The Role of Adaptive Software in Business Process Re-engineering

Mark John O'Brien
Department of Computer Science, University of Nottingham,
Nottingham. NG7 2RD U.K.

Introduction

With their predilection for stasis economists are perhaps the only inhabitants of the contemporary world who do not see change as a constant ingredient in human systems. Certainly professionals in the computer industry positively embrace the exciting nature of new concepts and the manifestation of those concepts in technology. Equally, if somewhat more circumspectly, business managers feel compelled to alter their own beliefs to meet the concrete manifestations of new ideas. A point in space rich in interest is the intersection of the developmental trajectories of these two professional groups. But it has become rather more than a point in space, it is rather more an inter-twining of action and reaction, of need and satisfaction. The dynamics of change viewed from these two quite different perspectives and the mutually destabilising effects of this interaction form the subject of this paper.

The malleable nature of software has been, at one and the same time, one of its most endearing yet frustrating characteristics. Programmers have the freedom to alter behaviour and functionality of software through nothing more strenuous than editing a file using a text editor. But the very simplicity of the tools hides a complexity of structure which makes this apparently facile exercise one that is fraught with danger. To a large extent the subject of software engineering is concerned with monitoring and control of the change mechanisms available to programmers. The simple observation that the vast majority of software development is concerned with the maintenance of existing systems forcefully emphasises this concentration on the management of change of software systems. It is only recently that some researchers have viewed the possibility of change as something which can be harnessed positively where software is designed to change.

Whereas software artefacts can be objects of scientific study the business world allows no such dispassionate viewpoint. It is a world dominated by the present; practical considerations of the possible consume much of the total management effort available. Yet if managers recognise the dynamic nature of their environment they are not totally at the mercy of that dynamism. If they cannot alter the world as it is, and as it will be, they can at least alter their organisations to meet the changing circumstances of that world. At root, this is the raison d'etre for the set of ideas and concepts that evolved into the subject of business process re-engineering, or BPR as it is sometimes known. Despite the zeal with which BPR has been promoted in many respects it is merely another branch of the subject which goes by the name of business or corporate strategy. While it is true that BPR involves much low level tinkering with processes the adherents of BPR state that its real value lies in its global, corporate-wide application. The polemics of such arguments need not detain us here, BPR exists and with the prevalence of software systems, both BPR and the software must at some point mutually recognise one another.

This paper draws these two strands together. If BPR demands change as a matter of choice then any software systems embedded in those changing processes will also be required to change. The degree to which the software facilitates or hinders such changes can be taken as a measure of the quality of design of the software. This paper is concerned with software systems where the possibility of change has been designed into the systems from their very conception. In the case of software systems which have been designed to meet requirements specific to a particular point in time and space, BPR can be hindered by the relatively inflexible architecture of the software.
The structure of this paper is simple. It describes some of the issues relavent to adaptive software. This description ranges over both the technical and economic factors in building adaptive software. The aim is not to provide a detailed consideration of adaptive behaviour but rather a comprehensive overview. The second section is concerned with the conjunction of business process re-engineering and software, and the relavence of adaptive behaviour to this conjunction. These sections are essentially generalised and theoretical in nature so the final section describes a particular adaptive system that has been built for estimators in the construction industry; this last section highlights the theory and renders the ideas in a concrete form.

Adaptive Software

All software can be easily changed. This ease results from the basic alphanumeric form of all programming; in essence an editor is all that is required to change the textual basis of a program and through such a change the behaviour is altered. Yet the ability to change a software system does not imply that all software systems are adaptive. To call software `adaptive' requires a degree of volition on the part of the designers, it requires the use of particular design techniques and the use of particular mechanisms during the construction of the system.
Before proceeding any further a potential source of confusion should be removed: by adaptive software we do not mean software that has been built to implement any flavour of neural net. While neural networks have certain characteristics in common with more general adaptive software they are too narrow in their abilities. No matter how complex a neural net might be all that each net does during training is to create an algorithm to recognise the inputs fed to it during that training process. The behaviour of the network, while it may change over time, remains limited to a particular function.
More generally adaptive software monitors its own environment and alters its behaviour on the basis of that monitoring. This type of environment-system interaction leads to a form of evolution of the system. It changes with time as conditions change. Before proceeding to the implications of such changes the mechanics of adaptation must be described. These characteristics constitute a set of dimensions along which various forms of adaptation can be measured.
Perhaps the most important question to be resolved in the design and construction of adaptive systems is deciding on the agent of change. Simplistically, there are two possibilities: the system can automatically adapt itself or some form of human intervention can be the agent of change. The former is the more radical, and problematical, approach. In general system induced change can only take place within limits which have been predetermined by the designer of the system. This requires a degree of foresight on the part of the designer. The issues behind such foresight are

complex and have far reaching consequences for the current approaches to systems analysis. At the moment the aim of systems analysis is to elicit the user requirements and then using a process of reductionism distill these requirements into the functional description of a conceptually complete and internally consistent system. Systems analysis for adaptive systems requires a quite different approach, an approach where diversity is recognise d and maintained throughout the entire process: no reductionism takes place. The diversity in fact comes to form the `envelope of adaptability' marking the boundaries of the system's behaviour. This is not to say that the system cannot behave in manners not lying within this envelope, merely that this is what is designed into the system at the outset.

Where change is controlled by human actions it too can utilise the established envelope, but quite naturally the possible system behaviour extends far beyond this envelope. At the limit any of the forms of system maintenance can be viewed as adaptation. When dealing with human invoked adaptation the question naturally arises as to who is, or will be, performing the adaptation. Anticipating somewhat, in the adaptive estimating system described below the system designers had hoped that estimators themselves would have the requisite skills, yet during evaluation it was found to be beyond their capabilities. This despite major efforts to relieve them of programming in any form.

The next dimension to be considered in designing an adaptive system is one of time: exactly when is the system to be changed. Most systems involve a monitoring of the environment and when certain metrics reach critical values they trigger the agents of change into action. The design of these trigger metrics is a crucial element. Just as important, however, is their treatment by the system. A system cannot respond instantly, and continually, to changes in the environment. The resulting system would be extremely confusing for the users since its behaviour would alter from moment to moment. Thus some form of delay, or more precisely `momentum', must be embedded into the system. The system will only change as the result of a set of cumulative environmental events; moreover, the accumulation must be the result of a reasonably consistent and homogeneous collection of events.

The most common form of environmental events that are monitored by adaptive systems is the user interactions with the system itself. Thus a system may monitor the mistakes made by the user and as a result the error messages and help information supplied to that user would be couched in terms appropriate to the user. In such systems the user behaviour comes to determine the system behaviour.

Finally, in this review of the dimensions of adaptation a brief consideration of the mechanisms for embedding change must be given. In simple terms the primary method is one of `design deferral'. Design deferral means that whenever the system designer is confronted with a number of possible options to be included in the system he or she includes all the options and leaves the decision as to the use of one or other of the options to the agent of change. Given that most systems are implemented as run time executables perhaps the most common form of implementing design deferral is through the use of configuration files. A simple example of adaptation that is now available to most users is the setting of screen colours. In the past this was established by the designer and coded into the system; the users had no choice but to accept the colours selected by the system designers. But with design deferral the screen colours are determined by the values held in a start up file which is read by the system when it begins executing. It is revealing that

configuration files are becoming enormous, controlling a wide variety of possible system behaviour.
The use of configuration files is linked to a second method of achieving adaptability, namely the use of parameterisation. Again rather than explicit values being embedded into systems to determine system behaviour parameters are used to hold values which can be altered as the system executes. More recently the use of polymorphic behaviour in object oriented systems can be used to achieve a degree of adaptability. Finally the system may include non-compiled code segments which are interpreted at run time. As with configuration files these code segments can be edited while the system is running to alter its behaviour. A key issue with such code segments is the language in which the code is to be written; if it is intended for non-specialists to alter such code then it must be simple to understand.
Technically much of this goes against some fundamental ideas of software development. Take for example the idea of `referential integrity' which demands that when a block of code is invoked using a set of values, then if those values are unchanged on separate invocations then the behaviour of the code should be identical on each invocation. (This is the reason why software engineers steer clear of global variables. Global variables are the best way to undermine referential integrity.)
Drawing all these elements together a simple example might help to explain adaptive behaviour in software systems. The example comes from the most studied area of adaptive software systems: the user interface. The example is concerned with a simple menu structure where a number of options are listed one below the other. In a traditional non-adaptive system the ordering of the options is fixed by the system designer. The ordering of the options bears no relation to the individual requirements of the user. Thus in an adaptive system the system monitors the number of accesses to each option performed by a user; each number is a montioring metric. After a certain period of time, or number of accesses, the system is triggered to re-order the options such that the most frequently accessed options for a user appear at the top and the most infrequently accessed ones at the bottom. Over time therefore the system will create a number of personalised menus appropriate to each user. Indeed the system could go further an d begin to remove options from the menu which are never used by a particular user; this action reduces the perceived complexity of the system. In a more advanced system the links between successive options can be monitored such that wholesale reorganisations can take place which places linked options in close physical proximity on the screen. In such systems as this when the reorganisation takes place it must be signalled to the user who would otherwise be confused by the new, and unannounced, menus.

Software and Business Process Re-engineering

Business process re-engineering is either a fundamental set of techniques for the complete reconstruction of an organisation or merely a reworking of old concepts dressed up in new terminology. The difference is merely one of perception. Certainly, business process re-engineering is strategic in nature and can therefore be seen in the light of existing corporate strategy concepts. Moreover, the injunction to `obliterate' rather than `automate' (Hammer, 1990) suggests it is a direct lineal descendant of the ideas to change `effectiveness' rather than `efficiency'. In that particular dichotomy managers were encouraged to change what they did rather than merely improve that

which they already did. It is not the intention here to enter into this particular debate but merely flag the rather contentious position of BPR in theory and practice.
Also before proceeding no distinction is made in this paper between business re-engineering and process re-engineering; the former is all embracing and has a wide perspective whereas the latter is more narrowly focussed. Since the relationship between adaptive software and both of these manifestations of re- engineering is merely one of degree they will both be treated in the same way: hence the use of the phrase business process re-engineering. The only caveat to this is the example given below, an adaptive estimating system, does not display the cross-functionality which is often demanded in business re-engineering.
Whatever may be said of BPR it certainly involves a process of change. Since such changes are viewed from an organisational point of view they must necessarily, in general, involve changes to the entire environment and, in particular, to the embedded software systems. Herein lies the dichotomy of such software systems. It is usual for such systems to have been designed to satisfy the existing business processes. The software will have been fine tuned and developed so that it fitted an existing way of doing things. Such systems have therefore been implemented with a view to automating the existing processes. Equally, such systems will have been crafted in such a way that they are intolerant of change. Flexibility is rarely in the mind of software developers who are pressed to deliver systems which are designed to satisfy specific business needs. In an environment where change is taking place such systems can therefore act as a break on the change process itself. Rather than being an enabling technology, tradi tional software represents a significant drag on any forward movement. This problem is well understood by the proponents of business process re-engineering and it is not untypical to find authors stating that the software systems must be abandoned and new ones implemented. In the hard world of commerce this is not an attractive proposal. Senior management is only too aware of the costs involved in designing and implementing new software, and the thought of decommisioning large scale software systems is painful. It is at this point that the idealism of business process re-engineering comes up against the cold pragmatism of legacy systems.
Current software systems are largely designed to be unadaptive, which suggests that their role in business process re-engineering is largely obstructive. They represent an in-built barrier to change. But given the introductory comments with regard to change being a factor in all human endeavours the benefits of adaptive software need little explanation. And in particular the use of such software is positively beneficial when process re-engineering takes place. Thus the real benefits which accrue from adaptive software are not realised at the moment of commisioning but rather in the future.
Despite the disparaging opening comments on economists one economic observation needs to be made with respect to adaptive software in a commercial setting. The creation of adaptive software requires effort above and beyond that required of non-adaptive software. This extra effort involves cost. Moreover, it is an extra cost which cannot be recouped as soon as the software goes live. In effect adaptive software involves increased expenditure at the start of the software lifecycle but this is balanced by benefits at the end of the lifecycle. To put it mildly, this temporal mismatch of expenditure and benefit meets with little enthusiasm from the organisations involved.

An Adaptive Estimating System

All the above has been somewhat theoretical and general in tone. This section describes the construction of an adaptive system to support the estimating function and serves to provide a practical example (see O'Brien and Pantouvakis, 1993). It has to be admitted that the initial impetus behind the construction of this system was not to build a system which could be altered over a period of time. The aim rather was to create a system that could be adapted to meet the needs of a variety of estimators. Equally, the thought behind the system was to create a tool that could itself be used to build estimating systems. Such a tool proved impossible to build without direct reference to the functions of estimating itself; this in turn resulted in the tools and the system being completely integrated. In some respects the resultant system was greater than the sum of its parts, and the abilities of the system went beyond the expectations of the designers.

The system was built using dBase III+. Two reasons motivated this choice. First, the embedded language of this database management system is interpreted. This means that code can be altered while the system is actually running and the flow of control changed on the basis of previous actions. Secondly, dBase III+ is clearly a database management system which is designed to support the creation, modification and access of large scale data structures. These data structures in turn contain data values which can change over time. This suggested that the data structures to be stored in the database should be the very estimating system itself. Thus much of the behaviour of the system was defined by the code that was embedded in the database. The tools that were built to access the system were thus concerned with editing code held in database structures. This conjunction of requirements and capabilities proved a natural environment in which to build the system. It was hoped that the tools would be built to allow changes to be specified using high level constructs. The hope was that the estimators would specify what was required and the system would deal with how those requirements would be implemented. To this end the system was described as being `declarative'. In retrospect this proved to be a vain hope.

The agent of change in the system was designed to be the estimator. While the system is running the estimator has access to a special set of options which collectively can be used to alter estimating functions, menu structures, and reports. During evaluation of the system with estimators it was found that the advanced nature of the tools was beyond even the most computer literate estimator. To stress: this was not a failure on the part of the estimators, but one on the part of the designers. This failure was closely linked with the failure to produce a truly declarative system; the system required too much programming skill on the part of the estimator.

Since the agent of change was the estimator the need for triggers was neatly sidestepped. It was up to the estimator to decide when changes were required. Nevertheless the inclusion of monitoring metrics and triggers would not prove a major problem. If such monitoring were included the system would take on a far greater role in decision making. Thus, for example, if the system observed that a particular estimator displayed consistent decisions being made in allocating activities to bill items it could allocate those activities automatically. Wherever estimators took decisions the system could use metrics to monitor their choices.

The system supported a range of estimating techniques: unit rate, operational estimating, crude project estimating and so forth. Within each technique the calculations required were held in the database as functions enabling them to be changed.
The resulting system, while not designed to support business process re-engineering, turned out to be ideal in environments where changes, both large and small, take place regularly. Where the designers had envisaged an adaptive estimating system to suit the different needs of a variety of estimators, the final result proved equally adept at meeting the changing needs of one estimator as time progressed.
The introduction of a new concept at the end of a paper such as this is not usually a good idea but the lead into it is perfectly natural. One of the new management fads refers to the `learning organisation' which is in a state of continuous flux as it acquires knowledge concerning its own internal functioning and also knowledge on its environment. The monitoring capabilities of adaptive systems matches exactly this ability to learn as time progresses. The rather restricted aims of the initial adaptive estimating failed to take into account the wide range of application, with the result that despite the systems shortcomings it proved to be a developmental success beyond the hopes of the designers.

References

Hammer, M. (1990). Re-engineering work: don't automate, obliterate, Harvard Business Review, July-August, pp104-112.
O'Brien, M.J. and Pantouvakis, J.P. (1993). A new approach to the development of computer-aided estimating systems for the construction industry, Construction Management and Economics, 11, pp30-44.