Reviewing an architecture

Nick Denning
Chief Technical Officer
Strategic Thought Group

Management introduction

In recent articles for **MIDDLEWARESPECTRA**, *Nick Denning discussed the characteristics of an IT architecture. In this analysis he seeks to define a series of checklists — based around the concept of what an IT audit should be looking for — that may be used by organizations seeking to assess the appropriateness of any architecture it might currently have in place. In so doing he tries not to introduce new material, but rather to look at architecture from an alternative perspective, one of which is to use non-architects seeking to ensure the appropriateness of the current architecture to their particular business.*

The goal of this analysis is, therefore, to facilitate a process where an auditor reviews an IT architecture to:

- ***identify weaknesses***
- ***suggest remedial action.***

'If it ain't broke don't fix it'

Is there anything wrong? If everything is fine with the development processes, then read no further. However, if there are any problems with an architecture, then let us try to analyze these and to suggest remedial action(s).

Let me start by asking — what are the common problems apparent to many (most) businesses:

- **quality — are there too many bugs being introduced into production which are expensive to resolve and reflect badly on the quality of what is delivered?**
- **is the performance of solutions (applications) poor?**
- **is functionality, that was included in the initial specification, not properly, or fully, implemented in the final release?**
- **is development productivity insufficient to meet the needs of customers?**
- **does the evolution of any aspect of the solution have a major impact on the application code?**
- **are programmers worried about modifying certain areas of the application code?**
- **is change and evolution of the approach difficult and expensive?**
- **are individual responsibilities unclear?**
- **are solutions not intuitive out of the box?**
- **are initial user impressions generally not good?**
- **is documentation poor?**
- **is there internal 'sniping from the sidelines'?**

Customers are, as a consequence, unhappy. People know there is a problem. The issue is where to start looking.

Commissioning an audit

Essentially an architecture audit, in these circumstances, needs to understand the goals of the organization that is commissioning the audit and the possible outcomes desired. An audit is only commissioned in order to enable the organization to achieve a purpose.

The auditor and the 'commissioner' need to determine the specific objectives that must be met to meet the goals and achieve the outcome. Without this, any results will be worthless, or even worse.

Approach

The approach an auditor adopts in undertaking an IT architecture review is critical to the outcome achieved. What should the auditor be looking for and what should be the investigation approach required to achieve a positive outcome for the organization commissioning a review?

There are three high level areas that an auditor has to investigate:

- **people (does the organization have the right people and are they properly trained?)**
- **process (do the people know what they should be doing and are they doing it?)**
- **performance (are the processes being followed and is the architecture fit for purpose?).**

Before starting the auditor has to consider his or her preparation and planning. Much of his or her report will be revolve around fitness for purpose of the architecture. The audit study is not measuring against academic 'goodness' criteria but rather whether that architecture is likely to deliver what the business needs and expects — within the business cost envelope.

The auditor must, thereore, assess the information required to undertake the audit and verify that it not only exists but also that it will be provided. If it does not exist or will not be made available, this will impact the project approach and hence the Terms of Reference. (If an organization is not willing or able to provide information — such as the confidential strategic information necessary to obtain an understanding of the business requirements, the context as well as potential investment available and the rate of return required to make an investment — will be missing.)

Terms of Reference

From experience, I would that strongly suggest formal Terms of Reference (ToR) are written and agreed with the project sponsor. The ToR should define the objectives of the study and the circulation of the final report(s). In its widest form the architecture defines the business, so constraints need to be applied to the scope of the investigation and report. For example, agreeing a contents list for the final report may provide a straightforward mechanism to crystallize the scope of the report as well as who will see it.

An auditor has, therefore, to be extremely careful when agreeing his TOR, the form of the final report and the person to whom it will be delivered. Any observations that relate to organizational structure, organizational change and individual performance in roles within the structure may well be explosive and can probably only be delivered to a person of sufficient seniority to implement changes

(rather than to people within the structure itself). Again the report contents list may focus attention on what is needed and the auditor should request a copy of the organization tree to understand who the audience is, who might be implicated and who will be affected by the results of the report.

More than just an architecture

Many people see an architecture only in the limited terms of a IT technical architecture. They may well expect an architecture review to focus only on that aspect. This helps to constrain the scope; on the other hand it may avoid or hide any analysis of broader, critical issues.

Any audit report should also expect to make recommendations for addressing the observations made. If this involves changing people it becomes a dangerous brief again.

One approach might be to avoid a time consuming technical investigation and instead to propose an industry standard target architecture, to provide training so that the team understands the architecture. This involves them in identifying a plan to evolve the current architecture into this industry standard and will associate people with that architecture. It may, therefore, avoid a report focused on criticism.

On the other hand the organizational structure is vital to an effective architecture implantation. The study might envision a target structure. Clear direction should be given if the report is to include comment on the performance of the current team.

Thus the TOR becomes a critical project initialization document because it explains what:

- **will be covered**
- **will not be addressed**
- **the implications of any restricting of the scope.**

An effective approach may be to report in a limited manner, and identify in the conclusions the areas which need further analysis. A more senior business sponsor reading the initial report then has the option of commissioning further work reporting directly to him.

Indeed it may be helpful to explain, upfront, that such reviews may need a phased approach depending on the findings of the initial study. This is the same phased approach we, at Strategic Thought, advocate.

Baselines

I will refer a number of times to the concept of a 'baseline'. A baseline is simply a definition of the current environment.

At its simplest it is a single document that refers to all the other documents that are relevant to the current architecture or any other document that will be relied upon during discussions on any aspect of the architecture. It is an essential tool for the auditor because it should crystallize the situational information.

Ready to start?

You are now almost ready to start. But it is worth making a final check to ensure that the approach adopted is most likely to achieve the outcome you (the client) is looking for.

For example, there remain some further areas that the auditor must consider as the review progresses. In addition the auditor, or audit team, must be suitably qualified to carry out certain areas of the investigation. They must also be capable of recognizing where they may need assistance and if that assistance can reasonably be carried out by internal staff or if external assistance is required.

An important aspect is that the audit must stick to the project plan and not become side-tracked into other 'interesting' (tangential) areas. It must focus on delivering the agreed report which identifies the areas for further investigation (if required within the TOR).

Is this teaching 'granny to suck eggs'? While management consultants may have this aspect of customer management well tuned, IT consultants who understand the subject area may be less skilled in the client management side of this form of consulting. Consequently the client commissioning the analysis needs to maintain close sponsorship and monitoring of the review process and to ensure that the organization's goals are achieved. When the report arrives there should be no surprises.

Organizational structure

The current structure of an organization is important. The following should be investigated within the review scope:

- **an organogram identifying the organizational hierarchy and the extent to which this also reflects seniority, reporting lines and appraisal processes**
- **how the organizational structure maps onto business processes (and what are the business processes)**

- **whether there are terms of reference for each role, as well as some form of comparison as to whether the skills of people currently filling those roles are appropriate**
- **the extent to which individuals fulfil multiple roles.**

A possible proposition for an organizational structure, when developing a product might be as shown in Figure 6.1.

If a solution is being developed for use in house, I would suggest revising this as follows by adding an operations team to the product management team and merging it with the sales and support team (Figure 6.2).

The essential elements of this are as follows:

- **product management is responsible for identifying and prioritizing requirements working with the architecture team to construct a product road map and negotiating with customers to ensure that this road map meets their needs**
- **the architecture team is responsible for negotiating with the product management team to take prioritized requirements, to analyze the implications of implementing those requirements, to factor in time, to discard requirements which cannot be implemented, to plan a staged project implementation and to provide (back to product management) a product road map**

The architecture team should factor in time. It should also applies the 80:20 rule.

Once agreed, the architecture team will commission individual projects to be delivered by development teams:

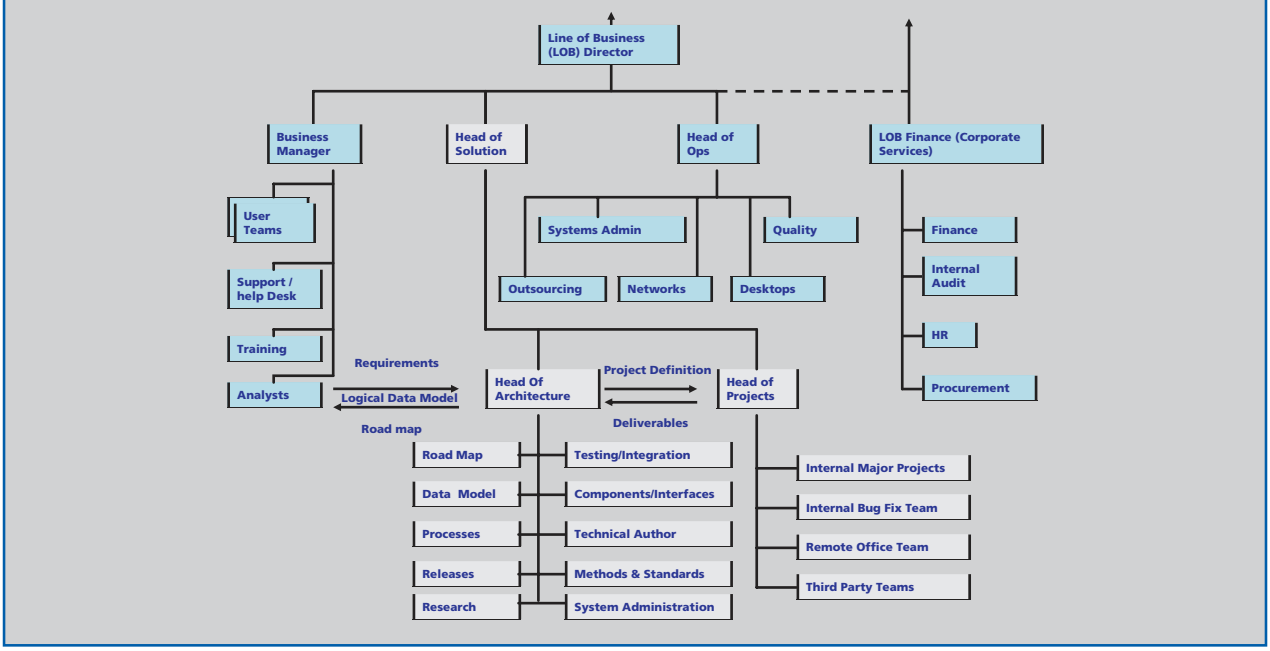
- **development teams will be provided with specifications (which must be implemented 100%), design approaches (which must be followed) and acceptance test criteria (which must be satisfied)**
- **the deliverables from each project will be provided back to the architecture team who will integrate them into the solution.**

Thus the architecture team is a substantial team requiring its own organizational structure as shown in Figure 6.3)

When we look inside the architecture team we see the need for the following:

- **development of the architecture road map, which extends the product road map; the**

Figure 6.1: Line of business organization (example)



product road map defines to customers where capabilities will be delivered while the architecture road map describes how all of the individual architecture changes will be implemented over time

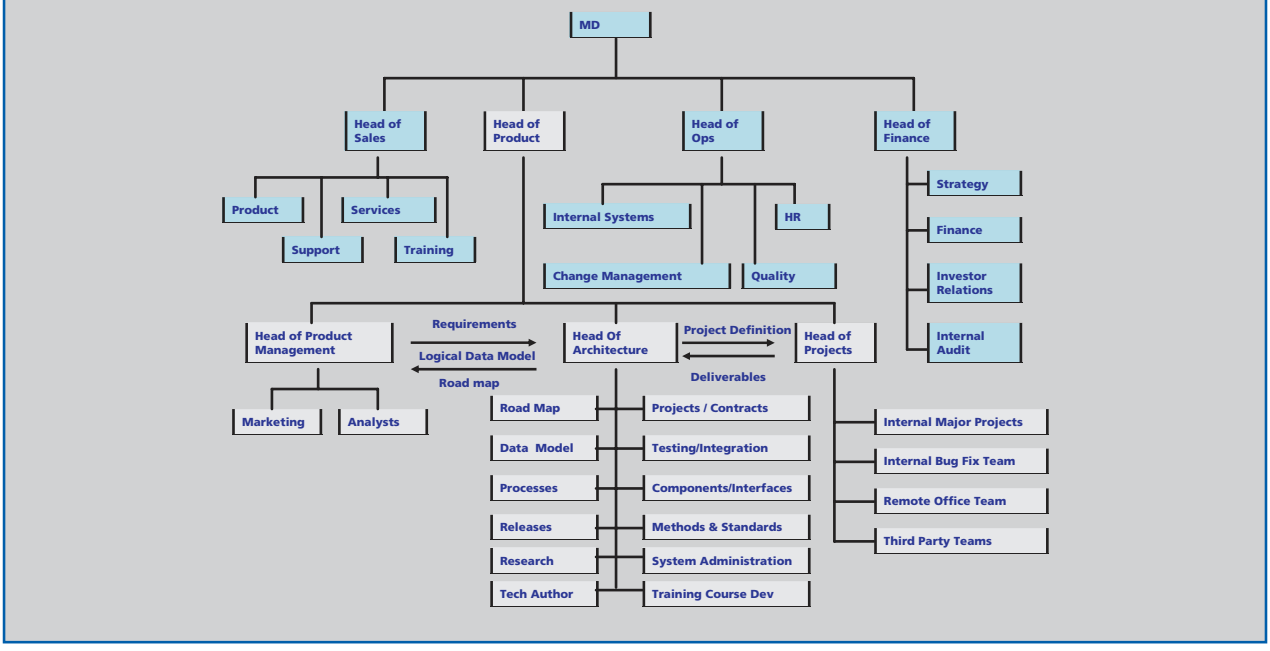
- methods and standards to ensure that individual development teams tasked with work have an appropriate framework consistent with the architecture — without having to know the architecture in detail
- the business process owner must understand the high level capabilities that must be delivered and the processes required
- the interface owner must take responsibility for the hierarchical decomposition of the solution into components plus the definition of the interfaces to those components
- the data model owner must agree the underlying data required to support the business processes and the shape of sub-schemas and data transformation to support each of the solution components
- the programme manager/contract manager must manage work being sub-contracted to development teams
- the integration and test manager must take responsibility for managing the solution regression test environment, for acceptance testing of the deliverables provided by individual projects, integration of the components into the solution and regression testing of the overall solution prior to release
- (optionally) a research role can be made responsible for the investigation of new technologies, the introduction of appropriate technologies into the architecture road map and working with methods and standards to ensure that the development approach is sufficiently well defined to be provided to individual development teams
- training
- technical authoring and documentation
- infrastructure systems management
- product packaging and release.

Governance — this should be business as normal

It is important to investigate the business processes that are in use by an organization to deliver its current product(s). To the extent that the following are implemented, and whether there an audit trail, will enable subsequent analysis:

- are terms of reference available for each role? (investigate any contradictions,

Figure 6.2: Product organization (example)



inconsistencies, overlaps or gaps in responsibilities that need to be addressed)

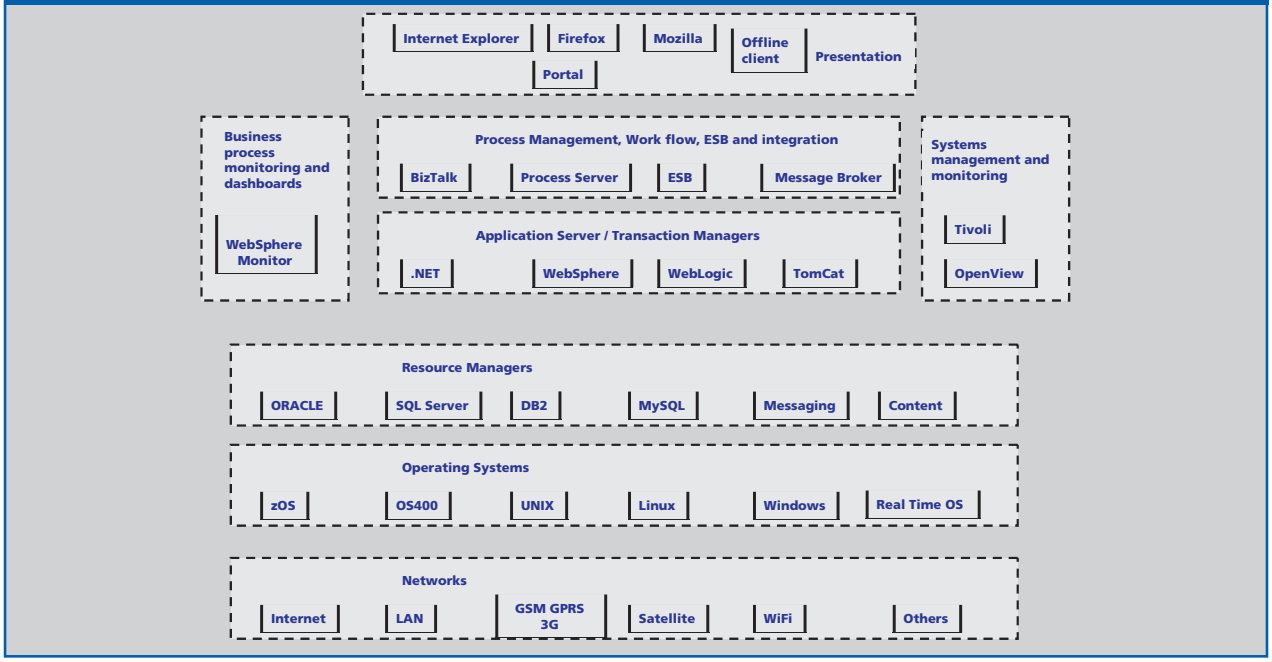
- strategy and planning: what evidence is there that a formal planning process has taken place, has been communicated and has been signed off by all concerned?
- customer needs: have these been captured and considered?
- requirements capture — how have customer needs and the strategic objectives been translated into business requirements?
- how have requirements been incorporated into the architecture and have they been incorporated partially, fully or postponed plus have these been reported back and agreed?
- has the impact of new requirements for the solution been adequately considered, the risks analyzed and the changes required allocated to specific projects?
- have past projects adequately implemented the requirements?
- have business users been informed during the planning, implementation and release stages?
- have the implications of changes for users been fully considered, planned, communicated and supported through consultancy, training and conversion tools?
- does regression testing adequately cover the new capabilities and is there evidence of acceptance tests being completed?
- has performance and scalability testing been carried out and where are the current hot spots, road blocks and limits to scalability?
- has the maximum capacity of the system — in terms of users, transactions per second and data volumes supported – been assessed?
- were the deliverables from individual projects delivered to the agreed acceptance criteria?
- were capabilities delivered to customers within an appropriate timeframe?

Governance — change management

An important aspect of assessing the architecture relates to its ability to accommodate change. Change arises for any number of reasons.

Technology products are often being upgraded. The impact of underlying technology upgrades can be significant, particularly when based on Microsoft technology. Moving from the Microsoft DNA to its .NET architecture will ultimately require most of the code to be re-written. With J2EE solutions, major upgrades are more frequent, API's are deprecated and replaced so a continuous

Figure 6.3: Technical structure



program of refresh must be planned. Can the architecture deal with these changes?

Changes in implementation patterns matter. For example, the advent of 'portal servers' is generating a momentum to include all user interfaces within a portal framework. The JSR 168 standard provides an open systems portal approach.

Limiting architectural dependence to a single supplier is an another example of why an architecture must support multiple technologies concurrently. To what extent can the current architecture support multiple:

- **RDBMS?**
- **operating systems?**
- **application servers?**
- **portal technologies?**

Decomposition into functional components with wrappers provides a mechanism for achieving this. The use of open standards — such as Web Services on the one hand and JDBC on the other — offers a mechanism for components implemented in different technologies to interact. Using generic languages — such as Java or C/C++ — which are supported widely across all likely platforms, enable base code to be ported. Confining code specific to individual technologies into technology specific libraries wrapped by generic interfaces ensures that the scope for non-portable code is limited and separated from any business logic. Libraries can then be ported between technologies.

Interface design need to minimise the degree of change to an interface when a change is required to a component accessed through an interface. Placing interfaces under configuration control, providing version control of interfaces and maintaining interfaces for a number of product releases, enables applications that invoke interfaces to continue to operate when new versions of an interface are included. (Provision of separate schemas, for individual components, prevents the need for wholesale change across an application as a consequence of a data based change to support one component is essential.)

Development productivity when modifying existing code or adding new logic must be optimized. A key aspect of adding a new sub-system is to minimize the changes required to other components, whether they be invoked from the component or invoke the component being changed.

A rigorous test framework should be mandatory. No solutions are immediately perfect, and some changes will impact the overall solution. The ability successfully to make major changes will be significantly enhanced by a regression test environment that will identify any failures across the system that arise. Impact analysis of changes is also mandatory. What tooling is used within the architecture to enable the impact of proposed changes to be identified before work starts?

Components of an architecture

If you ask an architect for the documentation set that defines his or her architecture, what would you expect to see? The following are a list of some of what I would expect to see:

- **one or more requirements documents that should be provided by the product manager or user representatives; these should be worked into a prioritized list of requirements and identify the timeframe and cost envelope within which the requirements is of value**
- **a road map generated by the architecture team, which defines the capabilities to be delivered and the timeline of delivery**
- **a project plan, created by the architecture team, which is used to identify the individual projects, their deliverables, timelines and costs; this has much more detail that the road map and will include non-functional projects such as upgrades to software**
- **baseline, constantly updated, summaries of the current status of the architecture**
- **a description of the target architecture**
- **a technical architecture that defines the individual product components which make up the baseline (and possibly the target architecture) and the factors that relate to the use and deployment of the relevant software products (from databases to application servers to portal products to management and monitoring products and including development tools such as compilers, modeling tools and software configuration tools)**
- **business process definitions, ones which describe the high level business processes supported by the solution**
- **logical data models, that describe the data held by the solution to support the business processes (this is part of the essential communications between the product management/user team when analyzing requirements and creating the product road map)**

- a functional component architecture that describes the functional elements of the solution at a high level, and how these are to be assembled into the solution
- interface definitions which describe the detailed interfaces between each of the functional components (and, effectively, become the requirements definition for component design and building)
- component designs which describe how each component has been designed and implemented (by calling lower level interfaces and appling business logic to provide callable interfaces by which each component can be invoked)
- methods and standards documents which lay out the styles or patterns for applications, the way such applications will be constructed and the use of the development environment to optimize productivity and quality of the code delivered (this should also include review procedures providing the framework)
- a development environment description that sets out how the environment is set up, how the software configuration will be managed, how builds will be performed and how software will be released into any test environments and ultimately into production.

Measures of 'goodness'

The following sub-headings provide an ad-hoc checklist of areas to investigate as general points of 'goodness'. It is likely that this list can be expanded over time:

- vulnerability to a single supplier: this relates to the flexibility of the solution to limit the risk of a supplier going out of business or raising prices (if software products are interchangeable then it may be possible to offer the solution on different vendor products to meet client needs)
- use of industry standard products: this is generally good — because the integrity and quality of the leading products is likely to be high, and (probably more important) the supply of skilled people is higher
- code metrics: certain aspects of coding practice enhance significantly the readability, quality, vulnerability and performance of code (one entry/exit point, no GOTOs, tightly cohesive, loosely coupled, checking of boundary conditions, consistent user interface design, etc.)
- solution adoption, configuration and flexibility: architects must work with project management teams to determine how any solution can be designed to grow with the customer to facilitate adoption; on initial deployment the customer may be a novice but, as the level of expertise grows, the user may wish to have access to increasing capability (the danger is that, without a configuration mechanism, a complex solution is the only one available and this is just too confusing for novice users so the solution capabilities are never fully exploited)
- openness: to what extent has the solution been architected to incorporate open standards to facilitate use of industry standard components and to enable porting of technical components to new platforms
- openness to new ideas: 'no' is terrible while 'not now' is better and 'let us consider where this should be in the road map' might be a far better approach to capturing new ideas and factoring them into the approach
- avoidance of technological zeal: if the architect starts to advance technology for technology's sake (without placing it within a business context) and cannot articulate the pros and cons, then there is a problem; if an architect arrives in a position where his or her reputation is staked on a particular approach then — by definition – it is time to change the architect regardless of the situation (the architect has to remain unbiased)
- avoiding IPR being locked in a single person's head: if your document baseline is 'thin' then there is no basis for communication between the team once it grows in size — plus there is vulnerability to staff loss and no accountability ('if it is not written down it does not exist')
- separation of roles: in a small team a person may do multiple roles, but some roles cannot be done by the same person without one role suffering; the architect responsible for the interface definition must not design the solution delivering the interface or it will be corrupted
- dynamic tension between teams: if it is all too cosy, how are new and innovative ideas going to be raised and considered (or are they being suppressed)?
- individual project size: projects need to be small, and have end points after which the teams can be restructured; if a project lasts

- **more than 3 months, break it into sub-projects**
- **development and quality metrics: the number of bugs, identification of sources of bugs and time to fix them must all be tracked and managed**
- **a formal test and bug detection framework: is there a regression test environment?**
- **an ability to identify the causes of bugs: can the source of bugs be identified?**
- **scalability: some solutions cannot scale up for any number of reasons; this must be understood, whatever the reason (too busy to train new team members, only certain people understand this code, etc.)**
- **management of multiple teams: if you cannot decide something in your team, how many levels of management to you have to traverse before you can make a decision — flatten the team?**
- **write-once documents: too many documents are written but never read (documents are often written for 'form' after they are useful); define the document baseline and make documents live so that they accurately reflect any situation**
- **defined patterns, methods and standards: build a screen doing 'this' built like 'that' simplifies the specification process.**

Resolving problems

When auditing an architecture you should expect to find some things wrong, for there will always be room for improvement. You need to be clear about the quality of the current approach and the relative importance and urgency of the proposed improvements.

It is likely that the auditor will need to provide at least two reports:

- **the first will describe the full range of findings, issues and recommendations for resolution, almost certainly in multiple stages; it is highly likely that this will be 'explosive' in most organizations but letting the organization 'have the whole 9 yards' (a full belt of ammunition in one burst') will waste the ammunition and kill the whole operation or the messenger**
- **the second report is one which can be distributed widely and will form the basis of the initial plan for change.**

Having reviewed all aspects of the architecture the auditor has to make recommendations for resolving the challenges identified. What are the principles that need to be followed:

- **Dale Carnegie said (among other things) 'never, criticize, condemn or complain'; 'give lavish praise' and 'refer to others' faults obliquely'**
- **Machiavelli in the Prince said 'never start a war unless you can win, and when you do, destroy the enemy so utterly that they can never get up'**
- **Sun Tsu said that the essential element is patience.**

Ordinary mortals just have to work out how to fix the problem. The following principles might be helpful:

- **reinforce success; do not punish failure**
- **evolve, evolve, evolve; do not break anything and start again**
- **adopt parallel strategies, both top down and bottom up.**

People are the most important asset. The objective has to be deliver leadership by ensuring the strategy — along with supplying the management to help people to obtain the most from themselves.

This can be translated into the following potential actions:

- **put in place a culture of change by constructing project teams, breaking them up at the end of the project and reforming different teams with different team leaders to meet new challenges; this avoids having to sack anyone who is obstructing progress and allows them to be re-assigned**
- **avoid making permanent appointments: where you can, always ensure that there are two people doing comparable jobs to encourage and stimulate performance and avoid dependence on a single person (if you cannot do this, time-limit any appointment)**
- **be consistent so that even the best guys are re-assigned; this means that new ideas can be heard from new people**
- **If someone is failing do not fire them but help them by reducing their load until they are able to achieve their objectives; above all do not add new people into a failing team, but find another team leader and start a parallel team — do not reinforce failure**

- **give clear terms of reference to all staff and translate these into monthly objectives**
- **be wary of any strategy involving starting again from scratch; this is a siren call but usually means that the full scale of the problem has not been understood (there is almost always an evolutionary strategy — so find it)**
- **introduce a common vocabulary and have it adopted so that all use have common terms.**

With regard to the content of the architecture, research how other people have addressed their architectures. The culture of the organization also comes into play here. Some organizations — particularly those that create value though exploitation of technology — are prepared to make innovative choices. Others want to take a more conservative approach to implementation:

- **consider the adoption of a range of technology products to support the architecture such as the new integrated development environments (IDE) and modeling tools**
- **when determining how to change peoples' roles, plan an evolutionary path to achieve this.**

An example technical architecture

Defining a technical architecture is really about selecting the software products that support the solution, the methods and standards that explain how to exploit the architecture and to manage the evolution of the overall architecture to facilitate the use of multiple products of the same type or to be able to switch out a product in favor of a competing product in the future.

The products actually selected will depend on the solution being created and the environment within which the solution must operate:

- **operating systems: a product should ideally be portable across Windows and Linux, and potentially therefore support most of the common flavors of UNIX (it may be acceptable for a bespoke solution to run on a single OS)**
- **virtualization technology: mainframe-like VM technology is now available on Intel platforms (using VMWare from EMC and a Microsoft Windows specific alternative); such technology is extremely important to consider because it provides the ability to have multiple virtual machines on a single system in the event that one ends up with a mix of products that are incompatible in a single installation (it also facilitates DR/BC solutions)**
- **the RDBMS dimension: any product solution needs to be able to support multiple RDBMS products or at least follow programming standards that facilitate porting between RDBMS easily; upporting ORACLE and SQL Server are musts, with DB2 a candidate and MySQL, Cloudscape and other open source RDBMS products as possibilities**
- **browsers: will the solution chosen support only Internet Explorer, or must it consider Firefox and others?**
- **application Servers: the most common ones are WebSphere or .NET with a number of open source products available such as TomCat, JBOSS and Geronimo plus other vendor solutions such as WebLogic**
- **presentation: portal technologies — such as Sharepoint and WebSphere Portal Server are the primary candidates, with a JSR 168 Open Source portal product another possibility; my own view is that a portal is an essential aspect of integration at the glass (however, where a more classic stove pipe application is still required, the use of Windows SMART Client and solutions exploiting in Client .Net or J2ME local application technologies — for instance, when writing offline applications such as those to go on a mobile phone — may be needed)**
- **process management: adopting a process layer makes sense and portal technologies are starting to include such capabilities (although a separate specialist work flow product may be required)**
- **Service Oriented Architecture (SOA) and Enterprise Bus (ESB) technology may be important; there are a number of these products, with probably the most important being BizTalk and the WebSphere ESB**
- **content management: again portal technologies are starting to include a content component but content requirements may need basic content management supported by document management and records management**
- **language technology: the primary candidates remain Java, C or C++ in the open systems area or J#, C# and ASP.net in the .NET environment; while Java and C# are the most popular, I suggest strong consideration of J# to facilitate common skills between developers (though C# is most widely used and may receive more investment from Microsoft in the future)**

- **application integration approach: adopt stateless routing and transformation between incompatible interfaces as being an essential capability of this element of the architecture**
- **system management: decide how to monitor the state of the system and make configuration changes or control components of the solution**
- **business monitoring and reporting: decide how to analyze the data within the system on a real time basis through a dashboard or via reports; the reporting system may be one of the most critical elements of the solution**
- **design styles: decide whether you are following a middle out approach for SOA or a top down approach for a classic application**
- **implementation patterns and examples: decide how each type of application (dashboard, work flow, data entry, asynchronous message processing, compensation logic, as examples) will be implemented**
- **definition of the open standards that will be adopted: decide how these will be exploited**
- **methods and standards for interfaces between technologies, in particular open standards: this element of the technical architecture is critical for interoperability between technologies**
- **methods and standards for facilitating scalability and high performance: decide how to partition and where to place code to minimize the number of calls across the network (avoid 'chatty' applications), the volume of data transferred across by each call and the overall latency of any application response through network overhead**
- **methods and standards to facilitate development productivity and solution quality: determine how to build a solution that can be debugged during development and regression tested after integration as simply and reliably as possible**
- **Quality of Service requirements (QoS): decide how to deliver a range of quality of service options within the network (vitally important and regularly overlooked).**

This is not an exhaustive list. Each architect must consider carefully the elements that are required to support each organization.

An example target architecture

In this section I identify the elements I would expect to see in a target architecture, with a brief explanation of the contents of each element:

- **high level business processes; these must represent the target business requirements**
- **a logical data model; this should describe the high level data model needed to support the target business requirements**
- **a functional or component architecture: this should describe each of the functional blocks of the solution, what capabilities it provides and which other components does it exploit**
- **interface definitions; these should describe the interfaces between each of the functional components of the solution at a high level**
- **a technical architecture (as described above).**

Thus the target architecture is made up of a set of non-trivial documents. It is relatively straightforward to document a template for the technical architecture (as I have suggested). However the functional architecture is the most important element of the architecture because it breaks down the solution into manageable components separated via interfaces on which each component can depend without needing to know the elements of the underlying components.

A major architecture is too complex for any individual to hold consider all details at anyone time. This approach is, I believe, the only viable approach for breaking the problem space into independent sub-components. A good functional architecture will facilitate the ability to have alternative functional components and the opportunity to select and assemble a range of specific solution by bringing together the architectural components in different configurations to support alternative quality of service needs.

The interface specification defines the inputs and outputs to each component. It provides the requirements specification for the designer of each component.

Finally, the target architecture will evolve over time. Does its structure lend itself to evolution and how can the evolutionary path be presented? How will the current architecture continue to support and embrace elements that were delivered and are now in operation that complied with a previous version of the architecture?

Auditor review

In reviewing the documents the auditor therefore needs to consider:

- **has the architecture been constructed so that it can be understood?**
- **are the individual components loosely coupled (are they independent of each other except across the common interfaces and could they be swapped out)?**
- **can a design be constructed with the only inputs being the interfaces?**

When the architect gets down to the detail, the most difficult aspect of independence is likely to be managing transaction boundaries, state and persistence of the data within a common database underpinning the solution.

For example there is probably a requirement for a hierarchy within an object model to provide:

- **transaction management**
- **data access and persistence**
- **stateless transformation logic**

and hence to avoid all three elements becoming intertwined within a procedural approach.

The challenges of partitioning were discussed in various previous **MIDDLEWARESPECTRA** Reports. In particular, refer to the consideration of:

- **General Business Objects (GBO) within a component**
- **Application Specific Business Objects (ASBO) to provide component interfaces within an architecture.**

Management conclusion

Architecture is almost always a people-based challenge. Have we the right people and are they being well managed and effectively led?

The fundamental aspect of an IT architecture is change management. As Mr. Denning argues, there are four critical elements for an architecture:

- ***requirements — what we need to deliver***
- ***baseline — where we currently are***
- ***target (architecture) — where we are going***
- ***route map — how we are going to get there.***

"I wouldn't start from here" but we have to, and the skill of the product manager is working out where 'here' is and then how to move forward effectively from here. Remember that if an approach is perfect, or 100% right, then it is likely too expensive. You could have done it for less and accepted some imperfection.