Let's Design Everything Again: Thoughts on Computing and Its Teaching

Ricardo Baeza-Yates

"Man is the only animal to trip over the same stone twice"
"*El hombre es el único animal capaz de tropezar dos veces en la misma piedra*"
Spanish popular saying.

"Everything is highly intertwined"
Ted Nelson, inventor of Xanadú.

The aim of this article is to give a personal view of our field; a critical and constructive analysis of the current state of affairs and the implications this has on education. Although this view is rooted in a local context, most of the issues considered are also of relevance in a global context.

Keywords: Training in Computing, Software Engineering, Databases, Interfaces, Public Software

1 Introduction and Motivation

The first problem we have with our field is what name to give it. Are we talking about Computer Science and/or Computer Engineering or Informatics? Is the correct term computer, computation, computational or informatics? Where do information systems fit in or should they be they considered as a separate area? I still do not have any clear answers. Perhaps the problem is intrinsic. As the joke goes: Computer Science has two problems: Computer and Science. Have you ever heard of the science of washing machines or any other machine? Do mathematics or physics need to say they are sciences? In short it's a problem of coming of age, of maturity, and consequently of insecurity. In any case it is clear that what we do is rooted both in engineering and basic science. Before we go on it is necessary to make something clear. Many of the things I say here are self-evident or common sense. Nevertheless, despite being obvious, many of them have not been said by anybody else and that's why I am saying them here. Is it that they are very obvious or that they only become obvious once you know them?

The ultimate aim of any software is to transmit some knowledge to the mind of the person using it and vice versa. The biggest bottleneck occurs at interfaces, at the final point of communication with the user, and the problem lies not only in bandwidth but also in the very way information is represented (Figure 1). As Djikstra said recently, we have still not been able to rise the challenge of making large software systems less complex.

How to eliminate this bottleneck is the main aim of this article, which may seem like a hotchpotch of apparently unconnected arguments. However, we often forget to analyse our universe as a whole, from the point of view of an outside observer. For anything we wish to study, form and content should be given their due degree of importance and consideration. I will start by presenting an analysis of the relationship between technology and culture, and an analysis of computing itself and in the context of our profession. I will go on to talk

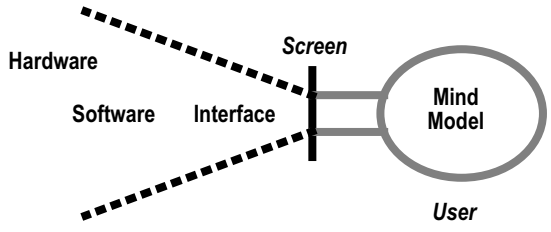

Fig. 1: Comunication of Information.

Ricardo Baeza-Yates is a Doctor (Ph.D.) in Computer Science (University of Waterloo, Canada), Magister in Electrical Engineering and Computer Sciences from the University of Chile and Electrical Civil Engineer by the same university. He is currently a Tenured Professor in the Department of Computer Science of the University of Chile and his fields of research are algorithms, information retrieval and visualization. He is the author of several books. He has twice been president of the Chilean CS Society (SCCC). Currently, among other posts, he is president of the CLEI (*Centro Latinoamericano de Estudios en Informática*) and is the international coordinator for the subprogram of IT and applied electronics of CYTED (Iberoamerican Cooperation Programme on Science and Technology). During the year 2000 he set up a search engine for the Chilean Web <http://www.todocl.cl>.

about some of the implications on education of this analysis and I will put forward some basic ideas regarding what to teach and how to teach it. In short, my message is that we should not forget the many kinds and many levels of relationships that exist, that we should constantly test the hypotheses we make and that we should really redesign and not merely reengineer.

2 Technology and Society

We are moving in a highly technological world so it is important to understand the way technology and our society interact. The relationship between technology and culture is one of love and hate, of successes and failures, of visionaries and monopolies.

How long does technology need to reach all levels of society? In many cases not many years, from a historian's point of view. For example, printing took a hundred years to reach all of Europe. However, for someone living at that time this was considerably longer than an average lifespan. The telephone or commercial aviation took more than 30 years to have any impact on a significant percentage of the population. The fax was invented last century but has only had any real impact on society in the last couple of decades and it is still not present in most homes. To quote Norman: "*Today we often hear that the pace of change has speeded up, that changes happen in "Internet time", in months or weeks, not decades or years. False*." [Norman 98]. The Internet has been with us for more than 30 years and it is still not in every home, even in developed countries.

History is packed with examples of innovative or high quality technologies which flopped. Here a few: Edison invented the phonograph in 1877 and in spite of this his company failed; nobody has heard of the first car company in the United States (Duryea); the Macintosh operating system was much superior to DOS, but it lost the commercial battle; Sony's Beta technology was better than VHS; etc.

One of the reasons for this failure is an inability to understand what the customer really wants. Logic does not always defeat the whims of the market. In the case of Edison, the problem was their choice of artists for their records. The public wanted to hear the best known singers. It didn't matter if there were others equally as good or better, it was the name that counted. In the case of a product, again quoting Norman: "*… it only matters that what is being offered is good enough for the purpose. Moreover, if you lead the marketplace in sales, it is permissible to use a nonstandard infrastructure. After all, if you have the majority of customers, then what you do becomes the standard. Your competitors have little choice but to follow. If you are not the leader, then having nonstandard infrastructure is a bad idea. Ultimately, it leads to extinction*" [Norman 98]. For software there are many examples of this kind.

2.1 Accidental Products, Damaged Products

Sometimes bad products are sold as part of a marketing strategy to break new ground and get a name known. Speed is the important thing, not whether a product actually works or not. For this reason haste is the most important impediment to quality. Often technological quality, a good design, user-friendliness (or the contrary), are equally unimportant. To quote Norman: "*What kind of world is this, anyway, where horrible products don't matter?*". It's our world and the only kind we have.

"*Buy! The only 100% compatible 64 bit RISC computer, Posix compatible operating system and with total connectivity, even ATM. The solution for this world of open systems. Plus the software that you need free: object oriented development tools with graphic intelligent interface, transactional database and SQL server with support for 99 known or future formats and 64 more utilitaries*." Beware! As with other products of this consumer market of ours, the reality we find is different from what the advertising claims (although there are always exceptions to prove the rule).

Most of the most popular and influential technologies in computing nowadays were never meant to be used as they are used today, nor were they meant to dominate the market the way they do. The success of MS-DOS/Windows, Unix/Linux, several programming languages and the World Wide Web are proof of this. By this I don't mean they are good or bad, merely that they were not designed for what they have become today. Those of you who know the history of DOS and CP/M will know what I am talking about, or how prototypes like X-Windows or Mosaic have changed the course of computing history. These and other cases like them are clear examples of *accidental* products and systems that have ended up dominating our world while at the same time making us less sure of our ability to shape the future of technology.

According to Karl Reed this should be a task for the professionals of computing, who should not only report on new advances or problems but also steer the development and use of information technologies [Reed 98]. According to Reed this has happened due to an aversion to planning. I prefer to think that what is happening is that our planning is not successful either because of the environment I am about to describe, or because of a lack of time.

Would you accept a car with an engine you had to switch off and start up again in order to make it work (people tell this as a joke but it isn't really so funny) or a television that gave you an electric shock every now and then? Of course you wouldn't, but that's what we accept in software. In March 1999 Microsoft recognized in a private meeting with its distributors that five thousand errors in Windows'95 had been corrected in Windows'98 (but they didn't say how many new ones had been added!). That is to say that millions of defective software copies were sold at an irreparable cost to their customers. In short, what we need are serious, high quality software companies if we want to get back on course again. This also means changing marketing policies and not accepting with half finished products.

Sadly, better solutions take time and many companies are reluctant to invest in development and/or research if technology changes every year. That is the current paradox, and while it is understandable it is also damaging. Today, instead of thinking, we are using familiar solutions that were no use 20 years ago. It's true that it's not a good idea to re-invent the wheel, but neither is it a good idea to have no time to invent anything.

There will come a time when it will be pointless to invest in new technological advances if we cannot use them properly. When this time comes we will have to go back to the drawing board and think. Yes, think; something that the pace of the modern world let's us do very little of, to the point where we are forgetting how to do it.

2.2 Implications

The use of technology is conditioned to a great extent by historical accidents and cultural variables as much as by the nature of the technology itself. Social, cultural and organizational aspects of technology are much more complex than mere technical aspects. After a technology has been established, it becomes entrenched and it is very difficult to make any changes. We should try to bear this in mind for the future, since in our field there are many examples which we will be looking at later on, the most obvious one being the Windows operating system.

To close, let us remember that technological advances are of no use unless society advances alongside them. In fact some studies show that productivity has not actually increased, in spite of the degree of computerization the world has experienced [Brynjolfsson/Hitt 98], [Dewan/Kraemer 98].

3 Our Technological Environment

Technology advances so quickly that it leaves no time for thinking or using it to design efficient solutions. I would now like to show how many of the solutions we use (and consequently their designs) are based on premises that are no longer valid; and how solutions which existed in the past are now coming back into their own. However, in the long run the technological race is self-defeating. But let us start with a little history.

3.1. A Little History

Things have been rediscovered so many times. You would think that Windows had discovered graphic interfaces if you didn't know the history of Xerox Parc and then Apple. Others believe that RISC technology was invented by IBM with its line of RS-6000 equipment, unaware that it was developed in the mid 70s.

Let us make a brief analysis of the development of computing in recent years. Many technologies have advanced exponentially. This is the case of Moore's famous law that states that the capacity of microprocessors doubles every 18 months. This prediction, made in 1965, is still proving to be true [Hamilton 99]. The same can be said of the memory chip capacity per dollar which has increased by a factor of 134 million in the last 40 years. Recently a similar rate of growth can be seen with the Internet. The number of computers connected doubles every 15 months. This cannot continue indefinitely since today more than 20% of the world's computers are already connected and it would mean there would be as many computers as people by the year 2010. The growth of the Web is even more impressive. Since 1993 the number of servers doubles every 3 months, and today the figure stands at over 30 million. Similarly the United State's Internet capacity increased by 1,000 times in the 80s and possibly by that much in the 90s. In spite of this, traffic on the net is growing at an even faster rate.

If we compare a present day personal computer with a typical one 20 years ago, we can see that the storage capacity has increased more than 1,000 fold and the processing capacity by at least 150 times. These drastic differences in growth rates lead to problems. For example, improvements in speed on the networks (several Gb per second is being forecast) are difficult to exploit since processors are not as fast. Other technologies have not developed at the same pace, such as disk transfer rates, which have increased by much less, and are now as input/output one of computing's bottlenecks.

As far as we users are concerned we haven't even doubled our capacity and yet sometimes I am amazed at how easy it is to get used to something bigger (as one of the interpretations of Murphy's law states: *however big your disk is it will always be nearly full*). The same can be said of software, which hasn't undergone any spectacular advances either; you could even go as far as to say that methods haven't changed much in the last 10 years. While it is true that many computational resources are cheap, the solution does not lie in using the design that we already had, but without optimising it and asking the user to buy a computer twice as big and twice as fast.

The most important features of present day computing are largely dictated by the impact of the Internet. Among them we can mention interactivity, distributed information and processing, digitalization and the use of multiple media, use of shared resources and collaborative systems, standardization and open systems. It is difficult to make predictions; many have got it wrong in the past. The most famous examples are from IBM's founder, Thomas Watson, in 1943: "*I think there is a market for five computers*" and from Digital's founder, Kenneth Olsen, in 1977: "*There is no reason why a person would want to have a computer in their home*". Among short term speculations are: the mass adoption of optical fibre, the development of wireless networks, the convergence of PCs with Unix workstations, the greater use of collaborative tools and of course the total massification of computers and the Internet.

3.2. Operating Systems and Networks

Most premises upon which traditional operating systems were founded are no longer valid. In the past hardware resources (CPU, memory, disk) were very expensive so their use was limited. This led to many solutions being unnecessarily complicated, in order to reduce their cost or the impact they had on shared reduces.

These premises changed in the 80s and while costs went down, the speed of processors and size of memories increased by more than 100 times. Why then aren't operating systems at least 100 times faster? To adapt existing solutions at first improvements were used in the interfaces (for example the *cache memory*). However the complexity of the solution itself imposed a maximum limit to these improvements. The solution is to simplify the solution. Enter the paradigm "*simplicity equals speed*". One of the corollaries of this paradigm has been the shift from CISC processors to RISC ones.

Also history is forgotten. Windows 1.0 was never sold, Windows 2.0 was a failure and only Windows 3.1 was a success, being no more than a good patch on DOS. Windows NT needs a minimum of 16Mb and 32Mb is recommended. What happened to the 64**K** that DOS needed? Is memory so cheap that we can afford to forget about being efficient? Are processors so fast now that we can forget about good data structures and algorithms? Defenders of Windows NT will say that it is much more than DOS, that it includes a system of windows, network connectivity, multiprocessing, etc. All right, but for example Linux with X-Windows works with 4Mb and better with 8Mb. Why then does Windows NT need so many resources? Clearly there is a design problem somewhere. The rise in mobile computing may help to improve designs in this particular area, since with laptops we cannot afford the luxury of having a lot of resources or of using a lot of energy (battery).

A similar phenomenon has occurred in networks. In the past they were expensive and slow. Now they are cheap and fast. Most current technologies have had to adapt to the changes, although there is still a lot more to be done. For instance, ATM was designed in the 60s and is now coming back onto the scene, because it is simple and fast, reaching speeds of 155 Mb/s. However that is still a long way from speeds of several Gb/s that can be reached with optical fibre.

Another example is X-Windows, the most popular Unix windows system, which is transparent to the network protocol used. That is to say, it is a distributed windows system. The communication protocol used by X-Windows assumes that the network is fast and the screen graphics are slow. However nowadays that is not true, because while networks are fast, they are congested and shared by many users. And the speed of screen graphics has also increased.

3.3. The Art of Programming

Programming is perhaps the heart of Computer Science. It is the world of algorithms and data structures and programming paradigms. Throughout its evolution programming has been more an art than a science or engineering. It is not for nothing that Knuth's famous trilogy on algorithms and data structures is called *The Art of Computer Programming*.

For many people programming is not an entirely respectable job; that's what programmers are for. But we should make a distinction between people who are able to design the solution to a problem and turn it into a program and those who are only able to turn the solution into a program. A real programmer, as Yourdon would say, is someone who can carry out the whole process, from analysis to implementation.

Programming keeps you in training for solving problems whether they are big or small. Programming should be satisfying. It should in no way be demeaning for an engineer, or whatever we think we are, to program. Quite the contrary; it is we engineers who will often make the best programs because only we fully understand our own solutions. Another important point is that good code is not the least comprehensible or the most extravagant code, but the one which is the clearest, most efficient and best documented. Many people also equate being a keen programmer with being a hacker. As with any addiction, extremes are not wholesome. Neither should we see hackers as evil programmers. There are good hackers and bad hackers, and the former are indispensable.

3.4. Software Engineering?

In 1999, an important executive of a major US computer company told me: "*We could afford to do it well because we had the resources and we wanted to break into a new market*". Of course at a technical level we would always like to do things well, but the market is telling us otherwise. There is no time, there are no resources, it's now or never. The result is badly designed and poorly tested products. These days the only company in the market that could afford to do things well is Microsoft. But they don't seem to want to.

Perhaps the best place to start is with the famous year 2000 glitch, or the millennium bug (although really this century started in the year 2001). Whichever way you look at it, this was a ridiculous problem which had a massive impact. Should this embarrass us? I don't think so.

Was it a mistake to consider only two digits instead of four for the date? Everybody knows that the main reason was to use less memory, a resource which 20 years ago was much more expensive than it is now. I think it was neither a mistake nor was it good design. The real reason is that none of the designers thought that their software would still be in use after more than 20 years. Not even today do we think that, plagued as we are with annual changes of hardware. It is true that in some instances programs have developed without the original design having changed, but this is not the normal case. Why do we go on using that software? Because of the bad software development habits we mention below.

Computing changes, but that does not mean that it improves. Many companies may prefer not to change software which we know works or which we know *where* it doesn't work. This software survives successive changes of hardware until in many cases it loses its original source code as a result. Other companies have tried to change it, but their projects have failed due to not using the right methodologies and/or tools. There again, today we can see the other extreme. There is an excessive use of resources and the design is of secondary importance. For example, Windows'98 has more than twice the number of lines of code than the latest version of Solaris and occupies much more memory when running. The reader can draw his or her own conclusions as to which operating system is better designed, leaving aside the fact that the more lines of code there are the more chance there is that there will be errors. Just because memory today is cheaper doesn't mean we should overuse it.

Why does this happen? Let's draw a parallel with civil engineering. Can you imagine a bridge being built that falls down five times while under construction due to design faults? Unthinkable. Worse still, can you imagine that, at the very moment of opening that same bridge with 100 people on it, you find a fatal error in its design? Impossible. However, everyone in programming uses trial and error techniques. Now consider the number of designers. A house is designed by one to three architects. What would happen if there were dozens? And when

a house is being built you don't make major changes to its design. How many times do software implementers change the design? Plenty, partly because often they the same people and having two roles without separating them clearly is always a problem. We used to talk a lot about reusability, but it is only now, with class libraries and design patterns, that this word has any real meaning. In the past it was difficult to make use of what had been done by other people for countless reasons: code not available, different language or environment, lack of documentation, etc. Modularity and component independence is vital if we want to integrate different products and technologies. We can also talk about quality. If we add reuse and using the right control tools, in the future we may be talking about real software engineering [IEEE 98]. Although I am aware that others might shoot me down here, I would say that software engineering is actually software handcraft. TeX is perhaps the best example, since in the beginning it was the work of an excellent craftsman, Don Knuth, and for the last 10 years not one error has been found in its code (and the final cost of each single error grows exponentially). Unless we change our way of thinking and stop relying on always being able to test, and that if there are errors it doesn't matter, programming will continue to be an art in which few will be masters and most will always be apprentices. This change will need to be radical, since even the biggest software companies are still not in a position to say that their product has no error. The following examples from Windows illustrate this point.

Windows'95 contained nearly 15 million lines of code. Applying Caper Jones' estimates [Jones 96], a code of this size has a potential number of errors of nearly 3 million, which gives an idea of how many tests need doing. To get this figure down to five thousand requires at least 18 repeat tests [Lewis 98a]. Although software companies should perhaps perform more tests this would raise the cost and delay the products release on the market.

Sadly history shows that bringing out new versions quickly often means a more successful product. This happens because customers do not base their choice on quality, though this is less true of critical products such as a Web server. Here quality is more important which is why the Apache server wins over a Unix type operating system, although it is public domain software. Many companies say that they don't use public domain software because it doesn't have support. But most PC products, especially Windows, don't have support either. Windows NT has around 25 million lines of code, which means more tests should be carried out to ensure required levels of reliability. Moreover Windows NT is supposed to be certified at security level C2 for use on Internet. However a study carried out by Shake Communications Pty. Ltd. revealed 104 problems, some of them very serious ones, which make it vulnerable to hackers [Lewis 98b].

In the case of software, suppositions similar to those regarding operating systems were made: expensive and limited resources. Now resources are cheap and plentiful. But it is also bad to misuse resources by writing software needing large amounts of memory or a lot of available space on the disk. This is acceptable only when it is really necessary, and on most occasions this is not the case. This is another side effect of not having enough time to design software and of producing it in order to bring out new releases as quickly as possible, because that's what the market is demanding. This abuse of technology has a harmful effect. For instance, if we want to do something faster, the most common solution is to buy a faster computer. However it is cheaper and possibly faster to use a better solution (better software, better parameter adjustment, better network configuration, etc.).

3.5. Artificial Intelligence?

Artificial Intelligence is one of the areas of computing which promised most and has progressed the least. Whether it be in games like chess or processing of natural speech, results have shown that good heuristics, black boxes or neural networks are only partially effective. But we are still a long way from the Turing Test. Let me use chess to help me put across my ideas. In May 1997, Gary Kasparov, then world chess champion, was beaten by *Deep Blue* from IBM (*Big Blue*), the champion of chess programs. Has the machine triumphed? By analysing this pseudo-victory of artificial intelligence over man perhaps we can put an end to the abuse of terms like expert or intelligent systems. Isn't a good algorithm intelligent? Is brute force intelligent?

At the beginning of the 50s it was predicted that in 20 years there would be programs capable of defeating the world chess champion. More than twice that time was needed for that to happen. So does this make computer programs intelligent then? No, Deep Blue doesn't think like a person (neither does it think, but let's just say it does something similar, for comparison's sake). Kasparov knows what lines of play to analyse and he studies just a few moves in depth. On the other hand, Deep Blue analyses millions of moves and appraises a large number of positions, but it can do it faster. The fundamental difference lies in intuition, creativity and long term strategy. If Deep Blue had the ability to assess positions like Kasparov does, it would be invincible. However, Deep Blue evaluates a position on the basis of heuristics. That is, rules that work most of the time but other times don't.

The more complex the game is and the longer term the objective is, the more difficult it is to analyse any given position. For example, for some time the best *checkers* program has been better than any human. Why? Because there are far fewer possible positions in checkers and the rules are much simpler, which allows it to assess every possible move. On the other hand, in the oriental game of *Go* it is necessary to control the board little by little, without knowing until the end if many of the pieces are still alive or not. This makes it more difficult to analyse, because long term strategy is required. In this case intuition and experience are much more important than memory (as in the game of *bridge*) or an ability to make rapid calculations (as in checkers).

The first misinterpretation we can make of Deep Blue's win is that it may seem as if computer has defeated man. What has actually happened is that a group of experts in computing and chess have programmed a high powered computer and have succeeded in defeating the world champion. That is to say, a

group of people who have worked over a long period, concentrating especially on working out how to defeat the champion, have had more success than the intelligence and memory of one man working alone. I don't consider it such a big thing that a program can beat one person, since it's an unfair fight. Deep Blue has a large number of processors, it knows more than a million games by heart and can analyse 200 million positions a second. It would be an interesting experiment to see if with less time per match its calculating capacity might be less important. Could Deep Blue defeat a group of grand masters? I doubt it.

There are also factors which have nothing to do with intelligence that affect a chess player's concentration. According to some chess players, Kasparov had a great deal of respect for Deep Blue. Others say that he took his role as defender of mankind very seriously and that his defeat would be a milestone in history. And of course Kasparov is a human being, with emotions, who needs to eat, drink and sleep, and who feels the pressure of knowing that he cannot exert any psychological pressure of his own on his opponent. An opponent who neither makes mistakes nor gets tired. If we look back, one of the reasons for all the successful defences of his title was Kasparov's greater psychological strength.

Man defeats himself every day. Kasparov was defeated in public. That's all. When a computer can read a book, understand it and explain it, that will be something to shout about. Deep Blue is an example of software engineering, of a good program in a world where not many are to be found. A program that has been improved over many years, that uses knowledge from many sources and that has had time to evolve. If we made use of technology as Deep Blue does, we would be in a better world for sure.

3.6. Interfaces with Common Sense

Because of the limitations of the original Macintosh which, in order to keep its cost down, couldn't run two applications simultaneously, (very different from its powerful predecessors: Altos and Lisa), Macintosh's desktop metaphor was not centred on documents. The user was therefore required to choose an application and then choose a document, rather than selecting the document first and then the application to use it with. Though on the surface this looks like the same thing, it meant a radical difference in the development of interfaces. Only for the last few years has it been possible to select a document and run a predefined application or choose one from a menu. To quote Bruce Tognazzini, one of Macintosh's designers: "*We have come to accept that the way to create or edit a document is to open that document inside an application, or tool. This is equivalent to having to slide your entire house inside a hammer before you can hang a picture on the wall or hiving to put your teeth inside your toothbrush before you can brush them*" (from the essay Nehru Jacket Computers in [Tognazzini 96]).

Let's take a look at present day interfaces. The information we store is based on a hierarchy of files and directories in which we navigate from father to son and vice versa. That is to say in just one dimension. Not only that, but we have to remember where each file we create is and what name we gave it (not to mention limitations of length, symbols, or not being able to use identical names). Also, while the screen is a bi-dimensional space, the interface rarely makes use of this fact neither does it learn how we use it nor what order we do things in. For example, I might be moving a file right across the screen to put it in the waste bin and at the last moment my finger slips. Result: two icons end up one on top of the other. The interface might have assumed that what I was trying to do was to get rid of the file! In my opinion, part of the success of browsers and the model imposed by HTML lies in the fact that, as well as being a very simple interface, it has a single level link structure. New paradigms of visual representation of knowledge are already appearing [Greenberg 99].

The computer technology we use should be transparent for the user. In fact how many newbie users only use one directory to put all the files they use in? The user doesn't need to know that there are directories or files. Besides not everything can be classified into directories and files. A file should be able to belong to two or more different classifications which should be able to change as time passes. How we understand things depends on our place in time and space. Our surroundings are not static, but the computer unnecessarily forces us to keep our documents immovable in space and time.

Let's give this some serious thought. The computer should – and can – name and group together files and retrieve them using their content or the values of an attribute. For example, you might say: show all the letters I was editing yesterday, get the first lines of each letter, then choose the line I need. Another baseless premise is that we need a common interface for everybody. People are different, they think and work in different ways. Why not have interfaces which adapt to each user, which can be personalized and which learn the way and the order we do things in? To pave the way for the implementation of new interfaces, we should scrap the past and replace file systems with data organized in a more flexible and powerful way [Baeza-Yates et al. 99]. This leads us to our next subject.

3.7. Databases

One of the biggest problems of current databases is the large number of different models, although the relational model is the predominant one. However new applications need data which is not so structured and rigid: multimedia, hierarchical objects, etc. While suitable models do exist for these types of data, there are no tools which allow us to integrate well two or more models. In fact attempts to incorporate these extensions to the relational models have not been very successful.

If we forget past hypotheses, we may be looking at more powerful and flexible models. An example is the case of objects with dynamic attributes [Baeza-Yates et al. 99]. In this model the objects have a dynamic number of attributes, the values of which have type and are also dynamic. This model can be considered as an extension of the classless object model. However it is also a powerful query language that can handle object sets that satisfy arbitrary attribute conditions, including their non existence or if they have an undefined value.

There are many arguments in favour of this model: simplicity, flexibility and uniformity; the elimination of suppositions

regarding data structures and their relation with objects that contain information; its ease of use; the fact that it allows multiple views of the same information by means of queries; and the fact that it generalizes hierarchical file systems.

This model should simplify the work of users, programmers and applications for handling information. This model is also useful on the Web, where objects can be shared by aggregating specific attributes to each object's use. These objects can be manipulated and transferred in open form using XML.

4 Our Professional Environment

In addition to those problems directly caused by questions of a technological nature, there are also problems related to the market and the professional environment. For example, professional overspecialization, the lack of good software project managers or the scant interaction between theory and practice in software development [Glass 99]. We will be taking a closer look at some of these aspects in the following sections.

4.1 Complaints from Industry

The most commonly heard complaints from industry are that much of what is taught is of no use. Industry is saying that what is needed is for students to get more practical knowledge and skills, so they can apply what they have learnt to a business context and thus make their entry into the real world of industry so much easier. That what is needed are software engineers, not scientists [IEEE 97]. The first thing I would like to say is that all that is true, but it all depends on your point of view. Commercial objectives are short term while university objectives are long term. Other more specific complaints include the lack of industrial patents developed in universities and the absence of entrepreneurial innovation.

What a company wants is a young person with experience. The lack of practical knowledge is difficult to remedy in a system in which the technology changes so quickly. That is why concepts are so important, since they give the ability to adapt and learn. It is true that often companies lose the investment they make in training, but that is normal in a highly competitive market. Specialization (for instance, specific tools) and continuous training are the employer's responsibility, not the university's. However one of the main problems is that by investing in training the employer may lose the employee when he or she gets a better paid job on account of being better trained. This often happens as a result of the employer not giving enough importance to the investment they have made.

Finally, we come to the commercial aspect. I believe this problem goes beyond computer engineers; it is a question of the interaction between technology and society. We cannot have know-alls who are also good sales people with an understanding of business. These skills are often innate and cannot be taught (I often feel like I am trying to teach common sense, and the results are not too heartening). There are already shortcomings in the technical curricula given the present volume of different subjects there are in computing which cannot all be satisfactorily covered.

4.2 University-company Relations

Joint research by universities and companies has always been bogged down by various factors. These include the slow administrative apparatus of universities and the companies view, often justifiable, that universities are incapable of achieving short term goals. We need to develop an infrastructure for applied research and to increase technological transfer, which is what many developing countries really need to export software and maintain growth in this area.

Another way to provide an incentive for applied research would be to create research projects in which it would be obligatory to have some industrial counterpart. These projects need not only be for applied research but can also be for basic research, though with lower budgets and smaller working groups. This would allow specific problems to be tackled and would encourage support from companies since the risks would be smaller.

We should also be bringing universities and companies closer together in a way that is beneficial for both parties. Technological transfer, exchange programs and the like are ideas that have been mooted thousands of times already. The fact that universities register no patents is criticized. The same criticism could be levelled at Chilean software companies. First of all it is very time consuming. Secondly it is not cheap (at least 10,000 USD). Thirdly, in the course of the process the result cannot be published (which flies in the face of the current system of academic assessment based on publications, although this is changing in Europe). And point number four, ideas should not be patentable (for example, an algorithm).

4.3 Monoposoft vs. Open Source

The *Open Source* movement (that is, free source code) is gaining momentum every day and is beginning to attract media attention as a result. The classic example is Linux: Would its creator ever have imagined that it would be used now by millions of people? Meanwhile, Microsoft is fighting with the US federal government and their software is a source of money and jokes [Lewis 98a], [Lewis 98b]. Sadly many of these jokes should make us cry rather than laugh. But these jokes conceal important truths and lead some people to fight against windmills as romantic Don Quixotes.

How can it be that not only is there such a thing as free software, but its source code is public too? This doesn't make any sense in a capitalist market, where it would be hard to imagine asking thousands of programmers to work for nothing. My personal opinion is that Open Source only exists because Microsoft exists. If we have to choose between cheap software and Microsoft software, for many different reasons we are bound to choose the latter. But if the alternative is free we would be willing to take a risk and try that software. Also, though it may seem to be a contradiction, freeware may be better than commercial software. If someone finds an error and reports it, in a matter of minutes you will be able to go to an Internet news group and get a correction for the problem. And if not, a lot of programmers will take a look at the code and one of them will spot where the problem is. This inefficient mechanism is nevertheless highly effective.

Another advantage is that the process is scalable: as the code grows in size more people can get involved in its development. A few years ago a Microsoft internal document that talked about the danger that Open Source represented for the company was leaked onto the Internet (see this and other related subjects in [Sanders 98], [IEEE 99], [Lewis 99a], [Lewis 99b]).

Microsoft is a de facto monopoly. Every two or three years, millions of users have to upgrade their copy of Windows. They don't get complete compatibility with older versions but they do pay prices that keep abreast of the times: a captive market must be exploited. It's like having to regularly move house but not always to a better one. In 1999 Bill Gates published his 12 rules for the effective use of Internet in companies [Gates 99], which show that he was utterly converted to the world of electronic mail. He has also learned the advantages of free software, particularly when it also allows him to blow away the competition: Explorer. From an economic point of view software development in Microsoft is not the most effective (in fact it is the users who find most of the problems, and they can't always get direct help to solve them), but it is definitely the most efficient. Microsoft is perhaps the only company that is capable of stopping this snowball. It could even afford to take 5 years to develop a real operating system and applications with much better interfaces, as described before. But this is not going to happen, because it would mean earning less. Depending on the result of the anti-trust case, in which Microsoft has recently had a favourable but not definitive sentence, and on the advance of public code, this millennium will be the information age or the Microsoft age.

5 Final Comments

I would like to begin by quoting Peter Freeman [Freeman 97]: "*If we compromise the core of computing science, we risk losing long-term foundational skills. If we fail to take into account the concerns of the computing professional, we risk becoming obsolete. The key is to achieve the right balance – but there is more than one way to get there*".

We have to concern ourselves with both form and content. If we can create good professionals they will become agents of change [Garlan et al. 97]. That should be one of the main aims of university and I feel that it has also been a major personal driving force for what I do.

Most of what we learn in our lives is of little use, it's mostly technical knowledge. The important thing is the training associated with that learning process, the development of logical and analytical capabilities, the ability to abstract, conceptualize and solve problems. The objective is not knowledge per se, it is personal development. It is to learn and learn constantly. I believe that there is a better way to do this, by integrating knowledge and new tools in innovative courses in which the student has a greater understanding of the final goal. The main aims should be flexibility, adaptability, to put emphasis on concepts and to facilitate continuous learning.

All sciences have evolved in a real world context, not just in isolation, the origin of calculus being perhaps the most classic example of this. In the past there were people who knew most of what there was to know in terms of scientific knowledge, but nowadays this is very difficult, which forces us into group or multidisciplinary work. These two facts should help us to consider new ways of teaching. Two different lines of action present themselves. Either designing different professionals, based on the three elements involved: people, processes, and technology [IEEE 97], for example, an information architect [Baeza-Yates/Nussbaum 99]; or drastically changing the way we teach by integrating all the suggested classic contents [ACM] into one single problem solving based course [Baeza-Yates 00].

Although these proposals are of a preliminary nature, I think that they are a first step towards designing better, more complete and coherent curricula, and teaching them in a different way, by motivating students and giving them clear explanations of why they are learning each subject and how these subjects are related to the world they live in. In short, integrating everything, in a certain manner returning to the Renaissance, to encyclopedic and enlightened thought. It is also clear that we have to encourage critical thinking and lay great emphasis on matters of design.

This article is at the same time an essay on the many problems surrounding our field and a quiet appeal for sense. In our private lives as in our professional lives, we accept so many things as true, as basic hypotheses that we never question. Similarly, the ideas expressed here should be taken only as one more point of view to be considered. However, I do hope that these lines appeal to your common sense, that sense which is so important and at the same time in such short supply, and in passing create a little awareness of the multiple problems surrounding our field and our daily task. This is a constructive criticism and is in no way intended to be divisive [Glass 99].

Acknowledgements and Notes

I am grateful for comments and encouragement from Omar Alonso, Juan Álvarez, Karin Becker, Tania Bedrax-Waiss, Carlos Castillo, Helena Fernández, Terry Jones, Miguel Nussbaum, Greg Rawlins and Jorge Vidart. This article is an updated and abridged version of a paper that dates from 1999, whose full content, in Spanish, is available at <http://www.baeza.cl/manifest/manifest.html>.

References

[ACM]
ACM-IEEE *Curricula Recommendations for Computer Science and for Information Systems*, <http://www.acm.org/education>.

[Baeza-Yates 00]
Ricardo Baeza-Yates: Un Curso Integrado para Primer Año, *Congreso Iberoamericano de Educación Superior en Computación*, Mexico City, September 2000.

[Baeza-Yates et al. 99]
Ricardo Baeza-Yates, Terry Jones, Greg Rawlins: New Approaches to Manage Information: Attribute-Centric Data Systems, SPIRE'2000, IEEE CS Press, 2000.

[Baeza-Yates/Nussbaum 99]
Ricardo Baeza-Yates, Miguel Nussbaum: The Information Architect: A Missing Link, DCC, Univ. of Chile, Technical Report, 1999 (www.baeza.cl/manifest/infarch.html). Spanish language version presented at the Congreso Iberoamericano de Educación Superior en Computación, Mexico City, September 2000.

[Brynjolfsson/Hitt 98]
Erik Brynjolfsson, Lorin M. Hitt: Beyond The Productivity Paradox, *Communications of The ACM*, volume 41:8, August 1998, pp. 49–55.
[Denning 97]
Peter J. Denning: A New Social Contract for Research, *Comm. of ACM* 40(2), February 1997, pp. 132–134.
[Dewan/Kraemer 98]
Sanjeev Dewan, Kenneth L. Kraemer: International Dimensions of the Productivity Paradox, *Communications of the ACM* 41(8), August 1998, pp. 56–62.
[Freeman 97]
Peter Freeman: Elements of Effective Computer Science, *IEEE-Computer*, November 1997, page 47–48.
[Garlan et al. 97]
David Garlan, David P. Gluch and James E. Tomayko: Agents of Change: Educating Software Engineering Leaders, *IEEE Computer* 30(11), November 1997, pp. 59–65.
[Gates 99]
Bill Gates: *Business @ The Speed of Thought*, Warner Books, 1999.
[Glass 99]
Robert L. Glass: Is Criticism of Computing Academy Inevitably Divisive?, *Comm. of the ACM* 42(6), June 1999, pp. 11–13.
[Greenberg 99]
AIlan Greenberg: Facing Up to New Interfaces, *IEEE Computer*, April 1999, pp. 14–16.
[Hamilton 99]
Scott Hamilton:, Taking Moore's Law Into the Next Century, *IEEE Computer*, January 1999, pp. 43–48.
[IEEE 97]
IEEE Special Issue, Status of Software Engineering Education and Training, *IEEE Software*, Nov/Dec 1997.
[IEEE 98]
IEEE Special Issue on the Future of Computing and Software Engineering, *IEEE Computer*, January 1998.
[IEEE 99]
IEEE Special Issue on Linux, *IEEE Software*, January 1999.
[Jones 96]
Capers Jones: Software Estimating Rules of Thumb, *IEEE Computer*, May 1996, pp. 117–118.
[Lewis 98a]
Ted Lewis: Joe Sixpack, Larry Lemming and Ralph Nader, *IEEE Computer*, July 1998, pp. 107–109.
[Lewis 98b]
Ted Lewis: What to Do About Microsoft, *IEEE Computer*, September 1998, pp. 109–112.
[Lewis 99a]
Ted Lewi:, The Open Source Acid Test, *IEEE Computer*, February 1999, pp. 125–128.
[Lewis 99b]
Ted Lewis: Asbestos Pajamas: An Open Source Dialogue, *IEEE Computer*, April 1999, pp. 108–112.
[Norman 98]
Donald A. Norman: The Invisible Computer, *MIT Press*, 1998.
[Reed 98]
Karl Reed, Why the CS should help chart the Future of IT. *IEEE Computer*, July 1998, pp. 77–78.
[Sanders 98]
James Sanders: Linux, Open Source, and Software's Future, *IEEE Software*, Sept./Oct. 1998, pp. 88–91.
[Tognazzini 96]
Bruce Tognazzini: Tog on Software Design, Addison Wesley, 1996

English translation by ***Steve Turpin***