7 Long Term Information Sustainment

7.1 The High Level View: What and Why and Why?

For a high-level view, try an upper floor of a tower block. However, a high level view shows only the big and evident. And what is most evident from the 26th floor of Centrepoint - an office tower block in the centre of London's West End - is the layout of the streets. That has changed little since the building went up in 1966, because to change it you would need to demolish hundreds of buildings at the same time. If you go down to street level, you can start to pick out the individual buildings have been demolished and totally rebuilt. And going inside the ones that look original, aside the obvious paint job and new light fittings, you may notice slight irregularity in the plaster, the scars of old walls and windows. Moreover, of the many tenants that have moved in, customised the buildings and then moved on, there is no trace.

On the 26th floor, I went to a vague (early concept) meeting on "keeping computer data usable in the long term". But before we start, we need a better name for the subject:

- Archiving - but comes with connotations of file and forget ✘
- Retention - file and at least remember where the files are kept? ✘
- Sustainment - actively ensuring you have the right software to read the old data ✓
- Curation - a broadly equivalent term preferred by academics and librarians.

Where to start? Let's not start with the hardware - highly visible, but totally indifferent to what goes on inside the box. What we preserve in a library is knowledge, although the form - hardback, paperback, big print - is usually secondary. Even the language is negotiable - I can't read Gödel or Peano in the original, but I can access their knowledge from translations. However, it is not the library's job to understand the books - knowledge is in the brain of the reader.

But the libraries sector is highly influential on the long term sustainment of electronic information. A key work has been the Open Archival Information System (OAIS) [7.1]. Its central idea is that one must not only keep the physical data, but one must also retain the know-how to read that data - that is, retain the mapping from data to knowledge. And since that knowledge may itself be in the form of electronic data, additionally one must maintain the knowledge that tells you how to read that knowledge (and so on). Sustainment involves far more than file and forget.

At this point there is a fork in the road. Fork left, and we are on busy road, much studied in the libraries sector, focused on how people will continue to read documents. This has included devising virtual machines to emulate the machines that originally ran document readers. Fork right - here the LOTAR project [7.2] is highly influential - and there find methods to ensure that modern computers can accurately reconstruct the digital models created by cobwebbed and coffined software. I will mainly follow the right fork. But I will avoid repeating what I wrote in "Model Archiving and Sustainment" [7.3]. Rather, since it is easier to imagine the present than predict the future, this chapter will look backwards to the knowledge we should have kept in order to resurrect old data. In doing so, I will provide clues as to what we must do now in order to read - in fifty or a hundred years time - the information created today.

The biggest barrier to sustaining information is the IT industry itself. If a physical building fell down, investigators would look to the plans to see that the design was adequate, and check the maintenance records. They would not investigate the colour scheme chosen by the last occupant. But in IT, it seems that software companies see themselves more as decorators, supplying the latest

features to attract the user, rather than as architects, providing a solid infrastructure that will house the business as it develops. Weekly software updates, six monthly "major " software releases, and whole systems replaced after only a few years. And this is "supporting" business sectors such as automotive, where the design of a car may be in production for twenty years, or shipbuilding (30 to 40 years), or aerospace (70 years) or nuclear (100+ years)? Yet these are sectors which increasingly rely on software models to demonstrate that the products are safe to use. And, because product liability lasts years after production has ceased, they may need to resurrect them in the dim future, .

7.2 The Road Ahead

In this chapter I will be looking back, picking out examples of historical knowledge that has - or has not - been retained. My aim is to further illuminate the concept of knitwork, rather than provide a practical guide to long term sustainment.

Transcription from one medium to another - from papyrus to vellum to paper to CD - has a history of thousands of years. This tradition is still continued by specialist archiving companies, who use multiple repositories (in case of fire, flood, etc) and multiple technologies (in case one equipment fails). However, although the first step on the road might well be the ability to read old media, this is not a book about holes or magnets or shellac. What we crave is the data on the cards or tapes, or even the authentic voice of the old blues man. And yet...

Our first exercise will be the rendering of abstract data - encoded as binary bits - into something we can read. The decoding step becomes obvious once we look back at an old technology - the punched tape reader (Exercise for the reader: find one in a museum near you). After punched tape I will move on swiftly to document files - blocks of bits on tapes or disc; time to take the old PC out of the attic and reread those documents you (or your parents) wrote in the 1980s. We could investigate modern document formats, but they come with several decades of complications.

Documents are conveyors of information between people, but we are aiming at how information is interpreted by machines. Hence I will leap from documents to CAD models, illustrating how even something as simple as a point in space can be represented in different ways. Then if we take a simple step up from points to curves, we will see that data relies on algorithms to reconstruct the knowledge that the data represents - the lowest level of a knitwork between knowledge and inference.

At this point we need a quick sit down in Provenance cafe, where they serve up useful reminders of where the data came from, and therefore what its value is - locally sourced beef in the home-made steak pie, not ultra-processed foods blended from industrial farms. A homely image for eye-witness reports versus ChatGPT mash-ups (or you can choose this week's IT bete noir).

However, at the end of the road we hit the problem of people. Not only do we need people who remember how old software was used, but also people who understand the knowledge that is recorded via the data. Physical example: my neighbour gave me an old, cast iron, hand-cranked chaff cutter - it took three of us just to slide in down the road. The first question - what exactly is a chaff cutter? At least knowing it was called a "chaff cutter", I could then look it up - it's a machine for cutting straw into short pieces for mixing with fodder. A closer inspection showed that the large wheel had blades fixed to its spokes that brushed past a bar - rotary scissors - function inferred from form. And since 19th century machinery does not come with guards to keep little fingers away from sharp blades, my second question was how do I stop it working? *(Risk inferred from function).

That is, I used reverse engineering to understand what the system did, and ordinary (forward) engineering to work out the practical implications of having such as system. But I am old enough to have played with old machines, and to remember their lacking any health and safety features.

The examples below apply a similar approach to software - what knowledge can we recover from examining the mechanisms themselves, and what we then need to do to make use of that knowledge.

* BTW, the answer was a steel bar, fixing the handle of the drive wheel to the frame.

7.3 Writing Things Down (And not in Hieroglyphics)

The starting point for reviving an old program is to look for any documentation about it - assumed here to be written in English. But what if that documentation was itself written on a computer, and all you have left is the files which recorded it. How do you start?

The most obvious thing about an English text is that it is written down using letters and punctuation marks - not in hieroglyphics or ideograms. The most obvious thing about a computer file is that it is written in binary words - numbers between 0 and 255 - or 0 and 65,535, or possibly even 4,294,967, 295. Hexadecimal makes more sense - count up from 0 to 9 as normal, then keep going with A, B, C to F - so F is hexadecimal for 15, and equivalent to the the four bit binary number 1111. Therefore, the number of characters one can use is FF (= 8 bits = 255), FFFF (16 bits) or FFFFFFFF (32 bits). The use of binary is built in to computer hardware and memory, so at least we usually know where the binary words start and stop. Consequently, the first task is to identify the word length.

This can be aided by the second task, identifying the mapping from binary data to the alphabet used. Modern computers use either ASCII (an eight bit encoding) [7.4] or one of the Unicode [7.5] encodings - for which there are encodings using 8, 16 and 32 bits. However, older encodings include seven-bit codes [7.6], which left a bit spare for parity checks - remember the paper tape from chapter 6. Currently, all these formats are well documented in English, and available on the Web. In a couple of thousand years, and a nuclear war? climate change? (list your favourite doomsday scenario), People will be saying that Champollion had it easy (he translated the Rosetta Stone). So, going back to the first task, if we choose a character encoding and print out the result, then if we have got the right word size most of the output will be readable text, otherwise much of it will be gibberish.

One difficulty: embedded in the encoding are "control characters" - characters that were used to control the physical devices used to print out the text. For example, the hex code 07 was used to ring a bell, but what does "CR" mean in the table in Character Set section in the Wikipedia entry on ASCII [7.7]. and what does "del" (hex FF) mean - and why have I called it FF rather than 7F? Perhaps resurrecting a Flexowriter might help.

Assuming you can sort out a 240v 50Hz A/C power supply (knowledge recovery step 1), and the mechanics are working, a Flexowriter illustrates with brutal clarity the relation between letters and their binary encoding. It looks like a large, electromechanical typewriter. Press one of the keys, and with a loud crash, the Flexowrite prints the corresponding letter. It also punches out a single row of eight holes in a paper tape. (I quietly assume that suitable typewriter paper has been found and someone has cut strips of paper that can substitute for paper tape.) Type a few letters, convert the

rows of holes to hexadecimal, and you should see a correspondence between the letters and the codes in the table cited above. If you don't, then try reversing the order of the holes (you've been reading the tape from the wrong side). The only oddity will be that half the codes will be 128 more than they should be - that is the effect of the parity bit - ignore that bit in future translations.

Keep playing, and eventually you will get to the end of the line - press carriage return to see the whole carriage get flung back to the start of the line - the carriage carries the roller that the printout goes round. CR - or carriage return - makes sense now, and you can see that the corresponding code appears in the tape.

A little more experimenting and you will discover the backspace moves the printer carriage back one character, but not the tape - press "underline" to see the effect. Press "Del" (delete) and you should get a row of eight holes, and when the same tape is read through the reader, the eight-hole row is ignored - "Del" is eight holes because the only way to alter a row on the paper tape is punch more holes. There were, of course, later, quieter punched tape machines, but the loud mechanical clatter makes the experience more visceral [7.8]. Congratulate yourself on having recreated some lost knowledge.

7.4 The Old, Old Format

Imagine you have found an old PC file. Perhaps it was transferred from five and a quarter floppy to three and a quarter on a dual drive machine, archived to a tape drive, then later restored to a machine that took a USB stick. But without the software, could you read it? Get coding.

Start with simple program that reads the bytes and packs them into an array - the file is so old that you can assume an 8 bit encoding. There are two obvious ways of presenting that array: as characters or as hex codes. We'll try the character format. Use Courier for the font and present it as rows of eighty characters - that was the typical width of a printer or an old display screen (they only displayed text, not graphics). For the characters, assume an ASCII encoding, except for characters in the range hex 00 to 1F - the typical range of the control characters - for which add hex 20, and change text colour to red.

If it was a text file, then what you should see is some readable text with the occasional red control character. Let me avoid a long aside by assuming that the text is in English and you read English. That text will contain structural cues we use unconsciously - capital letters to start sentences and proper names, full stops, commas, and so on. Now look at the characters in red. There is a chance that you will see "-*" every so often, and never more than 80 characters apart - you have found the carriage return/line feed combination that signal start a new line. Then check for other formatting characters such as tab ([7.7] goes into more detail). Update the program to translate the formatting characters into actual formatting. You are starting to get something that looks like a readable document. Perhaps use the red letter B's to toggle bold text on and off. At this point it may be worth looking up some of the old text file formats - if the file came from an early PC, WordStar format would be a good starting point.

This exercise in reverse engineering reaches back to technology current some forty or so years ago - still in living memory (assuming I'm still alive). The hardware peculiarities and limitations of old equipment help: only one font on a daisy wheel, text only VDUs, and even the inheritance from the Flexowriter. If the text had been French, one would have needed to work out how letters were accented. Other languages could be more confusing - only 24 letters in Greek alphabet, 30 in the

Russian, and then for other Cyrillic languages, things get "interesting" [7.9]

Later, computing machinery became more flexible, documents could use different fonts, and document readability depended on having the correct fonts on the machine presenting the text - I have had equations reduced to a series of rectangles because the machine it was printed on lacked the fonts used. Better equipment meant that the parity bit was no longer needed, doubling the number of characters available. For example graphics character such as " ┌ ┐ └ ┘ ╟ " could be used to draw table boundaries. But with more complex formats came more bugs - an ex-colleague took a job writing printer drivers, where he had to correct for bugs in both the text processing and printer software, as any error in printing would be blamed on their software.

So let us review the knowledge to read an old document. The human side of this relies firstly on the skill to recognise letters, secondly in the skill to scan the text in the right way, and third on the ability to read the language and to spot deviations from from normal text. The geek side starts with character encoding, adds on basic text formatting, and overlays this with the fancy stuff of fonts, graphics effects and so on. And we haven't considered later complications such as style definitions - "heading 2 Aerial 14 bold" - then style sheets, and then the way XML can be transformed as it is read to be presented in ways tailored to its different audiences. Old documents are going to be easier to reverse engineer than more recent ones,

One approach to long term sustainment - an experiment by libraries - has been to build a collection of the applications used, along with their file formats and rendering software. The sector has done censuses of formats found and also of files found where the format is unknown. Those researching these studies may find them under variations on "digital curation" - for example, investigate the Digital Curation Centre [7.10].

7.5 Digital Data (Excuse the Pun)

Excuse the pun, but the topic here is numbers stored as sets of decimal digits. I will ignore binary representations of numbers - a technical topic for the most interested readers. Rather, I will look at data exchange formats where the numbers are represented as decimal strings, such as "123", "123.456" or even "123.456e78". Moreover, I will assume that you have dug out the documentation describing the exchange format - reading twenty old numbers is pretty much the same as reading yesterday's, see the previous chapter. The question for this section is not the data level problem of reading numbers, but how knowledge is reified as data and linked to its information mapping. And for this I return to geometry, to points in particular.

To model a point in space you need three things - a manifold, a frame of reference and a model of a zero dimensional subspace within the manifold linked to the frame of reference. Too complex? open your favourite sketch pad - the manifold is the page where you draw things, assumed here to be flat, rectangular and covering the two dimensions of width and height. And assume for convenience that the frame of reference has its origin at the bottom left corner, that width goes left to right, and height goes up. Draw a single dot - it has zero dimensions, in that the shape has no measurable length or breadth (unless you are a pedant with a magnifying glass). Its position on the ~~manifold~~ piece of paper can be recorded as the length from the left edge of the paper and its distance from the bottom edge (that is how we set up the frame of reference). If this seems rather complicated for what is, electronically, a single click of a mouse, then be assured that the geeks who wrote the software know all about the complicated bit. Sadly, this is also part of the knowledge needed to read old data, so, as one of the geeks, I have to transfer some of that knowledge to you.

Points are defined by their offset from the origin. In a 3D Cartesian frame of reference, the co-ordinates of a point are given by its offset measured along each of the three axes, so (3, 4, 12) is 3 units from the origin measured along the X axis, 4 as measured along the Y axis, and 12 along the Z. One can then compute the distance of the point from the origin as

Sqrt((3-0)**2 + (4-0)**2 + (12-0)**2)) = 13 (7.1)

To read equation 7.1, it helps to know that "**" is a programming language convention for exponentiation, so "**2" means " the square of". Also, expressions are evaluated by calculating the value inside the parentheses before applying the arithmetic operators attached to them, and that exponentiation binds more tightly than addition - meaning calculate 3 squared and then 4 squared, then 12 squared before adding those results together - and finally take the square root of the total (inside the outermost bracket). Which in turn illustrates that even with arithmetic, there is significant technical knowledge needed - the facts of multiplication tables and the practical conventions for combining multiple calculations.

Cartesian co-ordinates are not the only way to locate a point. Spherical co-ordinates specify a point by its distance from the origin, and its azimuth and elevation - again in a frame of reference. In spherical co-ordinates our point (3, 4, 12) would be (13, 53.13, 67.38) (angles in degrees). Spherical co-ordinates are used in surveying, radars, telescopes and so on - any system that makes an observation from a particular point. Cylindrical co-ordinates provide another variation - range, azimuth and height. Other, more complex systems are available.

However, from the pure data viewpoint, all three reifications will be presented as a triple of three floating-point numbers - for a mathematician, that is all that 3D means. One might hope that the exchange format comes with some clues - perhaps XYZpoint or PolarPoint - however, the original programmers will have had the context for what they were doing, and may just give a file entry "point(1,2,3)". Long term sustainment asks that such context be publicly documented (not a comment somewhere in the program code) and recorded in a form that is itself sustained.

Context becomes crucial when one moves on to curves, particularly smooth curves. A typical reification for a smooth curve consists of a contiguous series of curve segments which are joined together to avoid kinks (tangent continuity) and often avoiding step changes in curvature (curvature continuity). A short list of curves used in CAD systems include:

- Biarcs - two circular arcs per segment but with curvature discontinuities;
- Cubic Spline - equivalent to bending a plywood strip to follow the segments;
- Rational conic (aka Consurf or Ball surfaces, after Alan Ball, one of my predecessors at British Aerospace) - originally used in the aircraft sector;
- Polynomial (curves built from wobbles);
- NURBS (complicated).

Each type has its own quirks, For example, in a polynomial curve, each segment is defined by a series of coefficients, but with the additional complication that these coefficients are for a set of "basis functions" and an "interesting" (read complicated) transformation is required if the data was written using different basis functions to the software reading the data. A cubic spline might be represented as a set of co-efficient for a cubic equation for each segment, or by the knot points where the segments join and the tangent at each knot.

The point here is that part of the knowledge about the curve - the algorithm used to calculate it - is embedded in the software that reads it. Interface specifications rarely include the mathematics of the

curve type, and have been known to make assumptions that render the recovery of old data a matter of pot luck. Worse still, some 3D software vendors have developed their own algorithms for 3D surfaces and implemented them as propriety software - the user cannot reproduce their old models unless they use the old software, nor can they transfer their data to other software [7.11].

In fact, in any complex model - geometry, systems design, data communications, etc - the boundary between the data and the software is not well defined. The knowledge of what the model represents is held by the original system designers, but software developers hold the knowledge of how the model is reified as data and calculated through algorithms. This makes long term sustainment of that knowledge a complex technical, commercial and political problem. Read my previous book for an answer [7.3].

7.6 Remembering Engineering Intent

So, assume you have gotten back your old software models, what do they mean? That is, what is the business context that gives the data significance?

There is a technique for validating CAD models called "Cloud of Points". Validation techniques are a first check that the model has been correctly extracted from the archive and reproduced by the software reading it - more than just "the data has not been corrupted", but rather "the software reading it works correctly".

Cloud of Points works by adding additional "check points" to the model. These are points that were on the surface of the original 3D model, and which the software reading it in should check whether they are still on the surface. The heuristic is that a 3D point requires only the most basic software to reproduce it, but the complications - and the bugs - come with surfaces. That is, the "check points" of the Cloud of Points are not intended to define the model itself, they are artifacts of the quality control process.

A "check point" provides a simple entry to "Engineering Intent", used here as a broad term to cover what is the business significance of an item of data, whether it is engineering data or not. For example, consider a project planning spreadsheet. Typically used when putting together a bid for funding, this will have totals for each row and for each column (check figures), with the key figure "Total Project Cost" in the bottom right corner. In many costing exercises, this is the "driving dimension" - less "this is what the costs happen to add up to", but rather a target figure based on what the customer will be likely to pay for. An increase in any of the project costs is likely to be met with an exercise to prune back other parts of the project in order to get back to the original total.

This "intent" is not part of the data for the model, it is part of the business context - the rationale explaining the decisions make. Recording rationale with business artifacts has been a notorious knowledge management issue since for ever, predating the daily use of computers - people don't make the effort of recording their thought processes. Years ago, when reports were drafted in pencil, Chas, a senior colleague, was ask to help out on a project. The project team were having enormous difficulty meeting a performance target, and Chas could not work out why that performance figure had been specified. But then the "ah ha!" moment. He himself had worked on the early stages of the project, and the figure was his first guess, and should be updated to what was achievable. That is, there is no reliable sustainment for knowledge that is not written down.

Back to the "cloud of points". The simplest approach is to scatter points randomly across the surface of the part, and then check the points are on the surface of the regenerated model. However, features like corners, the edges of holes or the points where other parts fit are more indicative of engineering intent - the part won't work properly if these are wrong. Consequently, engineers may pick out specific points on the part to act as checks, and add additional knowledge, such as the point being on a corner, as further validation. An implicit sign of intent.

7.7 Trusting in Provenance

In an archive, provenance provides evidence that the information came from a particular source and we can trust that it has not been interfered with. For computer age technicalities we are back in the world of PDM and Trust, as explored in chapter 5. But PDM is expensive to buy and expensive to implement, and very few computer files sit in such a controlled environment. In most computer systems, adding or deleting a file leaves little in the way of long term evidence, and what evidence there was is likely lost as information is migrated to new hardware.

BIP 0008 [7.12] is a series of standards on the "evidential weight of electronic evidence", and these are concerned with demonstrating in a court of law that digital records actually record what they proport to record. These standards require that the business processes for adding and modifying electronic data are well defined, leave an audit trail, and the audit trail is systematically checked. Which, in turn, implies documenting business processes such as document ingest in a well-defined way, and that in turn gets us back to linking business process and the meaning of data - except that here the processes focus on data about data - metadata. That is, when, say, an item is added to a digital store, one should collect data on which process was used, who added it and when. This becomes part of the metadata on the item, along with trust data such as the original source of the item, whether the data is trustworthy, and so on. For example, a letter to your bank manager explaining that "the funds are in the post" will typically be accompanied by a note by the manager saying "don't bounce his cheque before the clearing deadline tomorrow" - that is, we trust this letter has come from our client, and we trust they are telling the truth, but not so much as to risk the bank's money. That is, all processes run with implicit trust assumptions, and documenting them is part of having a well-defined business process.

Reversing the view, twenty years later we will have the basic trust metadata. We should also have a knowledge base about our trust systems - twenty years of experience of audit, process failures, possibly even attempted fraud. If every breach has been carefully investigated, the investigation documented, and corrective action taken, we have a case to trust our audit system and therefore the data. If there are twenty cases flagged up as concerning the finance director and he/she has simply signed off every case with the minimum of documentation, then a much deeper investigation is needed - perhaps a police check on that director's bank account. That is, although we have data (trust metadata) and knowledge - the inferences around trust decisions - what is the information that the data reifies? Is it simply the ostensive mapping "Status = Trusted", or do we actually have to reground the mapping in the accumulated knowledge of the business system, and say "'Status = Trusted' is equivalent to 'Status = possible fraud'"? We will return to the question of grounding in a later chapter.

7.8 The User Interface - or Old Users Eventually Die

Running software requires three basic elements - a computer, the software and a user - the person who says "I want to run this software with this data". The first job of the user is to turn the computer on. Turn up to a conference to show a presentation on a large display screen and it will take ten minutes to find the tiny black on/off button hidden in the gap between the screen and the wall. But it is not just the physical interfaces that can challenge, but also the interface paradigms.

I moved to a new job, and took my working habits with me. At the time, the CAD department I moved to was not the priority for new hardware and most programmers were still working on command line consoles - for example, to move and rename a file, one had to type

"mv oldfilename ../directory/newfilename" [7.13]

- no click and drag. Work proceeded one command line at a time, and most terminals were still text only - graphical interfaces were limited. I sat down at one of the new graphics terminals supporting multiple windows, and I was aware of some bemusement among my new colleagues when they saw I was running four consoles at the same time, and switching between them, not waiting for each command to complete. Now reverse that - imagine the change in mind set that would be needed to take someone brought up with modern windows interfaces to learn how to work in an old environment - no mouse and you can only type commands, and double full stop means go up one directory level.

Users get familiar with particular interfaces and their assumption - and changing an interface requires relearning not only of the characteristics of the interface, but also unlearning various motor skills and habits that users have been practising, sometimes for years. Imagine swapping a trumpet for a saxophone - going back to an old style of interface will be just as challenging. But users get old and retire, and one cannot rely on being able to access their knowledge - knowledge of what software was used to solve a particular problem, what it was called, how it was invoked, how to use the interface, and what data to put in and when.

The LOTAR project makes the assumption that it is impractical to archive users, and that their knowledge of how to use old software will be lost. Moreover, experience has shown that the software itself becomes unusable because of changes in the operating systems it relies on, and the hardware that the operating system ran breaks down as chips age. No matter - what is needed is the users' knowledge of their business (but see next section), and how to use results of past work in the current processes. That is, this year's users need the old data, but they need to import it into the software that they are familiar with. The choice is either to migrate the data to each new version of the software, and hope there are no bugs in any of the versions of the software used, or choose a stable standard [7.14] - LOTAR prefers ISO standards - and ensure that current software can read data in that standard. And ideally, by using validation properties, demonstrate that the models are unaltered by the process. and also that they have the provenance needed.

This long term perspective points to situations where what I have blandly called knowledge may have a physical realisation - my muscle memory of which keys to press, or my unconscious processing that means I "click here" without thinking. The embedded knowledge of such a system lasts only as long as the system survives. The knowledge embedded in a computer program works only as long as there is a computer that can run the program. The knowledge of how the chaff-cutter works lasts only until it rusts solid.

7.9 When Technology and Process Change - User Business Knowledge

"The past is a foreign country; they do things differently there" [7.15]. Archaeologists are known to relearn old trades and skills in order to understand the artefacts they find. This is not a question of simply uncovering facts - say, that this flint implement has these characteristics. Rather, in learning the skills of flint knapping, they can learn not only how things were made, but what are the traces left - what the discarded flint flakes look like, and therefore how to identify a flint processing site - recreating ancient knowledge by relearning the skills involved.

Technology changes lead to old skills becoming obsolete, leading to the question "how do we plan to preserve current knowledge" and leads on to the need to cover both the facts and the chains of reasoning used. For example, the availability of 3D CAD together with the ability to print parts directly from the design will make obsolete the skill of reading 2D drawings, even though for the aircraft industry, such skills will still be relevent for, say, another 40 years. One way to preserve such skills would be to retain a student textbook, where a series of graded exercises build up the student's ability to move between a 3D shape and its 2D representation. Another would be to develop (and validate) a system to scan drawings and automatically create a 3D model.

A second example comes from old mainframe database systems. Many such systems remain at the heart of modern business, Moreover, much of the business knowledge is embedded in the code that is built round the database, but often the mapping back from the code to the business is lost or obscure. Migrating to more modern systems is challenging - a problem descibed as having to pay of the (huge) debt of maintenance not done [7.16]. However, the programmers with the requisit skills are retiring while the new generations of programmers are only trained in more recent languages. Consequently, there has been a proposal to create new training courses in old languages - to rebuild the skills and techniques that are in danger of being lost.

More challenging still is the wholesale replacement of a complex technology. Prior to the 1970s, radar systems were based on analogue electronics: signals decayed quickly, amplifiers degraded the signal and electronics could only process one signal at a time. If multiple targets were seen, either the results were "stored" for a few seconds on a display where the dots dimmed slowly, or the system had to chose one of the targets to process. By the 1970s, with the advent of digital electronics, signals could be stored as long as required and could easily be replicated without adding noise [7.17], leading to parallal processing paths that could follow multiple targets. A complete change of mindset occured. So, I asked a junior engineer to compare a digital retrofit with the analogue radar. Having been brought up with digital radar, they could not make the mindset leap back into the old way of doing things, and I was left to rerun the results for the analogue radar taking out all the assumptions of parallel processing. In such situations, it is the unspoken assumptions of the culture that are hard to pin down and so easily forgotten.

7.10 So, What Does Sustainment Tell Us about Knitworks?

Reviewing this field trip, what we saw was a number of interconnected knitworks:

- Alphabets - Data presentation, linking to older machines, alphabets and binary numbers;
- Text file organization - reverse engineering using machine characteristics and English;
- Digital data - the mathematics of points and curves linked data linked to software;
- Engineering Intent - a point is space as a trace left by an engineering technique;

- Provenance - linking data to business processes in a particular organization;
- User Interface - knowledge residing in users about how to use computers;
- User Business Knowledge - what data means when used in the business process;

That is, saying information is a mapping from data to knowledge is too glib, as the same lump of data needs to be interpreted in the context of multiple knitworks - for example, the data "(3.0, 4.0, 12.0)" can be:

- a sequence of binary numbers interpreted as presenting an alphabet;
- a complete unit of a text file;
- a point in space;
- a point used in the original model;
- a point used to check that a 3D solid has been correctly reconstructed from its data;
- an input at a particular point in using the software that developed the model;
- needed to generate the evidence that the part presented is airworthy.

The previous field trip on IX as brought up a number of knitworks linking into IX, such as techniques for linearising and de-linearising complex models and texts. Or specific knitworks describing how data is distributed across a number of different applications, together with how components of an exchange can be brought together into a single message. It also hinted at business knitworks, such as how to get work on standards funded, and setting up the business agreements to build an IX.

And the first field trip, on PDM, delved into metadata and the different knitworks that a single data element could bring together: business process and organization size for trust measures; access controls, security classification and policy languages for usage controls; computing, business and human factors viewpoint on identifiers.

It is sometimes stated that the human brain can keep at most seven things in focus at the same time - if we could connect them up, eight goldfish would run rings round us. Mostly we learn by small knitworks - I used to know half a dozen different way to drive from North London to the centre. Then one day I took a bus - a route designed to pick up passengers rather than avoid hold-ups. On the trip I recognised part of one of my routes, and then, just a short distance further on, part of another route - my route knitworks started joining up into a road network. These field trips have also been an opportunity to join up what are often distinct topic areas in computing. However, we are in danger of building knitworks that are too complicated for us to follow. Look again at the ten gotchas of IX - all very different, all aspects of a single topic. But these field trips have focussed on what is basically a simple topic - how we use data to represent information which is a pointer to knowledge. To move on to systems integration we need to move up a level.

But what does move up a level mean? In video games, it means using the techniques we practised at a lower level to flatten, avoid or jump over tougher enemies. But in what follows, I will not simply mean more of the same but harder. I will be using it to mean linking knitworks that are appropriate for different levels of detail, so that we do not have to think about seven thousand things at once. We will be moving up from the level of punched tape to representing numbers, and up to representing geometry and up again to designing a wing. If we wrote out a description of a wing as a series of holes in a paper tape, nobody would even attempt to read it, but many of you will have flown on an aircraft designed by such holes.

And once we have the idea of level change, we can apply it to semantics - what does a term mean? That is where we are going in part 3. And hopeful at that level, we can work out how Zane, Ludwig,

and his fruit machine always give Alan three green apples (except when Ludwig is being silly).

Notes and References

7.1 Wikipedia "Open Archival Information System" https://en.wikipedia.org/wiki/Open_Archival_Information_System (accessed 29/08/2024) [Yellow Banana] and "OAIS Reference Model (ISO 14721)" http://www.oais.info/ Accessed 29/08/2024) [Brown Banana]

7.2 "Welcome to LOTAR International" https://lotar-international.org/ (accessed 29/08/2024)

7.3 Barker S "Model Archiving and Sustainment for Aerospace Design" SAE International, Warrendale 2020 ISBN 978-1-4686-0132-9

7.4 Wikipedia "ASCII" https://en.wikipedia.org/wiki/ASCII#:~:text=ASCII%20(%2F%CB%88%C3%A6sk,telecommunications%20equipment%2C%20and%20other%20devices. Accessed 16/9/2024

7.5 Wikipedia "Unicode" https://en.wikipedia.org/wiki/Unicode Accessed 16/09/2024

7.6 Wikipedia "ISO/IEC 646" https://en.wikipedia.org/wiki/ISO/IEC_646, 24/7/2024

7.7 Wikipedia ASCII https://en.wikipedia.org/wiki/ASCII 25/7/2024

7.8 Punched tape is also a clue to the real programmers' desk tidy - historically made from the plastic centres of the tape spools welded together with Typex thinners.

7.9 Wikipedia List of Cyrillic Letters https://en.wikipedia.org/wiki/List_of_Cyrillic_letters 26/7/2024

7.10 Home https://www.dcc.ac.uk/ 13/8/2034

7.11 Adrian Bowyer, Stephen Cameron, Graham Jared, Ralph Martin, Alan Middleditch*, Malcolm Sabin, John Woodwark"Introducing Djinn: A Geometric Interface for Solid Modelling" - see http://adrianbowyer.com/inge/djinn/intro.htm, Accessed 16/09/2024. * my old professor

7.12 BS10008 (was BIP 0008) "Evidential Weight and Legal Admissibility of Electronic Information" British Standards Institute 2080

7.13 For example, see Dream Host "UNIX commands — Working with files" https://help.dreamhost.com/hc/en-us/articles/214750848-UNIX-commands-Working-with-files, accessed 19/08/2024

7.14 EN/NAS 9300-003 "- Aerospace series. LOTAR. LOng Term Archiving and Retrieval of digital technical product documentation such as 3D CAD and PDM data. Fundamentals and concepts|". See also note 7.2

7.15 From L.P. Hartley's novel "The go between", 1953 (published before I was born)

7.16 Johny Morris "Practical Database Migration" BCS Learning and Development Ltd, Swindon UK, 2020, ISBN 978-1-78017-514-0

7.17 The computer code "A := B" is in effect a noiseless 3dB amplifier.