The Usefulness of Hindsight

Neville Holmes, University of Tasmania

very likely fail in the long run.

A better approach, we felt, would be to recruit team members from within various parts of the company and teach them how to program. That way, the essential internal company knowledge would be there from the beginning, and we felt—and later proved—that in a week we could teach enough about programming to the recruits to get the project off to a good and speedy start. Luckily, we persuaded management to adopt our recommendation.

Recently, reading an excellent article on system engineering (Diane M. Strong and Olga Volkoff, "A Roadmap for Enterprise System Implementation," *Computer*, June 2004, pp. 22-29) gave me an intense feeling of déjà vu. Perhaps these feelings were so strong because they took me back to the early 1960s, when I first worked for IBM Australia.

Times were quite different then than today, but early commercial data processing problems were much like those facing today's enterprise systems, to go by Strong and Volkoff's observations. Two particular points they make relate to perhaps the most important system engineering issues of all, and both relate to the development of a large corporate project.

History shows that user and data issues remain core development concerns.

USER ISSUES

At one time, a well-known international car manufacturer located its Australian data processing operations and much of its manufacturing on the outskirts of a large provincial city in the state of Victoria. The manufacturer's previous operations had been based entirely on punched cards, with their computation programmed by plugboard for a machine with a peculiar magnetic-drum main store. The data on the drum was actually stored on a single wire that had three close-packed windings for each data track, with the wire screwed to the drum at each end. When the wire broke, as it occasionally did, an impressive mess resulted. Seating the replacement wire correctly took a long time.

Implementation

A new supplier replaced this system with an IBM 1401. This transistorized machine, surprisingly, didn't have any plugged panels at all. With this amazing new technology, so different from the prior art, a new role came into being. In the plugged-panel era, programming had been the responsibility of the more experienced operators, but the new transistorized machines required specialist programmers.

Further, the new equipment was expensive, which the customer planned to justify by implementing new systems for all sections of the company. Yet where were these programmers to come from? Such talent was scarce at the time, particularly in Australia, so the customer's management people decided to advertise internationally and offer big salaries. This dismayed both me and my salesman partner on the project. Going by similar attempts made by other companies and applying a modicum of common sense, we felt confident this approach would not only be far too protracted, but would

We knew that the key to this approach, the Programming Aptitude Test, could be used to get recruits who would quickly learn to program, provided the company accepted only those who scored well. So the personnel department undertook a company-wide testing of workers with a minimum of five years' experience in the firm. Academic qualifications were rare and not considered relevant. We put together a team that represented all significant parts of the company, including the warehouses, assembly line, foundry, and the chap from personnel who ran most of the aptitude testing.

The project proved very successful, although many team members left for jobs elsewhere when their acquired experience gave them the confidence to venture into the rapidly growing data processing industry.

Implications

At this time, the unfortunate split between system analysts and programmers had yet to take place, so the team both designed their programs and coded and debugged them. The team succeeded not because they were good

Continued on page 118

Continued from page 120

programmers, but because they excelled at system analysis and design. Usually, the people working on a program knew the users and their context, what was needed, and what would be useful. If they didn't have this information, they knew who to go to for the best answers. In Strong and Volkoff's words, "the best and most experienced users from the business units … will make an ideal team."

This approach has some further implications for the computing profession as a whole. Anyone with the aptitude for it can program successfully and with relatively little training, as long as the coding system is simple.

The contrast seems to be that old-style programmers designed the system to best suit the user, while enterprise-system programmers require the user to fit the system. As Strong and Volkoff observe, "Consequently, some users must not only learn a new system but must take on additional, sometimes unfamiliar, tasks and responsibilities."

Consider how different the software process would be if developers designed the system to be completely adaptable by the user. In the 1960s and 1970s, mainframe computers shipped with a great variety of features and parameters, and manufacturers tailored their operating systems with macrodefinitions that let a data processing department build its operating system not only to fit the computer but also to suit its users. Are enterprise systems all that different? Who are they designed to suit?

DATA ISSUES

Another international company, famed for its tracked vehicles, went into data processing with an IBM 1440. Because this represented a significant investment for the company, the general manager took a strong interest in the project and had the vendor install the computer in an open area next to his office. He liked to see the expensive machine at work when he came in from his car park in the morning.

The data processing people wrote a special program that read and punched empty cards, moved the seek arms on the disk drives, and printed noisily without moving the paper. They also scheduled an operator to come in early and start it running when no other work was ready.

Implementation

The company wasn't particularly big at that time, so it justified the computer purchase by implementing many small applications for different areas of the company. That the company had two of the relatively new 1311 disk drives—the first such relatively inexpensive drive with removable disk packs—made this strategy feasible.

The two most important parts of a computing system are the users and their data, in that order.

The programmers bent their backs to the work and quickly and successfully implemented a variety of programs for a variety of users. Then a problem arose. With many data files sharing a disk pack, it became feasible, and certainly desirable, to share data between departments. The programmers attempted this, but had great trouble putting code together and, when they got a program going, generated results that sometimes went strangely wrong.

After much panic and bewilderment, the cause eventually became obvious. The data that one department kept often proved incompatible with data kept by another. Sometimes the data had different names, formats, or units, and so on.

When management realized this, the data processing department reluctantly stopped developing new programs and turned their programmers to documenting the different data the company used in its manual systems as well as its already automated systems.

As this work proceeded and conflicts were removed, the programmers built an authoritative data dictionary that gave different data standard names for both programming and everyday use, definitions, machine and display formats, constraints, and explanations. In the final stages of this effort, the programmers revised programs already written and converted data already stored on disk.

Overall, this effort revealed a company culture that inhibited standardization: At the time, people across the company showed reluctance to waste their time explaining obvious things to upstart technicians. They also balked at making trivial and aggravating changes to unimportant details of their work.

However, after a while, it dawned on many of these scoffers that this data dictionary could be useful to them. By then, it was as much used by people outside the data processing department as by the programmers. Certainly, it became plain that the dictionary not only made further application development feasible, but also sped up the process.

Implications

The two most important parts of a computing system are the users and their data, in that order. Although the users' needs must be met, their most important need is that the system produce, without qualification, trustworthy data.

The actual logic in the programs and the arithmetic in the computer can give rise to errors, but thorough checking during development will rectify them. In batch programs, common practice took measures—a hash total in the case of nonnumeric data—of all data just after it was read in and just before it was written out, so that users could compare the totals at the end.

Programmers must, however, make sure the data is trustworthy in the first place. Frequent failures of this kind led to the old acronym, GIGO—garbage in, garbage out. As Strong and Volkoff note, "… data cleansing is one of the most critical technical issues for successful implementation."

This issue has two aspects: standardizing the data usage, as in the second of my two anecdotes, and blocking invalid data on its way in to programs. Good management can greatly lessen invalid data generation, but many human errors can be made while getting data into the computer.

In batch programming, common practice demanded that operators check the daylights out of all incoming data, and often supplemented this activity with independent file-checking programs. I remember reading in an old issue of *Datamation* that the US Air Force's data processing people spent 80 percent of their coding effort and space on input data checking—and considered the effort well worthwhile.

The excellent project management advice in Strong and Volkoff's article caused me to think of the two examples I've shared. However, their description of enterprise systems, and the separation their complexity brings between the technical people and the end users, reminded me of the strengthening of data processing departments in the 1970s and the rift between the typical data processing department and its users.

Data processing departments often seemed to exert a kind of arrogant tyranny, running their systems on their own terms. The introduction of Wang minicomputers as word processors in the late 1970s started breaking down this imbalance, a process carried further when business PCs became popular.

Hindsight suggests that this cycle might soon repeat itself. What will cut the enterprise systems people down to size now? Personal database machines with hardware to ensure integrity, authority, and sharing? Could these machines enable a hierarchy of entities down to the individual, which would let users look after their own data but use a network to get a coherent view of what they need from other entities? Maybe.

Obviously, more hindsight by computer professionals and their educators could greatly lessen the extent of the problems that Strong and Volkoff so clearly document. ■

Neville Holmes *is an honorary research associate at the University of Tasmania's School of Computing. Contact him at neville.holmes@utas.edu.au. Details of citations in this essay and links to further material are at www.comp.utas.edu.au/users/nholmes/prfsn.*