Business Processes, Legacy Systems and a Flexible Future

Peter Henderson
University of Southampton

October 1999

1. Introduction

Modern commercial and industrial organisations form a complex network of inter-related businesses evolving cooperatively and competitively. The challenge for IT, and for Software and System Engineering in particular, is to be able to support this evolutionary change economically and reliably.

A major problem is the extent to which IT is a disabler of business process change, when in fact it should be an enabler. This problem arises when the changes required by the business processes are made too expensive by the sheer size and complexity of the already installed IT. These so-called *legacy* systems, which are essential to the support the business process, comprise computers, programs, databases, networking and much more, as we shall see.

In order to make further improvements to the system delivery process we need to make substantial progress in understanding the way in which business processes are supported by these systems. If the relationship between the business process and the system that supports it is better understood, then we may be able to make substantial improvements to the cost of changing that business process.

Precise formulation of Business Processes needs to make use of more scientifically sound methods. Such methods have been developed in the past by the IT Systems Engineering community for application to software engineering problems. Application of such system engineering techniques to business process analysis, is leading to the development of shared formal models for understanding business processes [8, 9, 14]. This in turn is leading to the development of sound methodologies based on those models. Equally, the development of scientifically sound techniques for encapsulating legacy systems and for mapping business process change onto them is leading to IT architectures for direct system support of Business Processes, and hence to more economical evolutionary change [2, 5, 7].

Methods for implementing change to legacy systems, cheaply and incrementally, are building for example on advances in object-orientation, databases and transaction processing. More effective exploitation of IT (in particular open systems) is leading to increased competitiveness for end-user organisations.

This paper addresses these issues, with the objective taking a particular view of the options available

- To improve responsiveness to the increasing pace of change in business processes, in the presence of legacy IT systems.
- To understand how large scale computer systems can be made to evolve, so that businesses can change their business processes economically and so maintain a competitive market position.
- To improve the cost-effectiveness of making engineering changes to computer systems in response to business process change.
- To understand the relationships between business processes and supporting information technology and to improve our ability to specify, design and implement technological support for business processes in a state of rapid evolution

The key word here is *evolution.* There is the need to enable rapid and economic evolution of business processes while incorporating the huge business assets that are locked into legacy systems.

First we look at the nature of the problem. In particular we discuss issues relating to Business Process Change and to Legacy Systems. Then we discuss some of the approaches which have been taken to addressing the problem of enabling the first in the context of the second. Finally, we turn to the issue of the future of system architecture and discuss the design goals for the kind of flexible architectures which are emerging to ameliorate the problem.

2. The Nature of the Problem

Successful End-User Organisations run their businesses according to well understood, if not well defined, business processes. These business processes are supported by an IT system, often very large and usually a mixture of ancient and modern subsystems. Each organisation depends upon the continued working of the existing IT system, or at least of the services that it provides. Yet the oganisation requires to extend its functionality periodically. It was, at one time, the case that this periodic change in functionality could take place in a well-structured, release-oriented project. But today, the pace of business change is such that, increasingly, new functionality has to be realised and reliably installed in a matter of days, or even hours. Architectures are emerging which will support this. We discuss those eventually. But first, let us review the nature of business process change.

2.1 Business Processes

In recent years there has been a great deal of emphasis on the problem of business processes. In particular, they arise in the context of Business Process Re-engineering (BPR), a management activity (usually facilitated by consultants) which looks at the whole economic basis of a business, with a view to radical improvement [6].

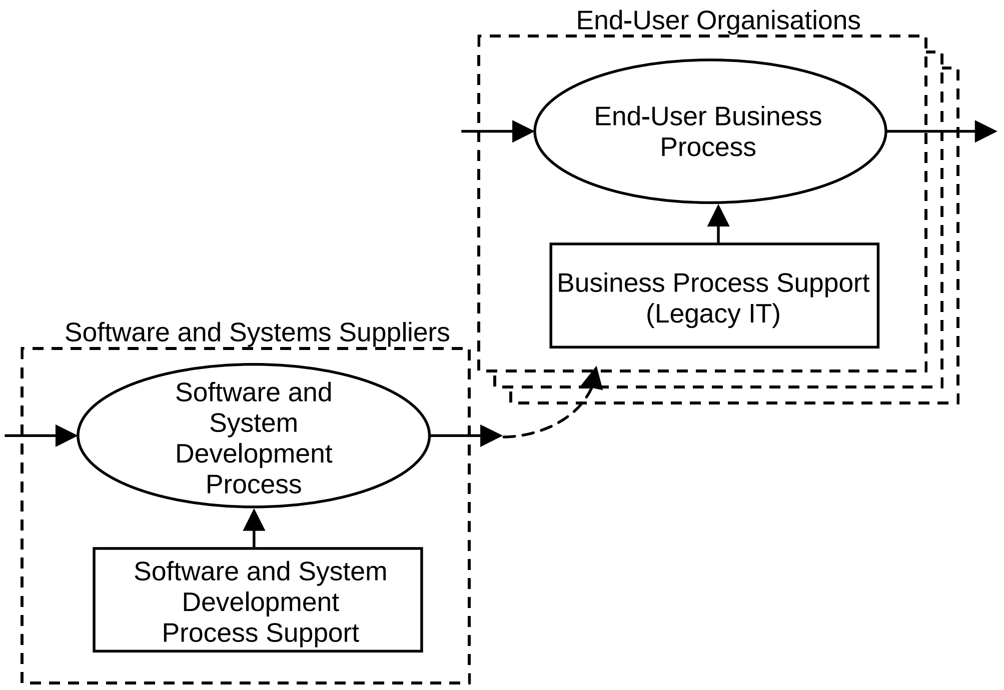
End-User Organisations
End-User Business Process
Business Process Support (Legacy IT)
Software and Systems Suppliers
Software and System Development Process
Software and System Development Process Support

Figure 1: System Suppliers and End User Organisations

Business processes are the activities that a commercial organisation performs in order to carry out its business. All organisations depend for their competitiveness on the efficiency with which their business processes operate. In Figure 1(top right) we have shown, in a gross oversimplification, the business process of an end-user organisation as an activity that has input and output.

For example we could say that the business process of a bank (say) is to input monies in amounts determined by investors and by the returns on its investments, and to output money to its customers, shareholders and into its investments. Within the bank’s money-handling business processes there are many subprocesses which determine what actually happens on a day-to-day basis. The business processes are operated by people and by computers. The competitive positioning of the bank is largely determined by how judiciously it has chosen these business processes and how effectively it runs them. To maintain its competitive position it has to be able to change these processes very quickly and very reliably.

The important issue is that end-user organisations have critical business processes that are supported by computer systems. It is this IT base which is both the enabler of change and its disabler. It is the enabler when we can support new types of business process that would be unmanageable without it (for example, the increasingly sophisticated customer services offered by banks). It is the disabler when the complexity of the installed IT is so great that the cost of making changes to it becomes unacceptable, and desirable business process change is either too late, too expensive or simply unachievable.

2.2 The related System Engineering Problem

A major problem then is the extent to which IT is a disabler of business process change, when in fact it should be an enabler. This problem arises when the changes required by the business processes are made too expensive by the sheer size and complexity of the already installed IT. These so-called *legacy* systems are shown in Figure 1 as supporting the business process, for that is indeed their role. The legacy IT system comprises computers, programs, databases and networking. But legacy is much more than that. The organisation has investment in skills, in data, in business processes themselves that must co-evolve. You can’t change part of the system without changing the parts that interface with it. And as systems get older, everything becomes related in some way to everything else, with the consequence that it soon becomes impossible to change anything without having to change almost everything else.

For example, consider the simple situation shown in Figure 2. This shows nothing more than the individual components that comprise a very modest legacy system. For the purposes of the argument I am going to make, we will consider these to be software components, although I am assuming that they are each of some considerable size (tens or hundreds of thousands of lines of code each).

To make the example more concrete, let us suppose the components in Figure 2 comprise a banking system. Perhaps component 1, on the left, is the Customer Service module. It is used to handle transactions with the customer at an Automated Teller Machine (ATM) and over the telephone. That is, it can render its interaction on the screen of the ATM or on the screen of the Customer Service Agent answering the telephone. Suppose that component 7 is the communications module. It connects the ATMs across the country with the database servers in various main branches and it connects the Customer Service Agents with these same databases over a network.

You can see that the relationship between these components is such that any business process change that requires changes to components 1 and 7, might in fact require changes to most if not all the components in between. In a realistic banking system, even though the components we are talking about are large (they may be client or server applications, for example), there may in practice be hundreds of components affected. This is the legacy IT problem. Old systems are so interconnected, as a consequence of a lifetime of maintenance, that even apparently small business process changes can require substantial re-engineering work. We shall return to this example later, when we consider how a change to include the EDI requirements of an internet banking service might be implemented.

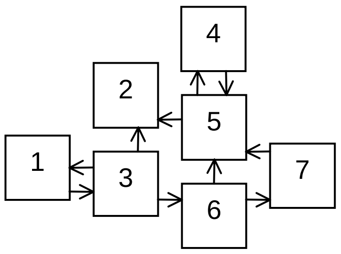

Figure 2: A legacy system of components

The investment in legacy IT is such that only an evolutionary approach to change is possible [1]. Even then, many desirable changes prove much more costly than the business can accept. And usually the end-users are not in a position to evolve the systems themselves. As often as not, they require that to be done by their system supplier (bottom left, Figure 1). Whether the driving force is market-pull (as we have so far presented it) or technology-push (where suppliers are making new technology available) the inter-relationship between the two types of organisation is clear. Change in one begets change in the other.

The organisations form a complex network of inter-related businesses evolving cooperatively and competitively. The challenge for IT, and for Software and System Engineering in particular, is to be able to support this change economically and reliably.

Therefore, we need to discover methods of systems engineering which are responsive to the need for business process change on the kind of timescales and at the kind of cost which the end-user businesses find acceptable. Moreover we need to formulate these methods in a readily transferrable way.

The question is, can the systems development process, for systems that include a large legacy component, be improved to the degree that development processes in other manufacturing disciplines have been improved in the recent past (through techniques such as concurrent engineering)? It is our belief that in order to make further improvements to the system delivery process we need to make substantial progress in understanding the way in which business processes are supported by these systems. If the relationship between the business process and the system that supports it is better understood, then we may be able to make substantial improvements to the cost of changing that business process.

2.3 The central issue

Looking again at Figure 1, there are clearly four areas where we could hope to make improvements. Reading downwards, from the top-right, these are

1. The End-User Business Processes
2. The Business Process Support (Legacy IT)
3. The Software Development Process
4. The Software Development Process Support (The Development Environment)

Each of these presents opportunities for us to address aspects of the problem of business process change in the context of inflexible legacy IT. Central, however, is item 2 where increased flexibility in the installed IT will do much to ease the problems which occur in the other three areas. A more flexible architecture for the installed IT will reduce the cost of change in the end-user business process. It will also loosen up the system development process itself. The ideal scenario is when the end-user conceives a requirement and the development process can very economically put together a solution because it comprises mainly an extension to the existing IT. Implementing the solution means no more than dropping new components into the current system, where it configures itself and begins to supply the new services required by the business process change.

Business process change comes about as a consequence of both market pull and technology push. Whichever is the catalyst, an organisation evolves by changing its business processes to accommodate new behaviour. So, understanding how to capture business processes precisely is an important part of solving the business process change problem [8, 9, 14].

Supporting those business processes is an architecture, which however you look at it is built from components [11, 12]. Understanding how architectures constrain or enable flexible change is an important part of improving the business process support. Increasingly, I have come to believe that this is the central problem to be solved, the one which makes all the others take on a new complexion. The key issue, as we have said, is that of flexible architecture.

Finally, the Software Development process and its support environment need to be more efficient. Modern development environments for new applications are efficient. For example Microsoft uses a daily-built process called synch-and-stabilise [3] which produces fit-for-purpose applications by a regimented but flexible incremental development process. This has been followed also by Netscape [4], again with great success. The Linux development method, which in fact Netscape have more recently adopted, is different but just as versatile [10]. It uses open-source development, where anyone is able to see the source and extend it but a central control enforces a strategy of only incorporating the best components in the “authorised” version. Both these development methods are extremely effective, and flexible.

Notwithstanding the success of these methods, and not concerning ourselves with the issue of whether they are applicable to the Legacy System problem, we can see that the flexible architecture solution is orthogonal them. A more flexible architecture will clearly alleviate some of problems we face with legacy systems. And given that flexible architecture we could still adopt synch-and-stabilise or open-source development methods, thereby getting the advantages of both.

3. Flexible Systems

Let us look at precisely what we mean for a system to have a flexible architecture. In the previous section we have referred to the idea that new components can just be dropped into the existing system, by which we meant that, on receiving the new component, the system would continue to run without mishap, eventually deploying the new functionality. Of course, with modern operating systems, we see this happening today at the lowest level. Devices are often plug-and-play. Applications can be launched, which share data with other applications. Moreover, modern environments for deploying components in distributed systems also provide the mechanisms for plug-and-play applications [5, 13] This is important enabling technology, but not exactly what we want. We want to be able to conceive a new business process and launch it, ideally without

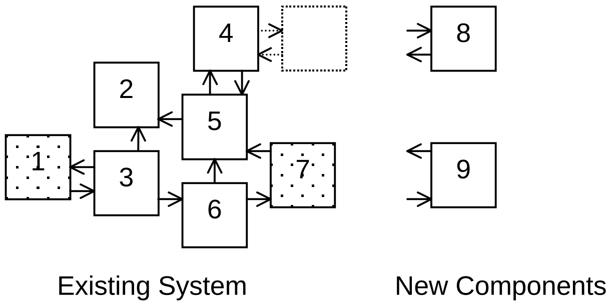

Figure 3: An evolving system of components

having to change anything which already exists. Clearly this will not always be possible. But we want to get as close to this ideal as we can.

For concreteness, let us return to our banking example. Refer now to Figure 3, which has the configuration of Figure 2 as part of it. The components of interest in the original syatem are shown with shading. Component 1, you will recall is the Customer Service Module, which knows how to render interactions with customers either via an ATM or via a telephone operator. Component 7 is the Communications Module which knows how to connect ATMs, Customer Service Agents and databases all of which are geographically distributed.

Suppose that the business process change which is required is to add some home banking service over the internet (or interactive digital TV) and that this consequently requires functionality not available currently in components 1 or 7. Two new components are designed, to add the new functionality. Component 8 can do all the work of component 1, and also generate web-based interactions. Component 9 can do all the work of component 7, and also communicate over the internet (or the modem in your digital TV receiver). Component 8 plugs into a different place from component 1, but component 9 must plug into the same place as component 7.

There are many scenarios as to how we could upgrade the system, including

- Replace 1 and 7 immediately by 8 and 9, respectively
- Replace 1 by 8 and use for a while just to check that 8 does indeed supply the same functionality as 1. Schedule the second upgrade later.
- Run with both 1 and 8 installed so that, if 8 proves inadequate, swapping back to 1 is possible (hot swap)

We have purposely concentrated on 1 and 8 here. Firstly, because they appear to be able to co-exist in not requiring the same socket and secondly, being responsible only for a simple interaction, they probably don't hold a lot of state. An error would then be much easier to recover from. It would be much harder to

- Replace 7 by 9 and use for a while just to check that 9 does indeed supply the same functionality as 7 (we may lose a lot of state).
- Run with both 7 and 9 installed so that, if 9 proves inadequate, swapping back to 7 is possible (would need extra components to allow both to co-exist)

Yet this is precisely what we want to achieve. Ideally, we want to be able to run with the new components installed alongside the old components in such a way that, if the new components prove inadequate, we can revert to the originals without loss of information.

These upgrade scenarios are summarised in Figure 4.

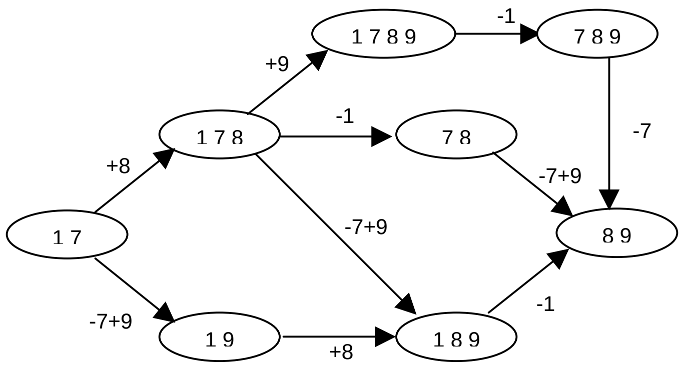

Figure 4: Order in which components are changed

Each node in the diagram denotes a configuration of the system. The nodes are labelled by which of components 1, 7, 8 and 9 are installed (we assume 2, 3, 4, 5 and 6 are always there). Ultimately, we will reach the state where 8 and 9 have replaced 1 and 7. But the route we take, the speed with which we move and whether or not we can reliably reverse a step dictates how effectively we can implement the change. A truly flexible architecture is one which will allow all these routes (and many others) and will allow every step, once taken, to be reversed.

The consequences of such a requirement for the information integrity of the system are enormous. If we were to install a faulty component and run with it for a while, it could go wrong in a way which results in unrecoverable information loss. This would be unacceptible. One solution would be to run old and new systems concurrently for a while, until the upgrade can be fully trusted. Only then is the old component retired. This may sound very expensive. But architectures which admit this form of evolution are possible and will become the ultimate flexible architecture of the future.

4. Conclusions

We have set out the problems which arise for IT supported businesses as a consequence of the huge rate of change now required of their business processes. Not least is the problem of the investment in Legacy IT. Of course, Legacy IT is a major asset, otherwise it could simply be disposed of, but it is also a major disabler of change.

Central to the issue of overcoming this disabling effect is the way that the business process support system, which includes the Legacy IT, is to be evolved. We have argued that more flexible architectures for the business process support system are necessary and will alleviate problems which arise both upstream (in the development process) and downstream (in the end-user business process) too. In fact, a summary of the approach we have taken is that we must take a business process oriented approach to system architecture.

A business process view of business service provision, when mapped down to the architecure level, does indeed embrace the flexibilty which rapid business evolution requires.It is not antipathetic to the need to evolve large legacy systems. The manifestation of the business process view in the software requires the adoption of flexible software architectures allowing for in-flight software change, that is, the new functionality is installed without the system having to stop providing its service. Embracing legacy systems (code, data, business practice etc) in this business process, plug-and-play world is still the major challenge. Once solved, a more flexible future awaits.

5. References

1. Bennett, K.H, M.Ramage, and M.Munro A Decision Model for Legacy Systems, IEE Proceedings - Software, June 1999.
2. Cardelli, Luca Abstractions for Mobile Computation Microsoft Research Technical Report MSR-TR-98-34 available at research.microsoft.com (1998)

3. Cusumano, M and Selby, R.W. How Microsoft builds Software *Communications of the ACM,* 1997
4. Cusumano, M and David Yoffe Competing on Internet Time – Lessons from Netscape and its battle with Microsoft *The Free Press (Simon and Schuster)* 1998
5. Dickman, Alan Designing Applications with MSMQ – Message Queuing for Developers *Addison Wesley,* 1998
6. Hammer, M and J. Champy Reengineering the Corporation – A manifesto for Business Revolution, Harper Collins, 1994
7. Henderson, Peter. Laws for Dynamic Systems, International Conference on Software Re-Use (ICSR 98), Victoria, Canada, June 1998, IEEE Computer Society Press
8. Ould, M. A. Business Processes – Modelling and Analysis for Re-engineering and Improvement, Wiley, 1995
9. Ould, M. A. Designing a re-engineering proof process architecture. Business Process management Journal, Vol3. No 3., 1997
10. Raymond, Eric S The Cathedral and the Bazaar available at http://www.tuxedo.org/~esr/writings/cathedral-bazaar/
11. Shaw M et al, Abstractions for Software Architecture and Tools to Support Them, *IEEE Transactions on Software Engineering, April 95*
12. Shaw, Mary and Garlan, David Software Architecture – Perspectives on an emerging discipline. Prentice Hall, 1996
13. Sun Microsystems, Jini ™ Software Simplifies Network Computing available at www.sun.com/jini
14. Warboys, B.C., M. P. Kawalek, I. Robertson and R.M.Greenwood Business Information Systems – A Process Approach, McGraw Hill, 1999