THE PROBLEM OF DATA MANAGEMENT

Presented by S. Bedikar
Burroughs Corporation
Detroit, U.S.A.

To New Zealand Computer Society Conference

1970

THE PROBLEM OF DATA MANAGEMENT

Presented by S. Bedikar
Burroughs Corporation
Detroit, U.S.A.

To New Zealand Computer Society Conference

1970

MAIN SPEECH NO. 4

THE PROBLEM OF DATA MANAGEMENT

Presented by S. Bedekar
Burroughs Corporation
Detroit, U.S.A.

Chairman P.W. Harpham,
Systems and Programmes Ltd
Wellington.

Mr P.W. Harpham,

In introducing Shree Bedekar, I would like to make a few introductory remarks of my own. Our conference is now approaching its end and each of us should appreciate some things a little more deeply. I think I appreciate more clearly the emerging maturity and humanity of our profession; it is made up of people - that is pretty clear to us. I would like to describe a couple of roles the people in our profession play, but we cannot talk about people as if they were definable packages. Shakespeare understood this: "Each man in his time plays many parts". That is obviously true. But as our profession matures we will attach labels to the different roles that we must play in it. We have talked quite a lot about the computer scientist and his role is perhaps to study computers to gain insight and understanding. We have also the role of the software engineer whose task is to create systems that work on time, reliably, and within a cost constraint, but we have not yet properly recognised the engineer's role.

I see a distinction between computer science and the engineering task of building systems time after time, in much the same way as we have, say, a chemist studying concrete chemistry and engineers using concrete to put up buildings. Now as yet, the engineer's role has not been properly recognised as a field for study and teaching, so that in a lot of cases the software engineer is terribly concerned with instructions still. Shakespeare know about these as well: "Bloody instructions which being learned return to plague the inventor". Now the software engineer needs help. If he is going to put up building after building essentially he needs scaffolding, bull-dozers, concrete mixers; he needs aids. Now too often, when somebody produces something practical - one of these aids - the inventor remains faceless. We are very fortunate tonight to have Mr Shree Bedekar to address us, so that we can know something about the man who fathered Burroughs' FORTE system, which is a very great aid in this software engineering field. I notice that it is called the "FORTE system" and not the "Bedekar system". This is a pity.

Mr Bedekar is manager of Data Management Systems for the Burroughs Corporation. In this capacity he is responsible for the design and development of advanced file organisation techniques, data base management systems, interactive problem-oriented language creation and translation systems and data communication capabilities. He joined Burroughs in 1966 from the Chrysler Corporation where he had been responsible for the application of simulation techniques to the design of industrial processes. He has a degree in engineering from the University of Poona, India, and a Master of Science degree in Industrial Engineering from Wayne State University in Detroit. I think we are very fortunate to have Mr Bedekar with us and I would hope that we will hear quite a lot more of his work in the future. Perhaps one day we will even see his name attached to some of this work. I will now ask Mr Bedekar to address you.

THE PROBLEM OF DATA MANAGEMENT

S. Bedekar

Somebody suggested to me a few minutes ago that if I got up here and could not think of anything to say, I could do a rope trick!

The topic of discussion today is the problem of data management in general, and the approaches people have taken in developing data management systems. I would like to point out a couple of things that have happened in the recent past. For one thing, we are getting into data handling situations which we have never faced before. A major police network in Europe consists of something like twenty-seven million records on-line, and this is a tremendous amount of data. If you take a major bank in the United States which is responsible for a balance of payment solution, the size of the data involved there is easily equal to a billion bytes of storage on-line. If you take a plant automation system which is responsible for the control of a rolling mill for example, you have another four or five hundred million items of data on-line. These large data-handling situations have really focused our attention on the basic problem of being able to manage data.

If you look, from the user's point of view, at the primary costs involved in developing an application, they primarily fall into five categories. The first major category is to translate the user's needs through some device normally known as a "systems analyst" into a form that a computer can understand, using a programming language or similar technique. There is also the problem of specifying the relevant data, and in essence, designing this data in terms of sizes of items, in terms of representation of items, and so on. Then comes the problem of the detailed transaction specification in the sense that once one finds a part number, what does one do with it? Then comes the problem of implementing this system and that involves a long response time. I was involved in a situation with a major automobile manufacturer where we went and proposed a substantial saving in tens of thousands of dollars, in terms of the "leased cost" of the machine if they were to use our system. The gentleman involved looked at us and explained that he really did not worry about that, as his costs in not controlling his inventory were so high that if we could implement a system sooner, or if we could provide better response time for his transaction related programs, he could probably justify completely redundant hardware. The last major cost in the area of developing applications is that of maintenance, and I submit that we really do not understand the size of this cost.

There is an Auerbach report which states that if it takes you six months to develop an application package in a progressive organisation, the life of that program will be roughly two and a half years, which means that people pay roughly four to five hundred percent of the development cost in support of it. This is a fantastic amount of money. If you allow me to go back to what Dr Hamming was saying, that if we take one hundred million words of programs that are placed on the machine and attach some arbitrary cost during the development phase, we are going to run out of our capacity to support these software systems very quickly.

In addition to that, collectively there seems to be a conspiracy to talk about generation changes in the sense that technology progresses, and changed technology is going to bring about different computer systems. The equipment capacity changes, the languages change, and things like word lengths, speed and representation change. Peripherals change.

Today we have data communication peripherals probably being bred faster than rabbits, and the problem in dealing with these changed environments produces the problem of conversion. I know of a company that went from a competitive piece of gear to another company and the conversion cost involved was roughly eight hundred thousand dollars. This is a fantastic amount of money to pay for a conversion with no significant improvement in performance. There are no alternatives unless it is done, so that there are these primary costs involved in applications. Then there are the problems of conversion because technology is going to change.

The third factor to look at is the trend in regard to equipment. There is a paper by Webber that was published. It was a paper that relates to measurements of computers and he has an elaborate formula which looks quite a bit more complicated than it really is. What it really says is that as time goes on, for each dollar, the amount of computing available will be higher. The costs that have changed - in fact are changing - are the software development costs. In the U.S. in an organisation like ours, the direct cost involved in programming increases annually; each year you can count on the salaries going up by roughly 11%. If you look at software, it is a very peculiar kind of business. Software primarily is an imperfect solution to a problem. If we, as manufacturers, knew for sure how people were going to do their payroll every time, with no deviation from it, economically it would be much cheaper for us to build a payroll machine and market that and make a considerable amount of money. The tragedy is (or maybe it is not a tragedy), the problem is that we do not know what kinds of uses the computer will be put to, so in the near future unless there are radical changes in the technology of software, the software and systems programming, and applications development costs can only go up. If you are to take a total computer-related cost picture in a given organisation, some very interesting things develop. One of Burroughs customers is a utility company. They have roughly a 4.3 million dollar budget every year of which 1.2 million dollars is hardware lease. All the rest of it is the cost related to providing the computer-related service. This company is run quite conservatively. I know companies where the ratio is much more lop-sided in favour of the software costs. One way to say it is that if 80% of all computer-related costs in an organisation are people programming, re-programming, designing, and redesigning, it would not be an over-estimate to say that only 20% of the total costs are directly related to hardware. What we really need to concentrate on is to attack the 80% of that pie. In other words, if we can reduce the cost involved in the 80% of the computer-related service, we could probably afford to go into an organisation and sell much larger configurations. We cannot seem to be able to do it without our motives being misconstrued.

The traditional method of programming normally involves the following cycle. Somebody recognises a problem that can be computerised. Having recognised the problem, it is translated through the systems analyst who develops it into some sort of programming strategy. The programming strategy involves data specification, storage and retrieval techniques, and (if it is a decentrally written program) some sort of an audit and recovery mechanism. Having done that, he codes it and debugs it (and all of us know that our programs work the first time), so that turnaround costs are quite significant. The problems with doing business this way are that these applications become frozen and I will give you some classic examples of this.

The Massachusetts Institute of Technology, a number of years ago, developed mathematical procedures to trace the path of a cutter tool which is used in quite a few instances for different numerically-controlled machines. I do not precisely remember what that date was but through the years these programs have been translated from one

machine language to another, from that to a compiler level language and then from that one compiler level language to another. There have been mistakes; there have been improvisations; but there has been no fundamental change to that design, and the primary reason is that the cost involved in converting it is so high that having done that, nobody has enough money to really sit down and do additional design in the area.

The Naval Weapons Laboratory developed a scheduling system. This scheduling system had some very sophisticated algorithms, and personally, I have spent many a night converting that program to our equipment with the glaring inefficiencies and the glaring inadequacies in that program. The problem was that with the time that was available to make that program available on our system it was not possible to undertake major redesign. These examples are almost classical in their nature, but I am sure all of us in our respective installations have programs which are essentially frozen; applications that have been carried from one generation to the next. There are enough people here who, if they looked into their own hearts, would find bad systems that would cost roughly $60,000 a month, running on their equipment.

In addition to this, there is also the problem of non-transferable data. If I, as a programmer, am responsible for my own physical mapping schemes, storage and retrieval schemes, data description and so on, unless my description of data and my accessing techniques are generally available, which they are not, because there is no way to document a program adequately, the data that is used by one system is practically unusable by another system. A case in point is the U.S. Census Department which probably has the biggest fund of information about people in the United States. But to read that data and make some sense out of it, the one thing that you do not need to worry about is for what purpose you are going to do it, because you will spend more time figuring out whether it is represented in 4 bit, 6 bit, or 8 bit; whether it is on a nine channel tape or a seven channel tape, and so on. This data is practically inaccessible. Individual designs, because programmers are a creative breed, involve some very peculiar things. Hardware idiosyncrasies creep into programs and if you are a programmer worth your salt you are going to write pornographic code. You are going to write code that tends to take undue advantage of the machine. The classic example of this type is a very well-known machine for which there was an instruction set specified by the manufacturer and a leading organisation ran some tests on it and realised that there were a large number of operators on that machine which were not specified by the manufacturer. Somebody in his spare time wrote a little routine to find out what all those unspecified operators really meant. I know that the documentation about the unspecified operators on that machine is much larger than the documentation that comes with the system for the specified operators! These pieces of random information are of great interest in retrospect but they are a nightmare when you are dealing with these types of programs in real life.

If you look at the debugging cost you will find that the majority of the debugging that is involved in the computing system is primarily in the area of storage and retrieval; changes from one level of memory to another. You have to deal with a head-per-track disk file with a record length of 180 bytes, and try to take it to a disk pack that has a length of 240 bytes; changes to sizes of record; changes to the basic specification of data means increased costs. Hence the traditional method of doing business has some severe problems. We here today, are not the first people to realise that these problems have existed. People have attempted to find solutions to these problems. These solutions have tended to fall into two categories.

One category is special solutions. These are solutions of the type where you attempt to recognise as many of those idiosyncrasies of data that you can, and you do that with a deadline hanging over you so you miss most of them, and develop highly sophisticated programming systems that are unique solutions to the data handling problems of one installation, and which are not transferable. The alternative is to develop programming aids or to develop programs that write programs. One of the examples of this form is a generator. An example, which we developed, is Disk FORTE. There have been systems that generate operating systems, but those solutions do not get away from the machine dependancy of the solution. In most cases recompilation is not in itself a traumatic problem as long as you do not have obsolete programs running against live data. Recovery of that form in many cases is quite expensive. I know of a situation in the United States where there is a State Police Information Network against which was run a program that essentially destroyed the data base. The primary problem was that the record sizes were different and the link field in one position did not correspond with the changed data specification. The system was down for roughly a week while they tried to reconstruct the mess that had been created.

Typical application programmers do not seem to realise the enormity of the program. They do not seem to realise that programming is a business in a commercial environment; they do not write programs because it is fun (despite the fact that it is).

A typical applications program consists of four things. The first of these is some form of data description. This is, in COBOL, very formal, and you call it the "data division". In ALGOL, if you are a decent programmer, you have a bunch of "defines" that describe the various layouts. If you are an ESPOL programmer you conceptually use the term "layout" and describe what it is. FORTRAN has equal facilities, so does PL/I. The second part of the program is some sort of logic that deals with validation of input, and some transaction-related functions in the sense that having encountered a customer number, what do we do with it? Then there are the GET and PUT functions, and finally there is audit and recovery. If you are to take a typical organisation, there are many such programs concurrently under development, each one of them having data description, input-validation, transaction-related logic, GET and PUT functions and audit and recovery. The majority of the costs involved in developing these applications are not in the area of transaction-related logic or the input-validation logic. The majority of the costs are in the area of data description, audit and recovery and GET and PUT functions. So, let us assume that we could figure out some way where we took away from the applications programmer not only the responsibility but the ability to describe data, to write his own GET and PUT functions and his own audit and recovery programs. If you could provide this as a service to the applications programmer, you would have something called a data-related service.

This data-related service is what a data management system is. It primarily takes away from the applications programmer the responsibility in the area of data description, GET and PUT functions, and audit and recovery functions. One can compare a data management system very easily with an operating system. Operating systems were developed because we realised that when we were in the process of doing an I/O, the processor was not doing anything, and some devious programmer figured out a way whereby he could keep it busy. Having done that we extended it further and we said that we should insulate the application programs from the nitty gritty of the hardware. Further, as operating systems became more sophisticated, we said that the only reasonable way to write programs is in high-level languages so that you can talk about the problem

and not about addresses and not about nitty-gritties of a given piece of hardware. We did that to allow people to express logic very conveniently in high level languages, but we completely missed the boat in terms of data. Consider a language like COBOL. For those people who write COBOL, I would ask, "What does COMP-1 mean to an application program? What does COMP-2 mean to an application program? What does REDEFINE mean to an application program?" So, that in just the same way that today's operating systems insulate the application programs (and I am including the term "application program" more widely than it is conventionally applied), from the idiosyncrasies of hardware, a data management system is expected to relieve the application program from the nitty-gritties of data. He should ask for data by name, and receive it in that form.

So, you can summarise briefly the five major functions of a data management system. It must provide for a centralised system, control in terms of the three things we talked about. It must reduce, if not eliminate the awareness of hardware idiosyncrasies. I submit that the fact that data is coming from a card-reader, or a tape, or a disk file is no business of the application programmer. Not only that, he should not be able to say that. It is important to take away the idiosyncrasies of hardware from the application languages. We must relieve the application programmer from the responsibility of physical data placement. I also submit that there is no difference between on-line environment and batch environment. We people in the computer industry, when we feel like it, invent a new term. The tragedy is not that we invent them, the tragedy is that we successfully market them! So, I submit that a data management system should not distinguish between on-line and batch environment. I submit that a card-reader is a read-only memory, that a lineprinter is a write-only memory, that a tape-drive is a read-and-write-painfully memory. But one can describe memory characteristics, and one can describe the characteristics of data that reside on these memory characteristics. The functions of a data management system should be to eliminate from application programming these fundamental idiosyncrasies that have been inadvertently introduced into the conventional way of programming or developing applications.

As soon as we say that you must have centralised system control, it immediately implies that you must have an organisational function called a "systems manager". He owns the hardware and software system. Just as he decides how many tape drives he is going to buy, and how many bytes of disk file storage he is going to buy, he should specify what data is under control and what data does not need to be maintained under control. Large organisations in the United States have taken this approach. People who are in the business of making steel, making food, making paint, banks etc., have gone to the approach of centralised functions to do these kinds of things. A systems manager primarily, therefore, requires a data definition language. This allows him to describe data in logical terms only. He must have a way of specifying the structure of that data. Because we cannot specify the difference between data and programs he must have the ability to use them interchangeably and therefore specify the relationships between data and programs. Because of the fact that in the final analysis, the throughput of the system depends on the physical mapping characteristics of that system, the systems manager must have control over the physical mapping strategy that is used.

Sufficient has been said at this conference about security. I am a poor technician; I am not a sociologist; I have no way of gauging the social impact of not being able to, or being able to secure data. I would like to point out a couple of things in regard to security though. I would be the first one to resist being assigned a number (in spite of the that people think I have a funny name and I would probably be able to live with a number more easily!). The fact

does remain that in just the same way that we need to protect the right of the individual, in terms of the information about him, we, as a society, need to look at the cost of not having information about an individual available to the system. So, security problems fall into two categories. The first is our hierarchal set of securities and the other is specifiable by an algorithm - an algorithm that is either hard or soft.

Finally, the systems manager requires some sort of an error correction facility, whether audit trail, recovery or retrials, or whatever mechanism is used. In addition to all this, (since the performance of the system cannot be measured by simulation, the primary reason being that before you can establish the performance of the system in terms of simulation you must have a way of being able to specify the capacity of the system in terms of the work it can perform, and the work it can perform in a large data handling situation must depend in some way on the characteristics of the data) the systems manager, because he must have the ability to correct his mistakes, requires facilities to evaluate the characteristics of data before and after he constructs the data base.

The data definition language must be independent of devices. I sumbit that there are only four types of data. There are the numbers because all of us understand those - that means that there is a defined semantics that we understand about them. There are letters, or "alpha" as they are conventionally known; there are the YES and NO types of functions, and because we are dealing with binary systems, you have the idea of a field. I do not think that it is relevant or important that a number be represented in a 4-bit pattern, numeric form, from the data description point of view. As a matter of fact, you should not allow the person specifying the data description to be able to tell you. The system in most cases can probably figure out, as intelligently as the application programmer can, what is the best form of physically mapping the data. I submit that there is one more form of data and that is "derived data"; the type of data where by naming it, an algorithm gets invoked, so that if you are to maintain TOTAL COST and if the application programmer from his point of view says TOTAL COST, you should be able to say that a procedure called CALCULATE TOTAL COST is tied to this reference, and every time he touches it you derive the value for it. This allows you to extend the idea of data description and it allows you to specify user-defined classes of data. The last characteristic that one would want in a data definition language is the idea of conditional data - data that exists if a certain algorithm is satisfied.

A very important characteristic of this data definition language is that it should be concise. We have seen the trauma that we go through because we proliferate statement beginners such as in COBOL. I submit that a very close theoretical basis for data definition does exist. There has been considerable work done in terms of set theory for a variety of purposes. The data definition language that one designs probably should have some direct correlation. I would think that one could represent data in terms of the contents of a set and the properties of the set. The properties of the set would be things like access authorisiation (who can enquire into that data base, update it, append it, delete it and so on). The other property may be the aliases in the sense that X is also known as Y. There are a series of properties of this type; one could specify a fair number of properties and one could specify some very clever ways of describing contents.

Such a data definition system would provide for a description of data that is independent of the physical mapping strategy that is being used. Because of the fact that the name and address file, when represented in a 6-bit mode, is cheaper than when it is represented in an 8-bit mode, the systems manager should have the facility to specify the physical mapping characteristic. I define that as being the "systems control language". The systems control language should be completely separated from the data description. It probably should be such that you can change it. If, in the simplest implementation, one would allow "canned" data structures to be specified in a system control language, one would allow "canned" physical mapping schemes to be allowed in a systems control language. One would also do something about data compaction in the systems control area. The user interface to such a data related or a data management system can be broken down in a practical world in two forms; the standard languages such as ALGOL, COBOL, FORTRAN, PL/I, and the ones that we have not designed yet. These languages primarily require eight functions. They should be able to open sets of data, they should be able to close them, they should be able to find items of data in them, they should be able to delete, modify, store, create items of data, or resolve all contention amongst themselves. The direct interface may be through a variety of problem-oriented languages or through various enquiry systems.

I would like to comment at this point about software standards in this area. I have already described to you my ideas about building a payroll machine. I feel that software standards in terms of syntax are to take an imperfect solution and to cast it in concrete. I submit that we do not need syntactic software standards. We need automated ways of maintaining compatability, and if we take away all the idiosyncrasies of data from the programmer, then we can write translator systems that can go across hardware boundaries without having to worry about the problems caused by the idiosyncrasies of data representation.

All these thoughts have some significant impact on the suppliers. It is going to increase our development costs phenomenally. It is going to increase our training costs significantly. The support costs are already so high that these types of systems can only make those worse, but the greatest impact in this area will be in terms of the hardware architecture. To be implemented properly, the systems of this type will require greater parallelism in operation. They will require not greater raw speeds but a better balance between the compute capacity and the I/O capacity of the system. There will be a greater need for softer machines; machines that are not cast in concrete too soon. The impact on the user, I hope, will be that of reduced programming costs, but at the same time I would like to suggest that having done this, it will probably increase the volume of data that the user attempts to control. It will also imply a change in the organisational structures; the whole idea being a gearing up for the systems manager type of function. I do feel that there will be a better utilisation of computers using these kinds of systems.

Finally I would like to tell you a joke which I heard some time back, and it is very appropriate in this area. There is a masochist and a sadist marooned on an island, and the masochist suddenly looked at the sadist and said, "Please, please kick me!" and the sadist looked at him for a second and said, "No!". Whether between suppliers and users - (laughter) - enough said!

DISCUSSION

Mr. P.W. Harpham (chairman)

I think for anybody who is interested in system architecture, we have had a magnificent example of one man putting into his head an enormous range of problems and ideas, and coming out with some concepts which we could really build on. Mr Bedekar would be happy to answer your questions

* * *

Question by A.J. Matthews, N.Z. Aluminium Smelters

Taking into account current software trends, how far off are we from having such a data management service?

Mr. S. Bedekar

Between the manufacturers and independent software houses, there have been enough attempts made in this area. The primary complaints against systems of this type have been the high "overhead". Primary problems have been the support to be provided to the customer, so I do not think that in today's environment it is impossible. In fact, I feel that it is almost a mandatory requirem nt to build these systems. To answer your question more specifically, it is quite practical in today's environment to build these systems and make them work.

* * *

Question by B.G. Cox, University of Otago

I wonder if we are going to meet the same problems we met when we went from machine language to higher level languages, in terms of resistence in saying that it is not efficient enough. Can these data base handling services be efficient?

Mr. S. Bedekar

Efficiency is a peculiar thing. If you find a solution in software (in conventional terms) then that portion of it that is not efficient can always be hardware. I will give you an example in my experience. On the 5000 the memory allocation scheme was implemented by using a series of procedures. These were ESPOL procedures but gave memory allocation, and when the 5500 was designed we realised that the problem was that the memory allocation was too slow. The 5500 memory allocation operators are a direct hardening of those procedures, so that efficiency is the problem of hardening enough of the system to maintain certain throughput levels.

* * *

Question by P.W. Harpham

Have you any special thoughts in terms of this control language? You said you thought we would use set theory. Does this mean you think we will use Iverson's notation or something of this sort?

Mr. S. Bedekar

I do not know if Iverson's notation was really derived from set theory; I am not sure that it was. But I think that regardless of what notation is used, if the characteristics of the media that we talked about are maintained, then it would be easy to write a translator that goes from one notational form to the other, essentially maintaining compatibility of description. We have already written translators that take one notational form and convert them to another.

* * *

Question by M. Martin, University of Otago

May I ask Mr Bedekar if in writing his FORTE he was satisfied with the facilities which COBOL offered as a target language, and if not, what sort of things would he have liked to have seen instead?

Mr. S. Bedekar

If you are asking me if COBOL should be extended to account for this, I feel that COBOL is a syntactic form like many others and I fell that the problem of data management goes across the boundaries of a given syntactic form, so that the extensions to COBOL should be in the spirit of COBOL to allow for the eight basic data management related functions that need to be provided. But I do think that we should stop people using the COBOL data division in the context of the data description.

* * *

Mr. P.W. Harpham (chairman)

I do not think I need to thank Mr Bedekar any more - you have already done so. We are very grateful to you, Mr Bedekar, for coming all this way to speak to us, and I for one feel tremendously excited about what you have said. Thank you Mr Bedekar.