

IT solutions for the euro: the four paths

Anna Karenina famously opens by telling us that all happy families are alike, but each unhappy family is unhappy in its own way.

In a similar manner, each euro failure will be similar, but that each euro success will be a success in its own way.

The failures will not be sudden and catastrophic; rather they will be the result of one or more poor decisions which result in more and more processing being done manually. Gradually, organisations will find that they are doing more and more business which involves 'limited scale manual workarounds' until so much effort is expended on these that the profits and business collapse.

One of the problems with the euro is that it affects virtually every part of every company, and that solutions to problems posed by the new currency will appear everywhere.

One American investment bank produced a list of 'necessary changes for the euro' which contained the requirement that all statements and advices to customers would need to be in dual currency for some time until customers became used to the new currency. Then someone suggested that the great majority of investors were Americans who didn't think in Franks, Marks or Lira anyway. They thought in dollars.

Assets in Germany or France are subject to different national market risks - if the French government does something which causes its credit rating to fall, then the value of its bonds will fall, euro or no euro. The asset value risk therefore retains strong national determinants. But after 1 January 1999 the *currency* in which the assets are expressed no longer depends on which 'in' country the asset is in.

So, reasoned the bank, the financial value is actually *better* described to an American investor as a euro risk, rather than as a series of risks in separate currencies. Indeed, the 'helpful' national currency additional information actually misleads, since it gives the impression of differences where there are none. Accordingly, the bank decided not to add a second currency, but rather to explain and persuade its customers of the superiority of this approach - which also happened to be much cheaper.

Clearly, the IT component of this solution was zero. If we wanted to describe this solution, the term 'marketing' would be better than 'system'.

But for all of this, it is generally conceded that the vast majority of the work to implement the euro will involve IT. Most estimates include a factor of over 65% for IT. And making mistakes on the largest component of a budget will usually result in significant problems.

So the decisions which determine the IT approach still have the greatest impact on the cost, and effectiveness, of an organisation's euro project.

The fourfold way

Murray Gell-Mann was one of the leading physicists of the 20th century. One of his theoretical approaches to particle physics identified a symmetrical set of relations which led to what he called, borrowing from Buddhism, 'the eightfold way'. Well, the euro is only half as hard as particle physics (which therefore makes it a bit harder than rocket science), so in place of the eightfold was we have the fourfold way.

The four paths which may be followed are:

- 1) Fix
- 2) Replace
- 3) Encapsulate
- 4) Outsource.

It will be seen immediately that these paths do not share a common scope. It is fairly easy to comprehend what is meant by fixing a system. It mostly involves code and data changes, and perhaps some organisational tweaks. By contrast, outsourcing runs the range from handing over responsibility for running a computer to handing over responsibility for payments, receipts, ordering and reporting to regulators. Not quite the same, I suggest.

Fix

By fixing a system, we mean actually going into the system and repairing the code. This is the most obvious and appealing method of making computer systems meet newly perceived requirements. However, it is not necessarily the best or most appropriate if the solutions are not well defined, or if there are conflicts with other projects.

A wide range of factors including determines the difficulty of fixing a system:

- Architecture of the systems to be changed (well-structured code or spaghetti? A good, simple design or just hacked together?)
- The availability of documentation on design and construction of the system
- Number of interfaces
- Availability of staff who understand the system, the language in which it was written and the hardware on which it runs
- Interfaces are simple, well-documented, and information passed from the modified system either does not contain the problematic data or can do so in a manner consistent with both the sending system
- Stability of the system - is it a Schleswig-Holstein system: "Only three people understand this systems, one is dead, one is mad, and I have forgotten how it works ..." - one of those famous systems which allegedly exist from the 1970s which 'only one person is allowed to make changes to...'?

Fixing is most appropriate when the origin and destination are well understood, and where there are no major competitors for resources.

For a medium to large sized implementation, the sequence of a 'fix' implementation will be, more or less, as follows:

- **Define the changes to be made.** These will need to be identified for matters relating to internal and external circumstances. These will be matters such as changes to payments and receipts, accounting, treasury functions and banking relationships. Any EDI links to other organisations will need especially careful review and understanding.
- **Determine how changes will be made.** This involves defining the approach to issues such as dual currency accounting and reporting, triangulation, rounding and related matters, and set guidelines for how changes will be made in all systems.
- **Set up software scans.** This involves three main tasks: determination of candidate field structures, definition of the systems environment, and preparation of the source code for review. For the year 2000, the structures are mostly fairly easy to identify - groups of storage defined as six units divided into three pairs, fields with 'DATE' as part of their names and so on. Also, date fields are not often the subject of extensive further processing; once you have found a date fields, there will probably not be many more fields to which it is moved or processes in which it is modified. For the euro it is more difficult. Fields defined with two decimal places are clearly candidates in many countries -but in Greece, Italy, Belgium, Portugal or Spain it is more difficult, since the currencies there have no decimal places. Fields with 'AMOUNT' or 'AMT' in their names will also be worth identifying and reviewing, but in general, it will not be easy to identify all the fields which have financial amounts contained in them and changes which they undergo. Whatever is done, it is important that the set of criteria catches *all* possible amount fields. False positives are acceptable, false negatives are not.
- **Conduct software scans.** Once the rules for change and identification have been set, the code will need scanning. The principles for this have, by now, been well established by the year 2000 projects, and this should present few difficulties.
- **Correct the code.** Once the false positives have been identified and eliminated the code changes should be made. As with the year 2000, this should be considered to be a once-and-for-all set of changes. Testing may identify errors in the modification of code, but it should not be necessary to have to re-traverse the code and correct large numbers of items, or to modify for a new set of rules.

For the year 2000 teams this is fairly easy. If a new type of date field were identified in a particular system, then it would have be structurally different from other types in order not to have been identified on the first scan. As such, it would be possible to identify the field structure and scan specifically for it. The new fields could then be corrected to a four year date format (or whatever else

had been determined as the solution)

For currency fields, it is more difficult. And the problem is not finding the fields; it is in being certain that the changes carried out are complete and exhaustive. If it is found that a type of change to functionality is required which had not formerly been considered (e.g. if dual pricing were to be legally mandatory, or some form of input procedure were found to be required), then it would be necessary to go through all of the same steps as in the original scan-and-correct exercises.

- **Test the code.** As usual, the code must be tested, and any deficiencies rectified.
- **Implement the code.** Finally, the tested code is implemented, with suitable version control and controls such that any other contemporary change are not going adversely to affect the system.

It might seem that the 'fix' route is a fairly safe and certain route, and indeed, many organisations have chosen this path. For one thing, it is the method which an IT department would naturally select if no other choice are identified by other parts of the organisation.

But there are some serious problems with it. For example, we have seen that it is an approach which anyone would only want to do once. Having to retrace the scan and replace route would be highly undesirable. For those organisations working in areas where the changes are unclear, however, this may be exactly what they will have to do if they follow this route.

If a national government suddenly requires that all accounting information is held in two forms (for example) an organisation which had decided that single base currency of euro would be caught out.

'Fix' is an option for those who are certain of the tasks which need to be done, and have the resources (human, financial and temporal) to complete the work in good time.

Replace

There are two main variations on the theme of replacement. In the first, it may be found that the vendor of a package has developed a new version of a system which includes functionality for the euro and makes an upgrade available. In the second, it may be found that there are features of the euro requirements which make it impossible to continue to use the current system and that it must be replaced.

While the two approaches appear very different, we should remember that the replacement of a simple accounting system which has no interfaces to other systems may well be simpler than an upgrade of a complex system embedded in the midst of many other systems. For this reason, the two cases will be treated together.

It has many of the benefits of 'fix' without the same constraint of development resources. Certainly work will be required to effect the implementation of the new system, but generally this will be significantly less than the work required for the fix option.

Furthermore, there is the 'safety in numbers' defence - if a large number of customers of the vendor are upgrading together, there is a higher likelihood of serious errors being identified and corrected than if a single organisation is determining the outcome.

The process of implementation is similar to the latter stages of the 'fix' approach. The software must be installed and tested against a sound set of statements as to the changes which the euro will bring.

Of course, one of the problems with this approach is that if your industry has not determined with some precision what the requirements for the euro are, then the software vendor's information is unlikely to be significantly better. And if the changes required do serious violence to the original design, you may be worried about the quality and content of the upgrade - especially if there is only a short time for testing.

Encapsulate

If the first two approaches are not sound options, whether because of uncertainty, lack of resources or because it has been determined that the current system is fine today, and will be once again be satisfactory in 2002, there may be reluctance to convert a perfectly good system for only a three and a half year problem. In this case, encapsulation may be an attractive option.

This option is sometimes described as implementing 'converters' which translate amounts from euro to national currency or vice versa, but this fails to convey the heart of this approach.

Encapsulation is an extension of the idea that only input and output functions need modification. The principles behind it are as follows:

- Changes to the logic of currency handling are localised in a layer outside the main functions and hence can be rapidly implemented and, if necessary changed at a later date as requirements become clear
- The core functionality of the current system is unchanged, so that an organisation undertaking a large or difficult year 2000 project can modify the core system as much as necessary, without the prospect of the euro project modifying critical functions thus causing conflicts
- Encapsulation can be rapidly implemented, because once the rules and logic are determined, the same functions can be called repeatedly.

Not all forms of encapsulation are equally good, however. An organisation using encapsulation techniques must beware that the additional software does not impose a major burden on the hardware running the core system with a large number of additional calculations. Similarly, it will be desirable to implement exactly the same functions over all systems, so the routines which are developed should be capable of being implemented on a wide range of platforms ranging from mainframes, to minicomputers and workstations and servers.

In the year 2000 world, we have seen many encapsulation tools which run only on IBM mainframes, often with limited scope even there.

This approach would not be satisfactory for the euro.

Outsource

In many ways, this is the most attractive of all options - but only if it is available! In the context of the euro, 'outsourcing' has something of a special meaning. It is not simply handing over the operation of computer systems to another organisations; it is instead having another organisation offer the use of its own systems.

It is similar to traditional bureau operations, but ideally incorporates some of the non-computer operations which are common to both businesses.

An organisation which has made the investment necessary to be able to handle euro business in all of its likely forms is in a strong position to begin offering services to others in the same business.

There will be an immediate concern that the provider will be tempted to 'take a peek' and seek commercial advantage from learning about its competitors business. In the world of international banking, however, this has been found not to be the case. JP Morgan first ran, and now provides the operational management for the Euroclear organisation through which the majority of settlements of international fixed income trades pass.

There has never been any suggestion that the bank has taken the slightest 'peek', and indeed it would be entirely against the commercial interests of the bank to do so. Euroclear will continue to benefit the markets and to pay a fine return to JP Morgan for as long as the bank chooses to do so. If there were any suspicion of malpractice, the business would immediately migrate to Euroclear's competitor, CEDEL.

Similarly, we would expect that 'Chinese walls' and the rest of the compliance apparatus which grew up in the City of London in the 1980s would be replicated in an outsourcing operation for the euro.

Summary

Coming to terms with the euro is not simple, nor is there a 'one size fits all solution'. Even within a single industry, there is no one answer - for some the best solution will be to invest and seek business advantage from selling euro services to others. For others, there may be a simple systems upgrade which will meet all likely changes to business.

But no-one can say, without long and careful contemplation of his or her business's situation, what the most appropriate solution will be.

Anna Karenina's solution to her problems was to jump under a train, let us hope that no IT managers are tempted to solve their euro problems in a similar manner.