF COMPUTING AND INFORMATICS

Jürgen F H Winkler

Friedrich-Schiller University, Institute of Computer Science
D-07740 Jena, Germany
http://psc.informatik.uni-jena.de

CHARACTERISTICS OF COMPUTING AND INFORMATICS

JÜRGEN F. H. WINKLER

Friedrich-Schiller University, Institute of Computer Science, D-07740 Jena, Germany
http://psc.informatik.uni-jena.de

After fifty years of very rapid evolution the new field of computing and informatics seems not to have found its proper place in the world of science. This paper presents some characteristics and distinctions of computing and informatics. The characteristics express the relation of computing to mathematics and other neighboring disciplines as e.g. electrical engineering. The last distinction is the similarity to the human brain and its accomplishments; this seems to be a unique characteristic of computing.

1 Introduction

The field of computing is a rather new discipline in the world of engineering and science [DM 97]. It is a "child" of the modern digital computer which has been invented in the thirties and forties of the 20th century. This new field has evolved quite rapidly and has therefore not even found its proper name. Traditionally, it is called "computer science" which does not completely reflect its current contents. Other names are "computer engineering" or "computing" [CDG 89], "information processing" (e.g. in IFIP) or "informatics / informatique" (Denning e.g. mentions "bioinformatics" [Den 97: 274]). A good name would also be "computational engineering" in analogy to "electrical engineering" or "mechanical engineering" [Lou 89, 95]. Sometimes it seems not completely clear what kind of field computing is [CDG 89: 9; Ber 96]. On the other hand, computing has developed into a field on which mankind already depends quite heavily. In these days the Y2K problem is one example for this dependency (see e.g.[Ber 99; NM 99]). The dependency of our society on computing becomes even more evident if we ask the question:

Characteristics of Computing

Finiteness
SW: Discreteness
Effectivity / Efficiency
High complexity in finite constructs
Taking Error Situations into Account

SW: Design = Product
SW: Versatility
SW: "Invisibility"
SW: No leakage and spurious effects

Similarities to the Human Brain

What would happen if suddenly
all programs in the world came to a stop?

In this paper we are neither concerned with the name for the (new) discipline of computing nor with its importance for mankind but with its basic characteristics and distinctions, i.e. in this paper we try to identify the basic characteristics of this new field and the relations of it to its neighboring disciplines.

This list of characteristics of computing / informatics has been compiled as part of the inaugural lecture of the author [Win 97], which was held at 1995.Jan.12 at Friedrich Schiller University in Jena. An abbreviated version has been presented at the IFIP World Congress 1998 [Win 98].

There are three groups of characteristics and distinctions: the first deals with relations between computing and mathematics, the second deals with relations between computing and other fields of engineering, and the last is a general characteristic of computing.

Some characteristics are labeled with "SW" because they seem to apply especially to programs or software. They may also apply to HW but in this paper they are discussed with respect to SW. This is no strong restriction because "most of the distinctions of computing are embodied in programming notations." [CDG 89: 11] and most computational engineers work in SW [GI 92]. The computer itself, i.e. the hardware, is typically not built by computational engineers but mostly by electrical and mechanical engineers.

2 Basic Properties of Computers

Since informatics as an engineering discipline is based on the computer it is useful to mention first the basic properties of this machine. They are listed in Table 1. The properties in Table 1 are those properties which distinguish computers from human beings when they are doing similar tasks as computers or from other machines, e.g. mechanical or electromechanical ones.

Basic Properties of Computers
High Speed
Reliability / Exactness
Endurance
Universality
Flexibility
Ability to communicate

Table 1

High Speed

Computers are e.g. able to add several millions of numbers with 10 decimal places in one second. This is very much faster than human being or earlier computational devices.

Reliability / Exactness

Despite the high speed mentioned above those additions (or other operations) are performed in a very reliable manner: the computer can perform those additions for hours without making any error. The probability of an error is much less than for a human being or for a mechanical calculator. Furthermore, the computer executes exactly those instructions which are in the program in its memory.

Endurance

In this fast and reliable manner the computer can work 24 hours a day and seven days a week.[1]

Universality

Computers are universal in two respects:

Universality of the bit string: any data or information can be represented as a bit string: numbers, texts, programs, pictures, films and sounds. An example for the latter is the music CD and an example for pictures are the digital cameras which have been developed in recent years.

Universality of the instruction set: the processors in the computers are built in such a way that they can perform a great variety of tasks, which are formulated as programs and are put into the memory of the computer. All tasks which can be described in a precise manner can be performed by a computer (if its finite resources are sufficient; see also sect. 3). All these programs can be constructed using a small set of instructions.

Flexibility

Computers are able to switch very fast between different tasks: adding numbers in one moment and playing chess in the second and formatting a document in the third moment. If all three programs are in its memory the computer can switch from one to the other in a fraction of a second. By loading a program for a certain task into its memory the computer “learns“ very quickly to perform this task; and if the program is unloaded the computer „unlearns“ also very quickly.

Ability to communicate

Computers do not only perform internal computations as suggested by the preceding paragraphs but can also exchange data with their environment via input and output devices. From a practical point of view we may distinguish between two kinds of such data exchange: (1) with the immediate environment via the own periphery as e.g. keyboard, mouse, sensors, CRT display, loudspeaker etc., (2) with remote computers as e.g. in computer networks. This last aspect has recently gained additional importance [Wal 99].

To sum up we may say that the computer is a very fast, reliable, tireless and obedient “computing and communication slave“.

[1] „Human fatigue robbed world chess champion Garry Kasparov of a win in the fourth game of his re-match against the supercomputer Deep Blue .. „I was tired and I couldn’t figure it out,“ a subdued Kasparov told ...“ [Reu 97a]

3 Characteristics of Computing and Informatics

F i n i t e n e s s[2]

"Now it is obvious that no *finite* machine can include infinity"
Charles Babbage 1864

All computers are finite machines. In that computer science is different from mathematics, which deals mainly with infinite constructs, especially for the last 130 years ([Mes 67; Rus 84: 783, 784], "Mathematics is the science of the infinite,..." says Hermann Weyl [DH 81: 108]). On the other hand, Donald Knuth published a paper with the title: "Mathematics and Computer Science: Coping with Finiteness." [Knu 76]. In a brochure of the department of computer science of the Technical University of Munich we read: "Genuine transfinite theories are irrelevant to computer science, only the (nonfinite) border of the realm of finite theories is directly useful."[3] [TUM 92: 21; transl. by JW]. In finite domains a lot of questions are effectively decidable; therefore we may say:

The computicists don't live in Cantor's Paradise[4]*, but rather in that of Leibniz*[5].

It seems that the computicists do not really appreciate the fact of finite machines because the resulting limitations are often not taken fully into account [KW 97]. One reason for this seems to be that things get more complicated and less elegant if the limitations of finite domains have to be taken into account. The theory of IEEE 754 floating point numbers is more complicated than that of the rational numbers. An even more simple example is the formula for the computation of the arithmetic mean of two (floating point) numbers. We often find the following statement for this algorithm [McC 93: 226; Mei 95: 88; Oli 93: 268; Sed 88: 562]:

```
   Average := (A + B)/2.0;     (1)
```

which reflects the typical corresponding mathematical formula [Cal 95: 799]. If the domain of Average, A and B is finite (e.g. -max .. max) then statement (1) does not work for all possible combinations of A and B. If the sum is greater than max we get no result (e.g. in Ada) or a wrong result (e.g. in Java or C).

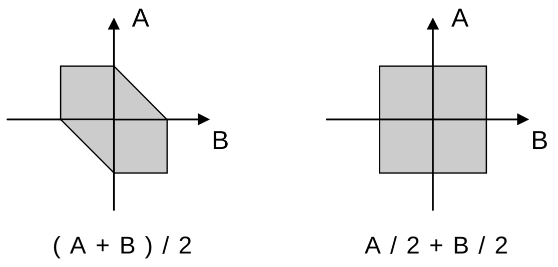

Fig. 1. Domain of Applicability of two Expressions for the Computation of the Average of A and B

An alternative for computing the average is:

```
                  Average := A/2.0 + B/2.0;                      (2)
```

[2] With respect to computations we may distinguish between two kinds of finiteness: *spatial* finiteness and *temporal* finiteness. In this section we address spatial finiteness. Spatial finiteness has an influence on temporal finiteness insofar as under spatial finiteness we are not able to distinguish an infinite number of points in time even if a program runs in an endless loop.

[3] "Echt transfinite Theorien sind für die Informatik belanglos, lediglich der (nicht-finitäre) Rand des Reichs finitärer Theorien ist direkt nützlich."

[4] This refers to the world of infinite (transfinite) sets. David Hilbert wrote in "Über das Unendliche" ("On the infinite"): "Aus dem Paradies, das Cantor uns geschaffen, soll uns niemand vertreiben können." ("Nobody should be able to expel us from the paradise which Cantor created for us."; transl. by JW) [Hil 26: 170; Mes 67: 87]

[5] This refers to the "Characteristica Universalis" of Gottfried W. Leibniz, a "kind of generalized mathematics, by means of which thinking could be replaced by calculation. 'If we had it,' he says, 'we should be able to reason in metaphysics and morals in much the same way as in geometry and analysis.' 'If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates, and to say to each other (with a friend as witness, if they liked): Let us calculate.'" [Rus 84: 572, 573] "Quo facto, quando orientur controversiae, non magis disputatione opus erit inter duos philosophos, quam inter duos Computistas. Sufficiet enim calamos in manus sumere sedereque ad abacos, et sibi mutuo (accito si placet amico) dicere: c a l c u l e m u s." [Ger 1890: 200] [Ger 1890: 26, 65, 125, 200; Krä 88: 104]

Fig. 1 shows the difference between the two statements (1) and (2) with respect to the domain of applicability [Win 90a].

Things become even worse when we compute the average of more than 2 values [AU 92: 114; BDM 91: 88] using the naive approach in statement (1).

Another infamous example is the factorial function [NW 89]. Most published programs for the factorial deliver for almost all possible input values either no result or a wrong result (e.g.: [Tuc 97: 2007]).

Discreteness

Spatial finiteness implies that we have only finite sets of values in the computer or in a program. Finite sets contain always discrete values. From the viewpoint of software the level of bits is the level of greatest detail which is taken into account. A bit is conceived as indivisible, and this led Heinz Zemanek, the leading Austrian computer pioneer, to the remark “Apart from the bit there is no atom.“ [Zem 92: 274]. The world of the computational engineer is therefore a discrete one. In this world the continuum of the real numbers does not exist. Whether it exists at all is for a computational engineer a more philosophical question; modern physics sometimes also sees the world rather as a discrete one [Mes 67: 218]. Norbert Wiener, the father of cybernetics, is told to have said: “Shannon is just crazy, he thinks in yes-no decisions.“[6] [Kit 94; transl. by JW]

Effectivity / Efficiency

Like engineers of other branches of engineering computational engineers develop things to be used by other people. In case of the computational engineers the products are mostly programs (SW). The result of the activity of the computational engineer has therefore to be a thing which can be used *effectively*; a pure existence proof (as e.g. in mathematics) is not sufficient. Additionally, *efficiency* is often of great importance. If a program needs 36 hours to compute a 24 hour weather forecast, then this program is not well suited for use in the daily weather report. Such aspects have also been mentioned by Heinz Zemanek [Zem 91: 178].

High Complexity in Finite Constructs

Programs belong to the most complex objects men have created [Bro 87: 11]. The complexity of a construct depends on three things: the number of components, it consists of, the different kinds of components, and the number of relations and interactions between those components. With respect to the number of components the constructs of computational engineering are similar to those of other branches of engineering. A program may consist of several million instructions and a microprocessor of several million transistors (e.g. Pentium or Pentium Pro). The Great Wall in China (ca. 6,400 km long) also contains a large number of bricks, even taking into account that it does not entirely consist of masonry. The main difference between the constructs of computational engineering and other branches of engineering is in the kinds of components and the relations and possible interactions between components. Fig. 2 shows an example from civil engineering. We see that only adjacent components interact with each other. Since the components have a certain minimal size only a very limited number of components interact at one junction point. In a pure framework the beams are only stressed in two ways: push (pressure) or pull (tension). This leads then to a very limited number of different interactions between the adjacent beams of a framework.

[6] “Shannon ist einfach verrückt, er denkt in Ja-nein-Entscheidungen.“

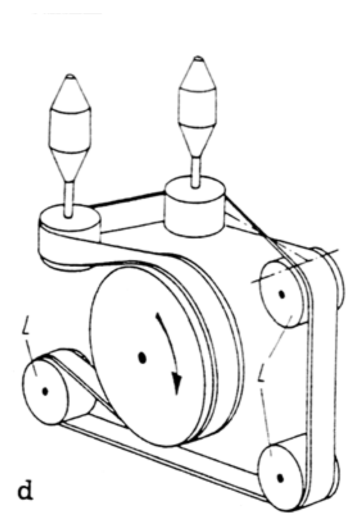

Fig.3. Belt Drive in Mechanical Engineering

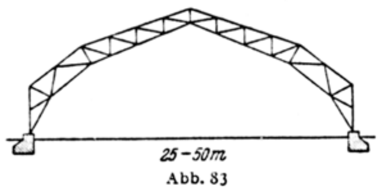

Fig. 2. Framework in Civil Engineering

In mechanical engineering we may distinguish two kinds of interactions: mechanical and hydraulic. Due to physical limitations mechanical interactions are mostly limited to neighboring components as e.g. gearwheels. By using chains or belts somewhat more flexible interactions are possible. Fig. 3 shows an example. The most flexible interaction in mechanical engineering is by means of hydraulic hoses because they can establish an interaction between somewhat remote elements. But those hoses still need some room and therefore the number of such interactions is still rather limited.

More remote interactions are possible in electrical engineering because the wires used for this purpose usually need less room than the hydraulic hoses, and furthermore electric wires can also be arranged in more flexible ways. But since wires still need some room the number of such interactions is limited. Fig. 4 shows an example of a classical electric circuit in which the collector of the transistor T5 is connected with the diode D1 and the base of the transistor T1. We may observe that Fig. 4 shows the logical design of the circuit and not its physical layout.

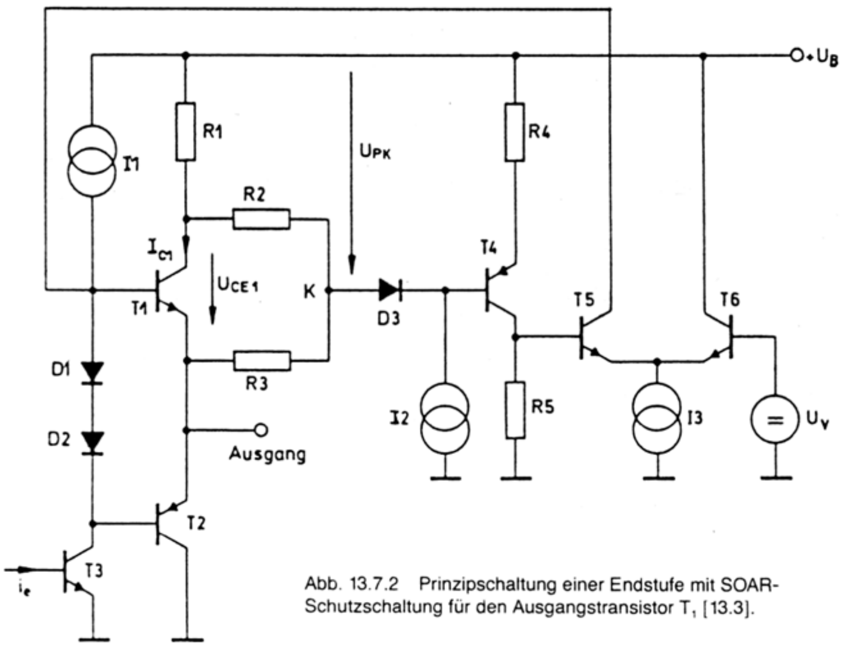
+UB
R1
T1
R2
UPK
R4
IC1
T4
UCE1
K
T1
D3
T5
T6
D1
R3
I2
R5
I3
UV
D2
Ausgang
T2
T3
ie
Abb. 13.7.2 Prinzipschaltung einer Endstufe mit SOAR-Schutzschaltung für den Ausgangstransistor T1 [13.3].

Fig.4. Electrical Circuit

In a program it is very much easier to establish an interaction between remote elements. As an example we may look at a program consisting of several million lines of code (LOC). This program would fill several dozen Encyclopædia Britannica volumes as depicted in Fig. 5. It is very easy to establish in Vol. 55 on page 27 a reference to an entity which is defined in Vol. 3 on page 396. We just write the name or identifier of this entity. In this manner we may establish thousands or tens of thousands of interactions in a program. Still the number of such references is limited because these references also need some room (in the source program). But there are two differences to other engineering designs

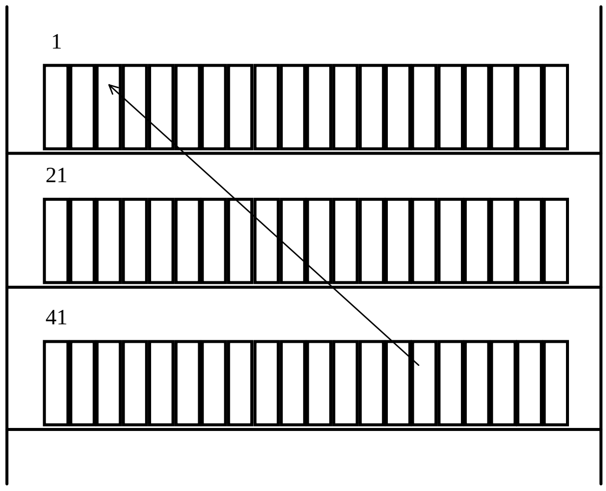

Fig. 5. Millions of LOC in 60 Volumes

a) references can span arbitrary distances (extreme cases are remote procedure call and hyperlinks in the WWW)

b) there are more different kinds of interactions established by the references: use of a type, assignment to a variable, reading a variable, calling a procedure, aborting a task, reading a value from a sensor, etc. This increases also the complexity of the program: “The real measure of complexity is not the number of elements but the number of (nonuniform) interactions.“ [Fra 97: 52].On the other hand, the (few) interactions between the components of the Chinese Wall are rather uniform.

Often such interactions in a program are not explicitly expressed but only implicitly, if e.g. both lines refer to the same variable; i.e. these interactions are barely visible which makes the understanding of such constructs even more difficult: “If you try to inspect all but the smallest program, the complexity overwhelms you.“ [Par 94: 272]. We may speak of the “invisibility“ of SW (see below). Things become even more complicated if the program contains parts which are executed concurrently.

Taking Error Situations into Account

Whereas mathematicians usually can think up their constructs in an undisturbed world of pure thinking in such a way that they look *elegant* and *beautiful*[7], the computational engineers have also to take aspects of the real world into account. One prominent example for this is the reaction to errors, e.g. in the form of erroneous operation. In mathematics division for numbers is defined in such a way that division by zero is not allowed. On the other hand, a pocket calculator must be designed in such a way that such input is also processed. It is not possible to exclude such an erroneous input “by definition“; it may just be the result of some inattention of the user. It may well be the case that such error handling accounts for 50 % of a program.

SW: Design = Product

Programs are the result of the design activity of (SW) engineers. In comparison to other engineering designs programs have the following special property: they can be immediately brought into use. I.e. the realization threshold (= production cost) is essentially zero. This is in sharp contrast e.g. to the design of an architect which in order to be brought into use has to be realized at the construction site. The cost of construction is usually much higher than that of the design. The situation is similar for the designs of mechanical engineers (e.g. car) or electrical engineers (e.g. power plant). Even if the design of the SW engineer is a mass product the cost of replication is rather low[8]. As a consequence of the low realization threshold of programs modifications, corrections or improvements can often be carried out quite easily. This is actually the reason for calling programs “soft“ware. A high realization threshold on the other hand implies that before implementation or realization a design is first checked and scrutinized with respect to possible errors or design flaws. Because of the very low realization threshold for programs such checks

[7] “If a mathematical theory has not yet proved useful, then it should at least be beautiful.“ [Pie 87: 311]

[8] “Although duplication of software might be considered to be manufacturing, it is usually a relatively trivial process.“ [Lev 95: 22]

and inspections are often not done (edit-compile-run cycle). As a consequence SW products quite often contain errors [Gib 94].

The low realization threshold of programs is a sort of temptation for the developer and may lead to the fact that designs are brought into use, which are not checked very carefully ("improvements" can always be incorporated quite easily; "bug fixing"). An indication for this temptation is the fact that a lot of SW companies use license agreements which contain disclaimers of warranty. This bad habit has only recently been stopped in the EU when programs were put on equal footing with other technical products with respect to product liability [EWG 92]. It seems to me that the property "design = product" is one of the reasons for what is sometimes called the SW-crisis[9] [Gib 94]. Other reasons are: (1) a new discipline always has its pains of growing (the first generations of people are from other disciplines [BF 97; GI 92], knowledge and methods have to be developed first); (2) the field of computing evolved especially fast, in a truly rampaging speed. With respect to "design = product" the SW engineers seem to have a more difficult job than their colleagues from other engineering disciplines.

This situation may thus be characterized by:

Glamour and misery of software is it's ease of making

Versatility of SW

If the function of a device or system is implemented in SW rather arbitrary functions are possible. Almost any computable function can be realized[10]. If the function is very irregular and has to be stored as a table storage requirements could result in some limitations. With earlier techniques as e.g. electric circuits only functions with certain characteristics could be readily implemented. One example are controllers which came in few kinds. Typical controllers are PD, PI and PID. With mechanics the functions were even more limited as we see at the flyball governor of James Watt.

This versatility of SW is based on the universality of the bit string and the universality of the instruction set.

"Invisibility" of SW

```
1110010101110000100
0101111011000111001
0101110000001110111
1011000011111100000
0110100100001001000
0100100001001000010
0010111100001111100
11110000000
```

```
11100101 01110000
10001011 11011000
11100101 01110000
00111011 11011000
01111110 00000110
10010000 10010000
10010000 10010000
10001011 11000011
11100111 10000000
```

Fig. 6. Binary Program

SW is a component of many systems and devices. As such a component it is contained in some storage device (chip, magnetic disc, CD-ROM). For separate programs, as we buy them for our computers, the situation is quite the same. If we look at such a storage device we don't see the programs contained in it and on the other hand, those storage devices look always the same, even if they contain different programs. On this level SW is invisible. But we can often read the program from the storage which contains it. We usually obtain a sequence of bits which represent a machine program for a certain processor. Fig. 6 shows a binary program in two forms: (a) as a sequence of bits and (b) as a

[9] In the company of Konrad Zuse SW led to problems, too: "Furthermore, the software problem is becoming a steadily increasing cost factor." [Cza 79: 17; transl. by JW] ("Zudem wird das "Software-Problem" zu einem immer größeren Kostenfaktor." [Cza 79: 17].)

[10] The reader should bear in mind that in real technical devices the domains of the input and output values are finite and often rather limited.

sequence of words, where each word corresponds to a machine instruction. On this level it is quite difficult to see what the program does and even more difficult, maybe even practically impossible, to observe the interactions mentioned before. The arrow in Fig. 5 is in the real program NOT present. The program in Fig. 6 contains interactions, but they are implicitly contained in the sequence of bits. As shown in [Win 97] the program computes the minimum of two numbers. People may mention decompilation techniques to overcome this unillustrativeness, but decompilation is usually not allowed [End 92].

During the development process of the SW the situation is somewhat better because the programs are then represented by graphical diagrams [Win 90b] and mostly in a high level language, which is usually more readable than the binary code in Fig. 6. On the other hand, Brooks thinks that this does not help and that SW is “inherently unvisualizable“ [Bro 87: 12].

This “invisibility“ of SW makes the job of the SW engineer more difficult than that of his colleagues of other engineering disciplines. It begins in education: handbooks for mechanical engineers or for architects usually document existing designs which have proven in products (see e.g. [BK 90: P83, R33] which shows the designs of whole engines and turbines). On the other hand, books on SW engineering usually only contain small examples designed by the authors of the books (see e.g. [Den 92; McD 91; Som 92]). Pomberger writes: “such designs (of SW systems, JW) are not documented in books.“ [Pom 93: 290]. It is therefore quite difficult for both teachers and students in computing and software engineering to learn how to design somewhat larger programs. For the teachers it is usually also not possible to learn about the design and structure of real SW products. Because of the “invisibility“ the companies have no motivation at all to publish those things; on the contrary, they will have a competitive advantage if they keep them secret[11]. This is very different for cars or other products of mechanical engineering, and architects are proud to publish their designs in books.

SW: No wear and tear or spurious effects

As already mentioned above the world of SW is a discrete one: a bit is either 0 or 1; there are no leakage or spurious effects [CM 80: 11, 12], fluctuation of properties of material, or effects of wear and tear[12]. Software doesn’t suffer form aging or transient failures. In this aspect the life of the SW engineers is easier than that of their colleagues in neighboring disciplines.[13]

Similarity to the human brain

Of all the artifacts made by men computing systems, especially those from the field of artificial intelligence, approximate the human brain most closely. Just the term “artificial intelligence“ may sometimes lead to heated debates. The main reason for this is that intelligence is very often seen as that aspect of man which distinguishes man from the rest of nature.

With respect to the question whether there is or can be any sort of artificial or machine intelligence there are mainly two camps:

those who are saying YES and those who are saying NO.

[11] “It is proposed that computer programs are best protected by keeping the source program secret and leasing object programs.“ [Fin 80: 135]

[12] “Since software is pure design, there is no need to worry about the random wearout failures found in physical devices ....“ [Lev 95: 27]

[13] The SW engineers may sometimes also be affected by such effects because the programs are stored using physical devices. If a high availability is required it may be necessary to store the (constant) program in a more permanent store (e.g. ROM) in order to have a high probability that the program really remains unchanged.

The YES-people point to the fact that computational systems are doing a lot of things which require more or less intelligence: adding numbers, differentiating functions, proving mathematical theorems, or playing chess. Douglas Hofstadter, who belongs to this camp, says: “WE HAVE COME to the point where we can develop one of the main theses of this book: that every aspect of thinking can be viewed as a high-level description of a system which, on a low level, is governed by simple, even formal, rules. The “system”, of course, is a brain ...“ [Hof 80: 559]. Klaus Haefner, a German computicist, belongs also to this camp [HW 92], and Konrad Zuse wrote already in 1937 in his diary: “Crucial idea 19. June 1937 - discovery that there are elementary operations, from which all operations of computation and of thinking can be built up. ... to solve all tasks of thinking which may be covered by mechanisms.“ [Zus 93: 41[14]; transl. by JW].

The NO-people claim that man cannot be completely explained by scientific laws, but that he contains something transcendental which is beyond science. If we examine this position critically we end up in a situation where the reasons are based on belief. To the camp of NO-people belong Joseph Weizenbaum [HW 92], who has worked at the MIT, Hubert Dreyfus [Dre 79] and others. Weizenbaum e.g. says: “Look, I think it utterly alarming that we have to discuss at all whether there is an essential difference between man and machine. The mere question is an indication of our technological frenzy. That’s no question, you know!“ [HW 92: 114[15]; transl. by JW]. This is from a debate between Weizenbaum and Haefner. The publisher gave the book the title: “Are computers the better men?“ [HW 92].

If we look at chess as one specific problem of AI we see that the NO-people sometimes have problems with their argumentation. About 20 years ago Dreyfus wrote: “In Chapter 1, I note that human beings avoid the counting out of large numbers of alternatives characteristic of a computer program by “zeroing in“ on the appropriate area in which to look for a move and I suggest that this ability is the result of having a sense of the developing game.“ [Dre 79: 30]. On the other hand, Konrad Zuse wrote already in 1938: “In fifty years the world chess champion will be defeated by a computer.“ [FAZ 94; Zus 93: 45]. It did not happen in 1988 but in 1997: “Garry Kasparov’s legendary resolve broke down Sunday in a defeat by the IBM supercomputer Deep Blue that created chess history -- the first time a program has triumphed over a reigning world champion in a classical chess match.“ [Reu 97b]

It seems that there is no end to such a debate between the YES-people and the NO-people. Each of us has to cope with this question by himself. It is part of a world view or a conception of the world: “In a way, this book is a statement of my religion. I hope that this will come through to my readers, ...“ [Hof 80: xxi].

A third position, which seems to be taken quite rarely, would let that question open since it refers to the development in the future. It is well known that it is difficult to predict the future: “the World Wide Web, which seemed to appear out of nowhere in 1992 ...“ [DM 97: xiv]. How should we know what evolution still has up it’s sleeve? Actually, evolution is often seen as a random process [Bre 77; Mon 82].

3 Conclusion

The characteristics and distinctions presented in this paper mostly characterize computing and informatics in relation to its neighboring fields and disciplines. We see clearly that computing differs from mathematics and also from other engineering fields. In its relation to mathematics it is similar to other engineering disciplines: it uses and applies results of mathematics. Computing may be somewhat nearer to mathematics because its basic mechanism is a mathematical one. On the other hand, it is very similar to the other engineering disciplines in that computational engineers also design and build practical and useful devices and systems. One difference to other engineering disciplines is that at least for the software “products“ the

[14] “Entscheidender Gedanke 19. Juni 1937 - Erkenntnis, daß es Elementaroperationen gibt, in die sich Rechen- und Denkoperationen auflösen lassen. ... sämtliche Denkaufgaben zu lösen, die von Mechanismen erfaßbar sind.“

[15] “Wissen Sie, ich finde es zutiefst erschreckend, daß wir uns überhaupt darüber unterhalten müssen, ob es einen prinzipiellen Unterschied zwischen Mensch und Maschine gibt. Schon diese Frage ist Ausdruck unseres Technologie-Wahnsinns. Das ist doch keine Frage !“

realization threshold .i.e. the production costs are negligible. This has been characterized by “design = product“. Based on these observations we may conclude that “computational engineering“ or “computing“ are be good candidates for the name of our discipline. We may observe that for this conclusion we do not need a definition of the term “information“.

Apart from this differential characterization it could also be useful to complement the given characteristics by some characteristics describing more directly what the basic mechanisms and phenomena of this new discipline are. As already mentioned in sect. 2 we have two basic things in the computer: bits and instructions. Instead of bits we may also speak of data or information or even knowledge; and instead of instructions we may speak of commands or statements or executable statements. As long as we speak of data we usually seem to have a common understanding (see e.g. [Gou 71: 1]); on the other hand, [RR 93] does not even contain an entry for “data“. It seems not so easy to give a good definition for the term “information“. This has been recently mentioned by Denning [Den 95: 23], and Zemanek mentions 15 definitions for “information“ [Zem 92: 10 .. 18].

Even for the computer, which is at the very beginning of our discipline and which is an essential ingredient of all computing devices and systems, Zemanek gives 7 different definitions [Zem 92: 4 .. 8].

References

AU 92 Aho, Alfred V.; Ullman, Jeffrey D.: Foundations of Computer Science. Computer Science Press, New York, 1992. 0-7167-8233-2

BDM 91 Vitek, A.; Tvrdy, I.; Reinhardt, R.; Mohar, B.; Martinec, M.; Dolenc, T.; Batagelj, V.: Problems in Programming - Experience through Practice. John Wiley & Sons, Chichester etc., 1991. 0-471-93017-2

Ber 96 Berman, Michael A.: OO Techniques in the Classroom. ACM SIGPLAN Notices 31, 2 (1996) 4 .. 5

Ber 99 Berghel, Hal: How Xday Figures in the Y2K Countdown. CACM 42, 5 (1999) 11..15

BF 97 Freytag, Jürgen; Brauer, Wilfried: Objektorientierung in der Ausbildung. Informatik Spektrum 20, 6 (1997) 326 .. 327

BK 90 Beitz, W.; Küttner, K.-H. (Hrsg.): Dubbel - Taschenbuch für den Maschinenbau. 17. Aufl., Springer, 1990. 3-540-52381-2

BR 94 Brunnstein, K.; Raubold, E. (Hrsg.): 13th World Computer Congress 94, Volume 2. Elsevier Science B.V. 1994. 0-444-81987-8

Bre 77 Bresch, Carsten: Zwischenstufe Leben - Evolution ohne Ziel ? - R.Piper & Co., München 1977. 3-492-02270-7

Bro 87 Brooks, Frederick P., Jr.: No Silver Bullet - Essence and Accidents of Software Engineering. Computer 20,4 (1987) 10..19

Cal 95 Calter, Paul: Technical Mathematics with Calculus. 3rd ed., Prentice Hall, Englewood Cliffs, 1995. 0-13-898875-7

CDG 89 Denning, Peter J. (Chairman); Comer, Douglas E.; Gries, David; Mulder, Michael C.; Tucker, Allen; Turner, Joe E.; Young, Paul A.: Computing as a Discipline. CACM 22,1 (1989) 9..23

CM 80 Mead, Carver; Conway, Lynn: Introduction to VLSI Systems. Addison-Wesley, 1980. 0-201-04358-0

Com 80 CompSac ‘80 - IEEE Computer, Software, and Applications Conference, 1980.

Cza 79 Czauderna, Karl-Heinz: Konrad Zuse, der Weg zu seinem Computer Z3. Berichte der Gesellschaft für Mathematik und Datenverarbeitung; Nr. 120. R. Oldenbourg, München, Wien, 1979. 3-486-23141-3

Den 92 Denert, E.: Software Engineering. Springer, Berlin etc., 1992. 3-540-53404-0

Den 95 Denning, Peter J.: Can There Be a Science of Information? ACM Computing Surv. 27,1 (1995) 23..25

Den 97 Denning, Peter J.: How We Will Learn = [DM 97: 267 .. 286]

DH 81 Davis, Philip J.; Hersh, Reuben: The Mathematical Experience. Birkhäuser, Boston etc., 1981. 3-7643-3018-X

DM 97 Denning, Peter J.; Metcalfe, Robert M.: Beyond Calculation - The Next Fifty Years of Computing -. Copernicus, New York, 1997. 0-387-94932-1

Dre 79 Dreyfus, Hubert L.: What Computers Can’t Do. Harper Collins, New York, 1979. 0-06-090624-3

End 92 Endres, A.: Der rechtliche Schutz von Software: Aktuelle Fragen und Probleme. Informatik Spektrum 15, 2 (1992) 89..100

EWG 92 Richtlinie 92/59/EWG DES RATES vom 29. Juni 1992 über die allgemeine Produktsicherheit. Amtsblatt der Europ. Gemeinschaften v. 11.8.92, L 228/24..32

FAZ 94 FAZ Magazin 1994.12.16 / 6

Fin 80 Finlay, Hugh D., Esq.: Trade Secret Protection of Software. = [Com 80: 135..139]

Fra 97 Frankston, Bob: Beyond Limits = [DM 97: 43 .. 57]

Ger 1890 Gerhardt, C.J. (Hrsg.): Die philosophischen Schriften von Gottfried Wilhelm Leibniz. Siebenter Band. Berlin, Weidmann-sche Buchhandlung, 1890

GI 92 Fachausschuß 7.4 der GI: Zur Berufssituation der Informatiker 1991. Informatik Spektrum 15,6 (1992) 335..351

Gib 94 Gibbs, W. Wayt: Software's Chronic Crisis. Scientific American 271,3 (1994) 72..81

Gou 71 Gould, I. H. (ed): IFIP Guide to Concepts and Terms in Data Processing. North-Holland, Amsterdam etc., 1971. 0-7204-2047-4

Hil 26 Hilbert, David: Über das Unendliche. Math. Annalen 95, 1926, 161..190

Hof 80 Hofstadter, Douglas R.: Gödel, Escher, Bach: An Eternal Golden Braid. Vintage Books, New York 1980. 0-394-74502-7

HW 92 Weizenbaum, Joseph; Haefner, Klaus: Sind Computer die besseren Menschen ? - Ein Streitgespräch -. Piper, München 1992. 3-492-11470-9

Kit 94 Kittler, Friedrich: Der zerstreute Mathematiker. FAZ 1994.11.26, Beilage „Bilder und Zeiten"

Knu 76 Knuth, Donald E.: Mathematics and Computer Science: Coping with Finiteness. Science 194, 4271 (1976) 1235..1242

Krä 88 Krämer, Sibylle: Symbolische Maschinen - Die Idee der Formalisierung in geschichtlichem Abriß -. Wiss. Buchges., Darmstadt, 1988. 3-534-03207-1

KW 97 Winkler, Jürgen F. H.; Kauer, Stefan: Proving Assertions is also Useful. SIGPLAN Notices 32, 8 (1997) 38 ..41

Lev 95 Leveson, Nancy G.: Safeware - System Safety and Computers -. Addison-Wesley, 1995. 0-201-11972-2

Lou 89 Loui, Michael C.: Computer science is an engineering discipline. Eng. Education 78, 3 (1987) 175..178

Lou 95 Loui, Michael C.: Computer Science is a New Engineering Discipline. ACM Computing Surv. 27,1 (1995) 31..32

McC 93 McConnell, Steve: Code Complete. Microsoft Press, Redmont, 1993. 1-55615-484-4

McD 91 McDermid, J. A. (ed.): Software Engineer's Reference Book. Butterworth/Heinemann, Oxford etc., 1991. 0-7506-1040-9

Mei 95 Meissner, Loren P.: Fortran 90. PWS Publ. Comp., Boston etc., 1995. 0-534-93372-6

Mes 67 Meschkowski, Herbert: Probleme des Unendlichen - Werk und Leben Georg Cantors -. Friedr. Vieweg & Sohn, Braun-schweig 1967

Mon 82 Monod, Jaques: Zufall und Notwendigkeit. Deutscher Taschenbuchverlag, München, 5. Aufl. 1982. 3-423-01069-X

NM 99 Neumann, Peter G.; McCullagh, Declan: Risks of Y2K. CACM 42, 6 (1999) 144

NW 89 Winkler, J.F.H.; Nievergelt, J. Wie soll die Fakultätsfunktion programmiert werden? Informatik-Spektrum 12,4 (1989) 220..221.

Oli 93 Oliver Ian: Programming Classics - Implementing the World's Best Algorithms. Prentice Hall, New York etc., 1993. 0-13-100413-1

Par 94 Parnas, David Lorge: Inspection of Safety-Critical Software Using Program Function Tables = [BR 94: 270..277]

Pie 87 Pietsch, A.: Eigenvalues and s-Numbers. Akad. Verlagsgesellschaft Geest & Portig K.-G., Leipzig, 1987. 3-321-00012-1

Pom 93 Pomberger, G.: Software engineering education - adjusting our sails. Educ. Comput. 8 (1993) 287..294

Reu 97a Kasparov Draws Again With Chess Supercomputer. Reuters, May 7, 1997 11:52 PM EDT; (in PointCast)

Reu 97b Deep Blue Humbles Chess Champion Kasparov. Reuters, May 11, 1997 11:24 PM EDT; (in PointCast)

RR 93 Ralston, Anthony; Reilly, Edwin D. (eds): Encyclopedia of Computer Science. 3rd ed., Chapman & Hall, London etc., 1993. 0-412-49710-7

Rus 84 Russel, Bertrand: A History of Western Philosophy. Counterpoint, London, 1984. 0-04-100045-5

Sed 88 Sedgewick, Robert: Algorithms. Addison-Wesley, Reading etc., 1988. 0-201-06673-4

Som 92 Sommerville, Ian: Software Engineering. Addison-Wesley, Reading etc., 1992. 0-201-56529-3

Tuc 97 Tucker, Allen B., Jr. (ed.): The Computer Science and Engineering Handbook. CRC Press, Boca Raton, 1997. 0-8493-2909-4

TUM 92 Technische Universität München: Fakultät für Informatik. München 1992

Wal 99 Waldo, Jim: The JINI Architecture for Network-Centric Computing. CACM 42, 7 (1999) 76..82

Win 90a Winkler, J F H: Functions not equivalent. Letter to the editor, IEEE Software 7, 3 (1990) 10

Win 90b Winkler, Jürgen F H: Visualisierung in der Software-Entwicklung (Visualization in software development). In: GI '90 - 20. Jahrestagung der GI, eingeladener Hauptvortrag, Informatik-Fachberichte Bd. 257, Springer, 1990, 40..72

Win 97 Winkler, Jürgen F.H.: Was ist und zu welchem Ende benutzt man Software ? Report 97 / 27, Friedrich Schiller University, Dept. of Mathematics and Computing, 1997 („What is and to what end do we use software ?)

Win 98 Winkler, Jürgen F. H.: Characteristics of Computing and Informatics. Proc. XV. IFIP World Computer Congress. Vienna / Budapest, 1998.Aug.31 - Sep.04., CD-ROM Edition file: /doc/000/000/669.htm

Zem 91 Zemanek, Heinz: Weltmacht Computer - Weltmacht der Information. Bechtle Verlag, 1991. 3-7628-0492-3

Zem 92 Zemanek, Heinz: Das geistige Umfeld der Informationstechnik. Springer, Berlin usw. 1992. 3-540-54359-7

Zus 93 Zuse, Konrad: Der Computer - Mein Lebenswerk. Springer, Berlin usw., 3. unv. Aufl. 1993. 3-540-56292-3

Notes

„Now it is obvious that no *finite* machine can include infinity“ Charles Babbage 1864 [Bab 1864: 124]

Bab 1864 Babbage, Charles: Passages from a life of a philosopher. Longman, Green, Longman, Roberts and Green, London, 1864 = Cambell-Kelly, Martin (ed.): The Works of CHARLES BABBAGE. William Pickering, London, 1989, Vol. 11. 1-85196-511-4