Andrew K. Johnston
Questa Computing Ltd
"Coppertrees"
Forest Road
Effingham
LEATHERHEAD
KT24 5HE

01483 283408
andrewj@andrewj.com
Fax 07957 398508
www.andrewj.com

Strategies for Flexibility

Organisations need to protect and maximise the value of their IT assets. To protect against threats from business and technological change systems need to be *flexible*: able to change to support new functions, new workloads and new working environments. Flexibility does not happen by accident – it is usually the result of planning, forward thinking and adopting strategies known to enhance and encourage it. This paper presents some of those strategies.

I am indebted to the recent work of the CBDi Forum, particularly a paper by Richard Veryard[1], and to Stewart Brand's excellent "How Buildings Learn"[2]. The latter describes some of these strategies as they apply to physical buildings, and I have translated them into IT terms. I have freely borrowed from both sources for inspiration and the seeds of some strategies, but others are empirical, derived from my own experiences.

Flexibility has many dimensions, and a concept of "scope", which accepts some barriers to economically manageable change. Characteristics like robustness are fundamental to the design of good but inflexible systems, so this paper assumes that flexibility is focused on three dimensions: changing functions (adaptability), workloads (scalability) and working environments (portability), within a "reasonable" scope.

A strategy for flexibility is likely to contain several (or all) of the following elements.

Look Ahead

Building architects often practice what they call "programming": interviewing prospective stakeholders, getting their requirements, and building a product to fit those requirements. However, the world of architecture has started to acknowledge that too often this process fails because the requirements are either incomplete, mis-prioritised or have been superseded by the time the building is complete. This sounds horribly familiar to those in the world of IT.

1 "Business Flexibility – A Framework for Understanding, Planning and Achieving Flexibility" by Richard Veryard, published by CBDi Forum Limited, June 2001

2 "How Buildings Learn – What happens after they're built" by Stewart Brand, published by Penguin, 1994

A more successful alternative uses the technique of "scenario planning", in which the widest possible range of future scenarios is considered, and their impact and desirability assessed[3]. The strategy, developed in response, should be able to cater for all acceptable scenarios, and ideally reduces the risk of any undesirable scenarios.

Business strategies may indicate the possible dimensions of required flexibility, but may need translation into IT terms. Conversely, you may have to translate dimensions of IT flexibility into business terms for discussion with key stakeholders.

All other things being equal, a good strategy will maintain future options. It will also allow smaller incremental decisions, with lower risk as a result. For example, a "best of breed" strategy, but using EAI to isolate each system behind controlled interfaces, allows a choice now to be changed later. However, keeping options open should not be an excuse to paralyse progress by refusing to make any decisions.

Design for Flexibility

Don't tie things together too tightly. This "loose coupling" is particularly important where two items are at different stages of maturity, or are changing at different rates - in this case tight coupling may actually break the systems as they change. Asynchronous message-based integration with a "hub and spoke" architecture is an example of good practice. Layered architectures separate things with different rates or types of change in a similar way.

In a component or service-based architecture, it should be possible to make many components "plug compatible". Different components will support the same interface, defined independently of any single implementation. A given interface will support various operations and data types (polymorphism). As well as allowing each component to evolve separately, this introduces the ability to handle different types of input in the same process, or, conversely, use different components interchangeably in the same place.

A well-designed system will allow new or user-specific functionality to be added in separate, maintainable components, which can be migrated in a controlled way when underlying software changes. Systems which expose their object model and host a development environment such as Visual Basic for Applications represent the best practice. Worst practice is probably where a customer or vendor "modifies the package".

A flexible, maintainable building has "accessible services", for example running pipes and wires in raised floors and conduits. The software equivalent is to expose specific interfaces for configuration, management, test and diagnosis, and to make sure that the underlying system software is not so hidden or constrained that it prevents maintenance. This should become an important consideration in solution selection.

All these concepts rely on well-defined and stable interfaces between parts of the system. Where possible, such interfaces should adopt public, de-facto or open standards, which will enhance their stability and ease their description and use.

[3] See "The Art of the Long View" by Peter Schwarz, published by John Wiley & Sons, 1998

Flexible buildings are built from maintainable, adaptable materials. The "materials" of systems are their languages and system software. The easier these are to work with, the more flexible the products. Choosing tools and system software which are well supported by available skills will immediately enhance a solution's flexibility.

High levels of abstraction are desirable, for example encapsulating business rules in a maintainable rule table, rather than hard-coding them. This requires lateral thinking in analysis, and a deliberate policy to "find out what's common" rather than "find out what's different", which drives so many analysis efforts.

It may help to think of the data itself as "layered". Going beyond normalisation, consider a "separation of concerns" for data as with other elements of the system. For example, a flexible system will separate the business rules about "what" data from those about data validation, in turn separate from formatting information and again from the data items themselves. This will immediately indicate dimensions of flexibility and possible structure for the system's architecture.

The most flexible systems provide a toolkit with which the users can solve a whole class of problems. The best such systems have a powerful Graphical User Interface, allowing "on-screen" direct manipulation of objects representing elements in the problem domain – this is the concept of "expressive systems"[4,5].

Allow Room to Grow

Buildings favour flexibility when they have room to accommodate new activities and growth in the family or business which occupies them. Even if the building itself is currently small, flexibility will be enhanced by the potential for growth, for example a larger site. The same idea applies at finer levels of detail by making things a "loose fit", for example leaving room for new wires in the conduits.

Clearly a similar principle applies to computing hardware and infrastructure provision. It is usually a mistake, for example, to size hardware precisely to current usage levels. However, you may not be able to afford much extra capacity at the outset, and deferring some part of the expenditure brings at least cashflow benefits. In computing, where Moore's Law[6] applies to most hardware and infrastructure, the benefits may be even greater. The challenge is to choose or design an architecture in which it will be possible to add extra capacity at a later date, and then ensure that expected expansion continues to be viable and reliable.

Software should be designed or selected to minimise built-in limits on growth. Classical examples are restrictive coding schemes and hard-coded limits on sizing. However the idea of leaving room for growth "unfinished" can also apply to software. Typically, you tackle the high value and/or low cost requirements first, but design an architecture into which later functionality can be added. This may be a natural result of applying an "80/20 rule", where non-key requirements are deferred to ensure success for the core.

---

4 "Expressive Systems – A manifesto for radical business software" by Richard Pawson, published by CSC Research Services, 2000

5 For examples, see the Questa developments RelQuest and ConQuest described at www.andrewj.com

6 Moore's Law can be stated as "The cost of given performance or capacity halves roughly every 18 months"

Build Strong Foundations

Flexible buildings tend to have very strong and high-quality basic structures, which allow significant addition or modification without endangering the building's integrity. If a compromise has to be made, it is usually better to spend more on basic structure and less on "finishing". This suggests that software build processes should initially prioritise architecture over functional detail.

Experiment

Experimentation is vital as a means to explore requirements, validate technologies and as a source of new ideas. The problem is that uncontrolled it is a major source of IT fragmentation. While a building experiment is easily terminated by dismantling the temporary solution and building a replacement, many IT experiments hang on, as difficult to kill as an obdurate vampire. Uncontrolled experimentation also provides an excuse for those who want to "do their own thing" counter to an agreed approach.

Simply saying "nay" will not stop experimentation, it will just move "underground". The only sensible approach is to allow experimentation within a controlled structure and budget - an R&D function for the IS department may be a good solution.

Accept Change

Change happens. General von Clausewitz said "No battle plan survives first contact with the enemy", and an even bleaker statement can be made: "All predictions are wrong".

This is an important message to all stakeholders. All strategies, plans and designs have to be living documents, responding to a changing situation. Used well, it's not necessarily a negative message – it can be an important tool to focus on the need for flexibility, on allowing for the future, and on sensible and timely prioritisation.

Acknowledge and Communicate the Strategy

If the strategy for flexibility is to succeed, it must be understood and accepted by all stakeholders. Unfortunately most system modelling methods concentrate on a "snapshot" of how systems interact at a given time (the "synchronic" view), and on the single expected outcome of development. Instead, you need a representation of "change over time" (a "diachronic" view), and alternative outcomes.

Two documents thus become very important: the scenarios for future development (ideally in both business and system terms), and a "roadmap" showing how today's systems will evolve over time to those of the future.

A documented flexibility strategy should include a discussion of change, the scope of flexibility required and the strategies adopted to meet it. It may also need an explicit acknowledgement that infrastructure fit and provisions for integration, scalability and adaptability may be more important than raw percentage functional fit.

Conclusions

Change is inevitable, and some measure of flexibility must be built into any IT asset which is to have a useful economic life. Preparation for change may be even more important than a good fit to current requirements. The management of complexity and change are arguably the two greatest responsibilities of IT strategists and architects.

However, it's impossible, and certainly not commercially viable, to make any system infinitely flexible. Furthermore there are many burnt fingers where flexibility has been promised and not delivered, or where over-ambition has damaged a project.

It's vital to understand the dimensions and scope of flexibility required, which should come from scenario planning, risk analysis and similar techniques. Ideally the solution flexibility should match the expected flexibility in the business itself, but this is not always possible.

Flexibility should then be delivered by a set of explicit strategies. The ideas in this paper may provide a good starting point. Where the strategies impact on project cost, risk or timescale this needs to be properly acknowledged and supported, although in many cases the ultimate cost will be no higher than an inflexible alternative.

Flexibility is neither a magic wand nor Holy Grail, but it can dramatically enhance the value of IT over time. The key to success is thinking ahead.