Computer Conservation Society

Aims and objectives

The Computer Conservation Society (CCS) is a co-operative venture between the British Computer Society and the Science Museum of London.

The CCS was constituted in September 1989 as a Specialist Group of the British Computer Society (BCS). It thus is covered by the Royal Charter and charitable status of the BCS.

The aims of the CCS are to

- ◇ Promote the conservation of historic computers and to identify existing computers which may need to be archived in the future
- ◇ Develop awareness of the importance of historic computers
- ◇ Encourage research on historic computers and their impact on society

Membership is open to anyone interested in computer conservation and the history of computing.

The CCS is funded and supported by a grant from the BCS, fees from corporate membership, donations, and by the free use of Science Museum facilities. Membership is free but some charges may be made for publications and attendance at seminars and conferences.

There are a number of active Working Parties on specific computer restorations and early computer technologies and software. Younger people are especially encouraged to take part in order to achieve skills transfer.

The corporate members who are supporting the Society are Bull HN Information Systems, Digital Equipment, ICL, Unisys and Vaughan Systems.

Resurrection

The Bulletin of the Computer Conservation Society

ISSN 0958 - 7403

Number 13

Autumn 1995

Contents

Editorial

Nicholas Enticknap, Editor

Progress in establishing a new working relationship with the Science Museum remains slow, though we hope to report positive developments in our next issue. In the meantime, the London-based working party activity remains in a state of suspended animation, so there is no Working Party report section in this issue.

The Pegasus Working Party has put in two sessions during the summer. One lasted a whole day and was devoted to locating and eliminating faults, while subsequently a further few hours were spent working on the package tester.

But though one of our mainstream activities is severely constrained at present, the other is flourishing. Events in London and Manchester in the first half of the year provided members with first hand information about the Leo and Deuce computers, and we hope to bring you edited version of these talks in future issues.

The meetings programme for the rest of the year offers something for everyone in both London and Manchester. Two events are planned in each location for the autumn session, while the North West Group has already arranged its first two 1996 events — details of all these can be found in the Forthcoming Events section on page 28. The London meetings committee is also working on two events for 1996, one on Atlas and the other on the ICT/ICL 1900 series. Planning for both is still in the early stages.

Our feature articles this time include a provocative piece by Gordon Scarrott on the nature of information engineering, which calls into question many commonly held attitudes and beliefs. It provides the philosophical framework for the author's research activity in Ferranti, ICT and ICL described in the last issue.

The other articles are both taken from the thriving North West Group meetings programme, and describe two of the most seminal developments in the history of computing. Dick Grimsdale talks about his experiences developing the first transistor-based computer at Manchester University, while David Howarth recalls the thinking involved in the development of the world's first operating system for the Ferranti Atlas.

News Round-Up

Chris Burton reports that the SSEM (Small Scale Experimental Machine) project is progressing well. This project, which is being run as a collaborative effort between the North West Group of the Society and Manchester University and is managed by Chris, has the objective of creating a replica of the Manchester University testbed computer of 1948. This was the world's first computer to run a stored program, and led to the development of the Manchester Mark I and later the Ferranti Mark I. The intention is to complete the build in time to run it on the golden jubilee of that first run, on 21 June 1998.

- 101010101 -

The revival of the meetings programme in London has proved very successful. The attendance at Gordon Scarrott's talk in March numbered 35, while the Leo seminar in May attracted no less than 52 people.

- 101010101 -

We hope that *Resurrection* will soon be available electronically. Progress is being made on arranging an FTP site to act as a repository for Society material such as simulators and the text of this publication. Watch this space.

- 101010101 -

We lost an important link with the dawn of the computer age in June when Presper Eckert died at the age of 76. Eckert was the co-inventor, with John Mauchly, of Eniac, a computer which executed its first program in 1945. Eniac is believed by many to have been the first digital electronic computer, though others argue that the Bletchley Park Colossus machines have a prior claim.

- 101010101 -

Doron Swade asks us to point out that his article in issue 12 should have been titled "The Phillips Economics Computer" and not as printed.

The Transition From Valves To Transistors

RL Grimsdale

This article covers the author's recollections of the exciting period when the arrival of the transistor was opening up new possibilities in computing, and particularly the developments at Manchester University. The author has employed the terminology used at the time throughout.

In 1950 when I graduated in Electrical Engineering at Manchester, FC Williams and Tom Kilburn had already demonstrated a working prototype of an electronic digital computer. I became a research student and was sent to Cambridge to attend the first Summer School in programming the EDSAC I. That was my first experience of computers.

On my return Tom asked me to write test programs (spelled *programmes* in those days) for the Mark I computer which was then being installed by Ferranti in its own new building on the university campus. At first the test programs frequently reported faults, but after a time it seemed that the machine had learnt "to do" them. In reality inherent errors in the machine, mainly caused by incorrect wiring, were being corrected as I went along by the commissioning engineers.

There were virtually no circuit diagrams then, so the test programs proved to be useful for locating the various parts of the machine — you just unplugged a valve and observed where the program reported an error. The Mark I had 4000 valves, used cathode ray tube storage and consumed 27 kilowatts.

About one year later I was about to build a small valve-based computer when transistors started to become available in the UK. This presented an excellent opportunity to investigate the possibility of building a transistor-based computer.

The point contact transistor had been announced by Bardeen and Brattain in 1948, but there had been reports of attempts to control the current flow in a crystal and cat's whisker before the war. There was some limited success, but the polycrystalline materials used, like galena, were not very suitable.

I obtained my first samples of point contact transistors early in 1953. These were the LS737 crystal triodes manufactured in Somerset by STC. I was fortunate in obtaining a good supply of them — perhaps 80% of the

total output. They were supplied in boxes of 10, and I can remember being very pleased when I received 70 in one day!

There were other sources: the GET2 from GEC and the OC51[1] from Mullards. It was necessary to test the transistors on arrival: in the earlier batches, up to half did not work at all, while many of those that did work had very varied characteristics.

The point contact transistor has a special property. In those days we referred to the alpha current gain, which was the ratio of I_C (collector current) to I_E (emitter current). The emitter was used as the input electrode and the collector as the output. The third electrode was the base, which was connected to +2 volts through a resistance.

Raising the emitter above the base turned on the transistor: assuming a typical alpha value[2] of 3, with 1 mA injected into the emitter, the resulting current into the base was 2mA. This current flowing in the base resistance caused the base to fall with respect to the emitter, and the cumulative positive feedback effect turned the transistor on hard. The collector accordingly rose from its negative off value nearly to ground. A single transistor could thus operate as a two-state device.

Unfortunately transistors could be rather reluctant to turn off, as they exhibited charge storage, and it was necessary to clear the charge to make them turn off. The turn on could be accelerated by increasing the emitter current, but this had an adverse effect on the turn off time, so a compromise was necessary. Typically a transistor would turn on in just under one microsecond and turn off in two microseconds. The clock frequency was accordingly chosen as 125 kilocycles per second.

Little was known about how the transistor actually worked, but it was thought that there was a multiple junction. The LS737 transistor was constructed from a small piece of single crystal n-type germanium mounted on a metal contact forming the base and two wire electrodes (emitter and collector) touching the surface.

To make this work as a transistor it was necessary to "form" it, by charging a 0.1 microfarad condenser to about 20 volts, and discharging it between the collector and base. Germanium was obtained from flue dust and there was some concern at the time that, should these new-fangled devices catch on, there would be a scarcity.

[1]The code numbering of transistors followed that used for thermionic valves, with O denoting no filament and C denoting three electrodes.

[2]Junction transistors have an alpha of just less than unity.

An important consideration in the design of the transistor computer was the choice of memory. The Mark I used cathode ray tube memory, but this did not seem suitable for a transistor computer, because the high voltages involved would be hazardous to transistors and also because of its large physical size.

I managed to acquire a magnetic drum which had been manufactured by Ferranti. This took the form of a bronze wheel 11.5" in diameter rotating on a vertical axis at just under 2500 rpm. The wheel was 4" high, and it had a nickel plated recording surface. It was a precision device with an eccentricity of about a thousandth of an inch, which resulted in a two-to-one amplitude variation in the read-out signal.

The head was positioned with a screw adjustment. The head was wound in until a "ping" was heard and then withdrawn slightly. The major timing waveforms were derived from tracks on the drum. The clock was produced by a track with 3072 pulses around the circumference. This corresponded to 64 words of 48 bits.

Creating the clock track in the first instance was quite tricky. A single pulse was recorded by discharging a capacitor through the head when the drum was stationary. Then, with the drum rotating, an oscillator was tuned to produce a signal corresponding to 64 pulses per revolution.

This waveform was then divided by 64, and the resulting waveform was locked to the output from the track with the single pulse per revolution. It was then possible to record a track with 64 pulses per revolution, and the process was repeated until the clock track with 3072 pulses was obtained.

Like the Mark I computer, the transistor computer used serial arithmetic for reasons of economy. Several working registers were required, and whereas transistor shift registers could have been used, the cost would have been prohibitive. So it was decided that regenerative tracks would be used for the registers.

A read head was mounted a short distance from a write head, in the direction of motion. The output from the read head was fed to the write head through appropriate gating circuits to form the regenerative track. For a single register the spacing was only about half an inch, but negligible head-to-head feedback was experienced.

The problem with the magnetic drum store was access time which was, on average, half the time of revolution. A one-plus-one address code was used, with each instruction containing an operand address and the address of the next instruction. By careful placement of operands and instructions

around the circumference, it was possible to reduce the average access time.

The first operational computer had 92 transistors, and executed its first program in November 1953. It has since been recognised as the first transistor computer[3]. About six point-contact diodes were used with each transistor.

Shortly afterwards the machine was extended with a multiplier and eight B-registers (or index registers). This machine had 250 transistors and 1357 point contact diodes, and consumed 150 watts. The operation time for the division subroutine was about a second, and that for square root 1.3 seconds. The mean time between failures was about 90 minutes: these were almost always due to memory problems. The transistor machine was comparatively small, and was constructed with tag strips mounted on a Post Office rack, in contrast to the Mark I computer which occupied a large room.

In constructing the transistor computer I was ably supported by Doug Webb, a Canadian and a Bugatti enthusiast, whom I was very pleased to meet again in 1992. Ben Delaney, a technician, did much of the construction.

Two young engineers, John Bailey and Peter Cloot, came to work with me from Metropolitan-Vickers Electrical Company. This proved to be a very effective form of academic-industrial collaboration as John Bailey later returned to Metrovick to construct the MV950, a commercial version of the transistor computer.

The company produced six of these machines for its internal use, and they were employed extensively in engineering and research departments for everyday design calculations. The machine was a close replica of the university machine and also used a drum store. It is believed the MV950 was the first transistor computer to become commercially available.

Metrovick merged with BTH (British Thomson-Houston) and other companies to form Associated Electrical Industries (AEI). I was retained as a consultant and helped with the design of the 1010 computer. John Gladman was the Chief Engineer of the Computer Department and was supported by John Bailey and Ron Foulkes.

The 1010 was quite an impressive machine for its time and was one of the first to be used for what was called Data Processing. The company

[3]By Simon Lavington in his book "Early British Computers", published by the Manchester University Press.

secured an order for a twin 1010 installation for stores management at RAF Hendon, a major system for its time.

I joined AEI in 1960, in the first instance to develop transistor versions of electromechanical instrumentation, but after one year I became one of the founder members of AEI Automation Ltd. This company was set up to sell computer systems for industrial monitoring and control applications, considered to be a growth industry. We based our systems on the 1040, an industrial version of the 1010 with the incorporation of an interrupt. Ferranti was also very active and successful in this field with the Argus computer.

One of the most interesting and successful installations we supplied was the Alarm Analysis System for Oldbury nuclear power station near Bristol. It had two reactors each with about 1500 alarm annunciators — little lights on the wall. When a fault occurred, 100 or more of these would light and a bell would ring. The operator had to react quickly to determine what had caused the fault, and if the problem could not be resolved he would be obliged to shut down the reactor, which was very expensive.

The Alarm Analysis System was an expert system which determined the relationship between the alarms that were lit and established the prime cause, the extent of the problem and any dangerous conditions. It also advised the operator on the best course of action. The results were displayed on a cathode ray tube system designed by Harold Hankins.

The contract specified that spares should be available for 30 years, a period that has only just come to an end. The system is of course no longer in use, but it did give valuable service over many years.

Returning to developments at Manchester University, reasonably fast junction transistors had become available by the late fifties, and these formed the basis of the Atlas computer. This machine was an outstanding technical success. It had many innovative features including paging and a fast carry propagate adder. This latter used the junction transistor as a switch, exploiting the low impedance between the emitter and collector of a saturated transistor.

The late Keith Bowden and I had the job of developing the read only memory for the Atlas. This made use of transformer coupling, with the primary formed by a pair of wires connected as a terminated transmission line and driven by a transistor. The secondary was a loop of wire feeding into a transistor amplifier. To store a "one", the primary and secondary were coupled by a small ferrite rod approximately 1mm in diameter and

6mm long. A copper rod was substituted to store a "zero".

The store was constructed using a mesh, about 4 feet by 8 feet, woven from enamelled covered wire. The mesh was mounted on two sides of a paxolin sheet covered with plasticine. For every digital cell a return magnetic path was formed using another ferrite rod. A small jig was used to load the ferrite rods into the mesh, and this required that the mesh had to be uniform. The mesh was made by a wire weaving company in Warrington, and early samples exhibited some non-uniformities which were caused by tea breaks! The memory had a capacity of 8K words of 52 bits and an access time of 100 nanoseconds.

Editor's note: this is an edited version of the talk given by the author to the North West Group of the Society at the Museum of Science and Industry, Manchester, on 13 December 1994. RL Grimsdale is Professor of Electronic Engineering, University of Sussex.

Editorial fax number

Readers wishing to contact the Editor may do so by fax, on 0181-715 0484.

Information Engineering: Exactly Where Are We?

Gordon G Scarrott

This article provides a philosophical backcloth to the research projects described by the author in the last issue of *Resurrection*. It considers the precise nature of computers and those who work with them in the contexts of science, engineering and technology generally. This line of thought leads to the fundamental conclusion that there is as yet no science of information, and this is the major reason for the problem known as the Software Crisis. The author advances some ideas of his own as a starting point for the development of a scientific basis for computing.

A classic foundation for the formulation of military operational plans is that they must be explicitly based on an appreciation in depth of the current situation. This principle applies equally to the formulation of business plans.

There are now many business and academic organisations whose operations are concerned with the design, supply and use of computers. This article is intended to offer an appreciation in historical depth of the current situation in their field of operations. The emphasis on historical depth is necessary to take into account the common experience that those who ignore history are doomed to repeat it.

It is first necessary to define the proper meaning of the familiar but overworked words *Science* and *Engineering*, in order to clarify the meaning of the perhaps unfamiliar phrase *Information Engineering*.

Science is a coherent collection of concepts and relationships that enable us to understand the structures and operations of some, but not all, of observed nature. Scientific research is driven by curiosity regarding aspects of nature that we do not yet understand.

Sometimes, but not necessarily, the understanding derived from scientists' efforts is eventually found to be useful. Even when this occurs, the full extent of the utility cannot necessarily be foreseen when the scientific work is undertaken.

For example Faraday was not trying to invent power stations when he formulated his laws. He was simply trying to understand whether and how the observed phenomena of electricity and magnetism might be related. When he succeeded so splendidly, others whom we recognise as engineers

devised practicable dynamos and electric motors based on Faraday's laws.

An *Engine* is any ingenious and useful artefact. This definition would include a Roman catapult regarded as a siege engine, so that it is much more general than the conventional use of the word to refer to the power unit of a motor vehicle.

An *Engineer* is an ingenious fellow who justifies his existence by devising, building or operating an engine to serve his fellow citizens. He expects to be paid for his services, so engineering is a business, unlike science which has long been recognised by the older universities to be one of the arts.

This clear distinction refers to the human activities known as Science and Engineering, but not to the people engaged in them. It is quite normal for a scientist to stimulate his curiosity by studying real problems in society and conversely for an engineer to allow his curiosity to be aroused when he encounters an aspect of nature for which science has not yet formulated the laws.

When the first computers were devised they were mostly used to carry out arithmetic operations on numbers, so it was appropriate that they should have been known as *Computers*. The situation has now changed profoundly, and current machines known as computers spend more of their time operating on words than doing arithmetic operations on numbers.

I suggest, therefore, that we should refer to present day machines as *Information Engines*, a term that includes the software and correctly implies that the essential common purpose for which information engines are used is to help people to handle information in its full general sense. Consequently those engaged in the supply and operation of information engines should be known as *Information Engineers*.

Every engineering product can usefully be regarded as a synthesis of Ends and Means. The engineer starts by formulating an understanding of the ends he is trying to achieve, stated in terms that are naturally meaningful to the ultimate user. He then selects the means to achieve these ends in the most cost-effective way. The ultimate user does not concern himself with the detailed choice of technological means, but he is very aware of their commercial consequences, such as effectiveness, cost, reliability and availability.

The catalogue of available means is conventionally known as *Technology*, so technology is essentially an engineering term, and technological development must therefore necessarily be justified by foreseeable utility. It is unfortunate that politicians and journalists so often bracket together

the two words Science and Technology, which refer to contrasting activities, and use them together as if they were a single word.

For example there is a Commons Select Committee on Science and Technology, and also a Government organisation known as OST, for Office of Science and Technology. It is difficult to see how OST can serve a useful purpose because if its members devise a policy to foster Technology it would kill Science, and vice versa!

Most technology is founded on laws of nature that have already been revealed by scientific endeavour. However sometimes engineers in their pursuit of marketable utility stumble over aspects of nature for which science has not yet formulated the laws.

For example the first steam engines were devised and demonstrated at the beginning of the eighteenth century, but the science of thermodynamics was not formulated until the middle of the nineteenth. So for 150 years the technology of steam engines was based on experience only, without any illumination of design objectives from science. When eventually thermodynamics was sorted out the design of heat engines made rapid progress.

There is an important distinction between academic and industrial research. The purpose of academic research is to push out the frontiers of our understanding of nature, so that scientific endeavour is motivated by curiosity, supported by patronage in its classic sense of investment without a foreseeable return, and controlled by committees of scientific peers.

The ultimate function of industrial research, however, is to help the company that supports it to survive in a competitive world. Few commercial managers would be willing to support research projects whose only justification is the satisfaction of curiosity.

However they are well aware that market requirements evolve by a Darwinian process of mutation and natural selection in the market place, and that their company should cultivate an awareness of foreseeable changes in market requirements and be prepared to survive such changes and perhaps take commercial advantage from them.

Consequently a proper function of industrial research is to identify customer needs that have not yet been recognised or met in the market place and explore ways of meeting such needs. This function is quite distinct from the process of industrial development which usually concentrates on tactical improvements to established products. So the activities generally referred to as Industrial Research and Industrial Development require

different personnel, communications, and management styles.

It is unfortunate that so many business men in engineering companies whose training was primarily in accountancy have failed to recognise the functional distinction between industrial research and industrial development. They refer to both these activities by the conventional term R&D, used as a single word.

The commercial consequence of this folly is that business managers in engineering too often give inadequate attention to innovation. Another consequence of these definitions is that science is concerned with natural objects with the object of formulating the laws of nature. A computer however is an artefact, not a natural object, so the conventional phrase *Computer Science* is, strictly, a contradiction in terms!

The practice of every branch of engineering is illuminated by an understanding of the relevant laws of nature. For example bridges seldom collapse essentially because there is a well established science of the strength of materials, but we do not call it Bridge Science. The common use of the term Computer Science demonstrates that those who use the term have not understood the relationship between Science and Engineering.

Upon considering my own career in industrial research, my main conclusion is that the example of the history of heat engines — that engineers designed them without the benefit of any illumination of their objectives from any science of heat — has been repeated in our own time in the story of information engines. They too have been introduced and evolved to their present stage without any science of information.

However we have good reasons for expecting that we will not this time have to wait 150 years for the situation to be resolved. Already in July 1994 an international meeting of scientists from a wide range of disciplines was held in Madrid with the explicit objective of formulating foundations of information science.

To draw attention to the situation I have written a paper, which was published in the *IEE Journal of Science and Education* (Aug 1994 Vol 2 Number 4). It was then reprinted in the *Journal of Information Science* (Vol 20 Number 2 1994) — so I have reason to believe that at least one person has read my paper and found it to be of interest!

This is not the place to repeat the arguments outlined there in full, but I will try to put the essence of the matter in a few sentences.

I am sure readers will be well aware of the essential distinction between a System and a Subsystem. When we formulate the information exchange

conventions between them we make the subsystem fit the system, not vice versa. We all know that if we try to match the system to the subsystem we run the risk of sabotaging all the other existing subsystems and all hell will be let loose.

In established practice we regard the computer as a system and peripherals and communications as subsystems. Accordingly, we assume that we have complete freedom to shape the computer design as we please, usually to match the latest technological fashion.

If however we start by considering the role of information in human affairs we are forced to the conclusion that human society is essentially an information system that has evolved by a Darwinian natural selection process for millions of years. A computer should be recognised as a subsystem to this existing human information system.

Some natural properties of the human social information system can be observed and understood, but cannot be changed. I conclude that the widely recognised unsatisfactory state of the software development process can be attributed to the fact that established system design practice has been evolved mainly by tactical considerations, ignoring natural properties of information derived from its role in human affairs. Consequently we have sabotaged many existing subsystems in the human social information system and all hell has been let loose.

If we endeavour to formulate an understanding of the place of mankind in the evolution of life on earth we can immediately observe that mankind is certainly the most successful species. Like many other evolutionary successes mankind is a specialist, but it is far from obvious what is our speciality.

When Linnaeus formulated his classification system his choice of the term *Homo Sapiens* implied that our human speciality was wisdom, an assumption that does not accord with most recorded human history. I suggest that the unique achievement of mankind is to have evolved natural language for rapid communication of concepts of far deeper subtlety than the grunts of other creatures.

Evidently the use of natural language must have conferred decisive survival value since otherwise it could not have survived natural selection. Most probably that survival value arises from practising our human speciality, which can be described most concisely as dynamically adaptive social organisation — for which communication by speech is obviously essential.

Consequently I suggest that an appropriate term for the place of mankind in life on earth would be *Simia Ordinans* — the organising ape. It can certainly be observed that mankind can organise almost anything by using his information handling skills, although his choice of what to organise is not always good!

With this view of our human speciality it follows that before embarking on a computer system design we should seek to understand observable features of the way that we construct meaningful messages in natural language to operate our dynamically adaptive social organisation. An observable statistical feature of natural language text, known as Zipf's law, has been known for nearly a hundred years but its significance for systems engineering has been overlooked.

The only credible explanation of Zipf's law — that the distribution of the usage of meaningful symbols (words) in every natural language is hyperbolic — is that meaningful text is constructed by a recursively defined process that has no sense of scale. In other words, meaningful text is an interdependent sequence of words and/or meaningful texts.

This formal definition is a consequence of the way that people instinctively handle information, since it offers a way to deal with arbitrarily complex matters by dividing them into manageable parts so that it is not necessary to consider a complex matter all at once.

We also use the recursively defined technique when we create a human organisation, or write a program for a computer. Many of the hazards of program development arise when a program A, describing an aspect of the overall problem that is currently being considered by the programmer, initiates another program, B, referring to a distinct aspect of the overall problem that arises later in the mind of the programmer.

When this occurs many of the hazards arise from store allocation, since for A to communicate with B it is necessary for some storage addresses to be accessible by both programs so that they can communicate with one another, while other space should be accessible by B only, so that A cannot accidentally sabotage B.

John Iliffe, my former colleague at Ferranti, showed in the sixties that these requirements can be achieved provided that the system design is shaped so that the structure of the information can be explicitly represented in the hardware of the system. His proposition was valid 30 years ago and it is still valid.

The Software Crisis has already been widely recognised but there are few credible proposals on the table to solve the problem. There is still a very long way to go to develop a valid and credible understanding of the natural properties of a meaningful set of symbols such as a computer program, but with the benefit of our present understanding of such matters we can recognise that in the evolution of computer system designs two fundamental errors occurred.

The first must be attributed to von Neumann, who made no attempt to represent the organisational structure of data and processes. We should not criticise von Neumann for this oversight so early in the history of information engineering, but perhaps we could wish that two generations of system designers had thought more carefully about their design objectives before following von Neumann's example.

The second error occurred when time sharing systems became fashionable and all efforts were put in to ensure intraprogram protection, while the equally important need for interprogram protection was ignored.

Perhaps those systems engineers who are trying to banish the software crisis should reconsider Iliffe's techniques and start by abandoning these hallowed practices.

If there is anyone reading this who is active in the academic world and who recognises that there is a need to formulate a science of information, they may be interested in some questions raised in my paper that I could not answer but, if answered, might help to progress the formulation of a science of information.

For example I pointed out that in the classic paper by Shannon and Weaver (*The Mathematical Theory of Communication*), the well established measure of the communicating power of a set of symbols, measured in bits, is an entirely valid measure of communicating power as Shannon intended, but it should not have been termed Information, since it did not take into account the natural recursively defined structure of a meaningful set of symbols.

I therefore proposed the term *Useable Entropy* (UE) to take into account the structure of meaningful text, quite distinct from *Communicable Entropy* (CE) as defined by Shannon. I gave some obvious reasons why UE is much less than CE but I could not propose a mathematical relationship between UE and CE.

However it ought to be possible to shed light on the question by experimental determination of the slope of frequency/rank plots for n word

groups, and counting the total number of n word groups that exist in a large sample of meaningful text. Evidence obtained in this way might stimulate theoretical analysis to account for the measured results. If by such activities it is possible to establish the relationship between UE and CE it would supersede the notoriously useless figure *cost per bit* as a measure of technological cost-effectiveness of storage techniques.

Editor's note: this is an edited version of part of the talk given by the author to the Society at Birkbeck College, London on 22 March 1995. Gordon Scarrott is a Fellow of the Royal Academy of Engineering.

Internet addresses

Readers of *Resurrection* who wish to contact committee members via electronic mail may do so using the following Internet addresses.
Chris Burton: chris@envex.demon.co.uk
Martin Campbell-Kelly: mck@dcs.warwick.ac.uk
Dan Hayton: Daniel@newcomen.demon.co.uk
Adrian Johnstone: adrian@dcs.rhbnc.ac.uk
Tony Sale: t.sale@qufaro.demon.co.uk
Robin Shirley: r.shirley@surrey.ac.uk
Doron Swade: d.swade@ic.ac.uk

How We Made Atlas Useable

David Howarth

When Manchester University started the design of the Atlas computer in the late fifties, it had become apparent that there should be software within a computer designed both to optimise the use of the machine's resources and to insulate users from the need to initiate routine machine operations.

So Atlas was the first computer to have what we now recognise as an operating system, though the term did not come into use until later, when IBM popularised it. The Atlas operating system was actually called the Supervisor. This article describes the thinking behind its development.

On Atlas, I/O operations were overlapped to save the processor having to wait for devices such as the paper tape reader. So the operation of reading a character consisted simply of starting the reader and then moving on to execute the next instruction (rather than waiting while the reader transmitted the character to a buffer) and then interrupting the processor to say it had done so. Today, every computer works that way. Atlas was the first.

If the Supervisor had merely done that — initiate the Read Character process, then carry on executing other instructions — it would have been hopelessly inadequate. Computer users do not wish to operate at that level of detail. There was a need to provide higher level facilities, such as Read Record and Print File, via standard software which automatically carried out lower level activity such as reading characters.

There were operational considerations as well. Atlas was not a desktop machine: it was a roomful of cabinets, which made it difficult to know exactly how it was running. Operators needed help to tell them what was going on and when to take actions such as loading magnetic tapes. They also needed a log of system activity, so that the users could be charged appropriately.

Finally, there was a need to use the machine's resources effectively, that is to maximise throughput and minimise response times. Not to make sure the processor is running say 80% of the time doing useful work — the literature of the time tended to focus on this metric, but it is not the main objective. My original boss at Ferranti, the late Stanley Gill, brought

this home to me by arguing that you don't apply that principle to a fire extinguisher, so why to a computer?

When I first joined Ferranti, Stanley directed me to work on the compiler compiler (effectively a means of generating a compiler in a hurry) alongside Tony Brooker. While doing this I heard about the Atlas Supervisor, and decided I would rather work on that. Stanley was not keen.

"It might sound interesting, but it is all done in hardware", was his initial response. What he meant was that the hardware played a major part in helping it function, which it did — it was designed to; that proved a stimulus rather than a deterrent. His next objection was that someone (Bruce Payne) was already working on the Supervisor, and he didn't think we needed anybody else. Both responses show how little even experienced computer scientists of the time understood the role that operating systems were to play and the problems of constructing them.

I eventually persuaded Stanley that two people might be usefully employed on what was to become a gigantic project. It was a momentous decision, because I stayed working on operating systems for virtually my entire time with first Ferranti, then ICT, and then ICL.

The two man team of Bruce and myself never grew much larger: our maximum size was seven people. I recall an occasion when Peter Hall, facing questions about the slow progress of development, called a meeting of the team in his office at 1030 one morning. The four of us assembled in this palatial conference room dead on time: we waited, and waited, longing to get back to our desks, when finally Peter came in and said "Where is everybody?". When we told him the entire operating system development team was in front of him, he was shocked. That is how the team got up to seven: it virtually doubled overnight.

Our first task was to evolve an operating strategy for using the computer. People do not suddenly dream up new ideas: they evolve from past experience and history.

In this case, our experience was of a machine room full of gigantic machinery. Users queued to get access to it (or they had a booking system to avoid actually standing in a queue, but this was still a logical queueing system). When it was your turn, you entered the machine room, unpacked your briefcase and settled down to use the machine. When your time was up you had to leave. If you were lucky, you took some results with you, but usually you were unwilling to leave because you had not had enough time, or because you had come across a mistake in your program. So you

had to rejoin the queue.

This process was refined by experience. To reduce the comings and goings, a system was evolved whereby jobs to be run were put in an in-tray: an operator would in due course take the job, enter it to the system, take the results and put them in an out-tray.

This procedure allowed jobs to enter the machine room in batches, rather than one at a time. That in turn increased the time the computer spent processing, because input activity occurred only once per batch, rather than once per job. Both the Ferranti Mark I and the Mercury were run in this way.

Then the Ferranti designers realised that the factor limiting performance was not the number of times material entered the machine room — this can be minimised simply by increasing the number of jobs per batch. The problem was that the input peripherals — punched card and paper tape readers — operated very slowly relative to the processor.

The way to speed up I/O was to avoid use of paper tape or punched cards as much as possible, by relying instead on higher performance input subsystems such as magnetic tape.

IBM had developed what became known as the Fortran Monitor system, which can be regarded as an embryonic operating system implemented in hardware. The operator took the input and entered it to a small satellite computer, typically an IBM 1401, which output the data onto magnetic tape. The tape was then fed into the principal computer, a 709 or 7090. That in turn produced its output onto magnetic tape, which was fed back into the 1401 for final output onto whatever was required — punched cards, a paper tape, or a line printer. This arrangement proved quite popular, at least among scientific users, as it certainly speeded up throughput, although at some cost in response time.

This was the background when we were considering what should be the operational characteristics of Atlas. We had to rule out the possibility of developing a multi-access system, because we didn't have a disc subsystem, or anything like it. (The first disc system, IBM's original Ramac, did not appear until late 1956.) To support multi-access, you need somewhere for storing bulk information: if you simply gave your users character-based terminals, they would spend all day entering their data. So multi-access systems did not arrive until later.

So our thinking focussed on what is now known as a spooling system. "Spool" was an acronym coined by IBM, standing for *S*imultaneous

*P*eripheral *O*peration *O*n-*L*ine, and introduced with the 360 series, but the concept was introduced by Ferranti on Atlas and Orion.

Our concept was to take the type of IBM Fortran Monitor system I have described, incorporate it into one system instead of requiring transfers between two separate computers, and get it to work automatically. To this we added refinements such as software recognition of what a batch meant. With all previous systems the computer had to wait until the operator fed it the contents of an in-tray: if that only happened once a day, then the users had to wait all day for their output.

Building this functionality into software allowed a more flexible interpretation of the batch concept. A job that was input could be run right away (especially if it was a small one), or queued for processing later (especially if it was a long one). Furthermore, you could have more than one job being executed at any one time — the process that became known as multiprogramming.

Some people understood the term multiprogramming to mean that the computer was doing many different jobs at the same time, and that therefore you get more work done. This was not the case. There was only one element of the machine actually executing instructions, and it could only process one instruction at a time.

In practice it was only occasionally you wanted to interleave one job with another, and the level of multiprogramming on Atlas was generally very low — only two or three jobs at once. Many jobs, both long and short, were not limited by waiting for peripheral activity, as they were protected by the spool buffers.

Nonetheless the multiprogramming principle was implemented in the Atlas supervisor. And that in turn created a need for an online scheduler, which decided in what order jobs should be executed.

The spool buffer was one element heavily influenced by past history. Our plan was for a system which executed jobs via a sequence of slow input, fast storage, execute phase, fast storage and slow output. So what should be used for fast storage?

In answering this question our designers felt that the IBM Fortran Monitor system was a good model. It required a lot of batching, but the idea of moving from slow I/O to magnetic tape and back was very attractive. So the initial plan was to have a magnetic tape device at each end of the system plus one for each program being run, with operators transferring tapes from deck to deck as required.

The problem with that was that it required an inordinate number of tape decks. There were only eight on Atlas, so people running large I/O bound commercial jobs could wait a long time for available resources.

Furthermore, if you looked at the machine you could observe that most of the decks were doing nothing most of the time. A deck was capable of operating 20 times faster than a paper tape reader, and if it only ran for 5% of the time this was not very cost-effective.

The next idea was a logical consequence of the design objective that Atlas should be a supersystem — more powerful than anything yet seen, with an extremely fast processor, a vast main store and a vast drum store. The combination of core memory and drum became known as a one-level store, because it could be addressed as if it were one logical unit.

So the idea was to use that one-level store to keep the spool buffers, instead of relying on magnetic tape. The store would logically be divided into two parts, one for the spool buffers and the other for executing programs.

The problem was the size of the main store. The designers were delighted with it, because it was bigger than anything we had ever had. The size was 128 kilobytes (Kb) — even that's an exaggeration, as each "byte" only contained six bits, but call it that. Today mainframes have that number of *megabytes* of main store, and still people want more. We all know the story — software expands to fill the space available. So it quickly became apparent that the store size was not adequate for spool buffers as well as program execution.

What we really needed to do was to extend the main store onto disc. But all we had was magnetic tape.

The magnetic tape system was organised in a very unusual way, rather like disc, in enormous blocks of 4Kb. The tapes were pre-addressed — that is, written before use with a set of fixed length blocks with an interblock gap between. Each block had a start marker, an end marker, and an address.

So you could search for a specific block, bring it into memory, and read it or overwrite it without affecting the next block. With most magnetic tape systems of the time you couldn't do that: you started at the front and wrote consecutive records, just like an audio tape. So our tape was in some ways like a disc, though it was slow and the seek time was enormous.

Since it was like a disc, you could use it like a backing store — search for blocks, read those blocks, wind the tape on or back and write blocks.

Some of my colleagues, such as Ian Pyle who was familiar with the Fortran Monitor system, could not believe this was practicable, but some simple sums prove the point.

Filling one block of tape using a slow input device took about 10 seconds on Atlas. That is a long time, but they were enormous blocks. If you have n such devices, you can set aside n blocks of store for data coming in from them, sequentially, and another n blocks of store dedicated to information waiting to go to tape.

That is not terribly expensive. You can keep the system occupied by writing to tape every 10 seconds. Those n blocks set aside for the slow input devices could only become available every 10 seconds. During that time you can move a lot of tape — over 200 blocks worth.

So the way you could operate was to write blocks to tape every 10 seconds. While the n blocks are being refilled by the input devices, the tape is idle, so you can move to a different part of the tape 100 blocks away, pick up useful information and give it to the execute stage, and then move back in time for the next write operation. Those 100 blocks contained a fair amount of information.

You would still get problems with something exceptional. If you had a job with an input file that occupied thousands of blocks of tape, it would take several minutes to read it all in. During that time everything would be piling up in main memory and there would be chaos. For those situations we decided to go back to the Fortran Monitor system.

For some applications we could give the users a tape, and they could put the data on serially. So we arranged a system for users with really long input streams so that they could trickle the data slowly onto magnetic tape themselves. We would subsequently read it on off the tape at tape speed. This was acceptable so long as everybody wasn't doing it.

The software needed to control these processes interacted with the hardware. The Atlas processor had a lot of advanced features. It was multi-state: you could run it in three different states, called *main*, *extra-code* and *interrupt*. *Main* was employed by users for executing their programs, *extra-code* provided library procedures (such as the instruction for Read Next Record), and *interrupt* handled the low level I/O devices.

Connected with the processor were a variety of stores which could be used differently in the different states. The main store could be accessed by users or by the Supervisor, though in practice the Supervisor used it rarely. This was because there were some parts of the store that were

dedicated to the Supervisor's use.

These included a read only memory, which was large for those days at 64Kb. It was used primarily for the code and the test routines. Then there was the Random Access store, used for data, which could hold the princely total of 8Kb. Finally there was the V-store, for I/O registers.

All of these elements together formed the main store, which was supplemented by the drum, and together these made, as I have said, a one-level store. Today we would describe this by saying that a data item has a virtual address which can be mapped to either the main store or the drum. Only one address, not two, as was the contemporary practice, one for the main store and one for the drum, which meant you had to program transfers between them.

The one-level store is now universal. We did not call it virtual addressing — that was a term invented by IBM. They claim they invented the concept in 1972, but the idea was there in Atlas in 1962.

But if the processor design was very advanced, the I/O interface was in comparison primitive. The software could do all the basic things, but we had not catered for the fact that I/O devices behave peculiarly sometimes. At the Atlas inaugural ceremony at Manchester University in 1961, this produced some embarrassing moments.

We had developed a demonstration to show how the system worked. We had set up several desks, each equipped with its own paper tape reader and punch. Our distinguished visitors were invited to use these to run elementary programs: for example they could type in their name and the machine would recognise it and output it not only onto a local paper tape punch, but also on a teleprinter where they could see it. We had a running commentary that said, for example, "Desk 2 start: Welcome Sir John Cockcroft: desk 2 end", as Sir John was being given the piece of paper tape with his name on.

That was the plan: unfortunately things didn't quite turn out like that. Occasionally something went wrong, and what actually appeared was "Welcome Sir John Cock desk 3 starroft...", which seems funny now but didn't go down too well at the time. This was the first example of what became known as a "Supervisor Incident".

What caused this problem was that occasionally the teleprinter would stop working for no apparent reason. Provision was made in software for restarting it when this occurred, at the point where it had left off. Unfortunately our procedure for doing so was logically flawed, in a way

that was very hard to identify (but very easy to fix once we'd done so).

Supervisor Incidents became a measure of reliability and of progress in software engineering — at any one time, there were up to 90 or so incidents awaiting attention.

How effective was the Supervisor? As far as performance goes, the throughput was initially just tolerable, but it got better as time passed, partly because it influenced the way users and operators behaved. We ended up, even on the system with the smallest main store, with over 70% useful processor time, and on some sites we reached 90%, which was very good by the standards of the sixties and excellent by today's standards.

Reliability was never as good as performance. Initially it was poor, largely because of the Supervisor Incidents I have described. It became tolerable, but was never satisfactory. I don't know whether it was better or worse than other systems of the time: when you're on the receiving end you tend to take a jaundiced view!

I give great credit to all the staff involved — they started off good, and ended up excellent. Anyone who could understand what was happening — who could appreciate, for example, that the teleprinter had stopped when it shouldn't — had to be good. The people who worked on the Atlas project have been exported throughout the world, as scientists, as professors and as systems programmers. So the ideas developed were widely disseminated, and who knows how much that contributed to the computer industry's progress?

Was Atlas a commercial success? Peter Hall has now retired, doubtless with his millions, but I'm very sure he didn't make them out of the Atlas Supervisor. It only ran at three sites, and I suspect Atlas was really a commercial disaster. But I think that one of the important things in life is enjoyment: the kind of enjoyment that comes from intellectual satisfaction, facing a challenge and meeting that challenge. From that perspective Atlas gets full marks.

Editor's note: this is an edited version of the talk given by the author to the North West Group of the Society at the Museum of Science and Industry, Manchester, on 12 June 1994.

Letters to the Editor

Dear Editor,

My father Sid Michaelson, with Tony Brooker and Keith Tocher, built the Imperial College computers ICCE1 and ICCE2 in the 1950s. I would very much appreciate any information as to the final fates of these venerable machines. It is possible that some or all of them are held by the Science Museum or Imperial College.

I would also be delighted to hear from anyone who used them. I have a copy of "Report of the work of the Computer Group, Part I, Department of Mathematics, Imperial College" by KD Tocher dated October 1952, if that is of interest to anyone. There is more information in the Computing section of the "Centenary History of Imperial College", and in my father's paper in Bowden's "Faster than Thought".

Yours sincerely,

Greg Michaelson
Edinburgh
8 June 1995

Editor's note: letters will be forwarded.

Dear Mr Enticknap,

I was reminded, reading Gordon Scarrott's article in issue 12, how disappointed I was — for myself and other users — that ICT did not go ahead and produce Cafs (Content Addressable File Store) in the 1970s.

In 1968 I led a team set up in the Management Services (Computers) Division of HM Treasury — the genesis of CCTA (Central Computing and Telecommunications Authority) — to examine the feasibility and economics of setting up a Civil Service Personnel Information System called Prism.

It was soon apparent that a basic requirement was to be able to extract quickly (ie from a terminal), from the mass of personal data for half a million Civil Servants, small sets of useful data which could not be predetermined, both for statistical and personnel management purposes.

In 1968 I visited Gordon Scarrott and was given a demonstration of

Cafs. I realised at once that it could go a long way towards meeting our information retrieval needs by means of hardware, and I subsequently tried hard to persuade ICT to see how important it could be, not just for Prism but in many business and administrative systems. However, I had no more success than Gordon did from within the company.

Subsequently the systems design for Prism[1] included two software suites: a user's information retrieval language Pirl[2] and a database called Countflow[3]. The latter was a "date sensitive" totally inverted file which could provide change or "flow" information as well as stocks. Peter Bellerby's paper outlined Cafs and looked forward to its adoption during the course of Prims development. He did not believe this would effect Pirl, that is the user interface.

Yours sincerely,

CP Marks
Banstead
Surrey
9 June 1995

Dear Mr Enticknap,

With reference to issue 12, summer 1995 of *Resurrection*, page 8, Harry Johnson is indeed correct that the packages for the prototype Pegasus were made by subcontractors. As I recall they were made by Costain John Brown, a firm that up to then I had regarded as builders, not electronic engineers! But they had some such activity near a London railway terminus — Waterloo, Cannon Street, or perhaps London Bridge.

The fact that I don't recall Racal being involved doesn't mean they weren't.

Yours sincerely,

D Hogg
Munich
Germany
20 July 1995

[1] "Manpower planning in the Civil Service". Civil Service Studies No 3, HMSO 1976. (See Chapter 13, Data Systems & Classification, by CPH Marks.)

[2] "Interrogating Date Sensitive Files" by FE Randall, *Computer Journal* November 1974, pp302- 305.

[3] "Prism Countflow - a date sensitive inverted file" by Group Captain PA Bellerby RAF, a CCA paper for Datafair 1973.

Forthcoming Events

2-3 September 1995 Bletchley Park V-J Day anniversary commemorations

11 October 1995 Half day seminar

The Zebra computer. Four speakers, including the machine's designer Dr van der Poel. Seminar starts at 1400 hrs in the Science Museum. Please note date has changed since the last issue. Organised by George Davis.

17 October 1995 North West Group meeting

J Howlett and A Bagshaw will speak on "Getting Atlas off the ground".

21 November 1995 Whole day seminar

The IBM System/360. Seminar starts at 1100 hrs in the Science Museum. Organised by Chris Hipwell.

28 November 1995 North West Group meeting

Leonard Griffiths will speak on "Computing at Rolls Royce in the Fifties and Sixties".

6 February 1996 North West Group meeting

The Origin and Development of Database Software, by Professor Peter King.

23 April 1996 North West Group meeting

Industrial Research in the Information Technology Field, by Gordon Scarrott.

The North West Group meetings will be held in the Conference Room at the Museum of Science and Industry, Manchester, at 1730. Refreshments are available from 1700.

Resurrection is the bulletin of the Computer Conservation Society and is distributed free to members. Additional copies are £3.00 each, or £10.00 for a subscription covering four issues.

Editor – Nicholas Enticknap
Typesetting – Nicholas Enticknap
Typesetting design – Adrian Johnstone
Cover design – Tony Sale
Transcripts of talks – Pat Woodroffe and Hilma Quinn
Printed by the British Computer Society

Committee of the Society

Chairman **Graham Morris FBCS**, 43 Pewley Hill, Guildford, Surrey GU1 3SW. Tel: 01483 566933

Secretary **Tony Sale FBCS**, 15 Northampton Road, Bromham, Beds MK43 8QB. Tel: 01234 822788.

Treasurer **Dan Hayton**, 31 The High Street, Farnborough Village, Orpington, Kent BR6 7BQ. Tel: 01689 852186

Science Museum representative **Doron Swade CEng MBCS**, Curator of Computing, The Science Museum, Exhibition Road, London SW7 2DD. Tel: 0171-938 8106

Chairman, Elliott 803 Working Party **John Sinclair**, 9 Plummers Lane, Haynes, Bedford MK45 3PL. Tel: 01234 381 403

Chairman, Elliott 401 and Pegasus Working Parties **Chris Burton CEng, FIEE, FBCS**, Wern Ddu Fach, Llansilin, Oswestry, Shropshire SY10 9BN. Tel: 01691 791274

Chairman, S100 bus Working Party **Robin Shirley**, 41 Guildford Park Avenue, Guildford, Surrey GU2 5NL. Tel: 01483 65220

Chairman, North West Group **Professor Frank Sumner FBCS**, Department of Computer Science, University of Manchester, M13 9PL. Tel: 0161 275 6196.

Chairman, Meetings Sub-committee **Christopher Hipwell**, Bretts, The Green, Newick, East Sussex BN8 4LA. Tel: 0182572 2567

Secretary, Meetings Sub-committee **George Davis CEng FBCS**, 4 Digby Place, Croydon CR0 5QR. Tel: 0181-681 7784

Editor, Resurrection **Nicholas Enticknap**, 4 Thornton Court, Grand Drive, Raynes Park SW20 9HJ. Tel: 0181-540 5952 Fax: 0181-715 0484

Archivist **Harold Gearing FBCS**, 14 Craft Way, Steeple Morden, Royston, Herts SG8 0PF. Tel: 01763 852567

Dr Martin Campbell-Kelly, Department of Computer Science, University of Warwick, Coventry CV4 7AL. Tel: 01203 523196

Professor Sandy Douglas CBE FBCS, 9 Woodside Avenue, Walton-on-Thames, Surrey KT12 5LQ. Tel: 01932 224923

Dr Roger Johnson FBCS, 9 Stanhope Way, Riverhead, Sevenoaks, Kent TN13 2DZ. Tel: 0171-631 6388

Dr Adrian Johnstone CEng, MIEE, MBCS, Department of Computer Science, Royal Holloway and Bedford New College, Egham, Surrey TW20 0EX. Tel: 01784 443425

Ewart Willey FBCS, 4 Sebastian Avenue, Shenfield, Brentwood, Essex CM15 8PN. Tel: 01277 210127