Extreme Architecture

A Minimalist IT Architecture Framework

Phil Robinson
lonsdale@iinet.net.au

Floris Gout
floris@floris.com.au

Abstract

IT groups have a pressing need to identify those things worthy of their attention. In this paper, the authors present an Information Technology (IT) Architecture framework that draws inspiration from legislation, enacted in 1996. The paper encourages a minimalist approach to IT Architecture by exploring a number of *extreme* points of view.

The IT Architecture framework is described using a single, uncluttered diagram. The framework is simple to describe and easy to recall. The diagram organises nineteen different elements that, taken together define an IT architecture. The authors suggest that these nineteen elements are the things that IT groups need to keep in focus.

Using the framework, an IT Architecture can either be fully defined in single planning project, or gradually fleshed out over a period of time. The simplicity of the framework helps to foster a shared understanding between IT groups and the business areas which they support.

The framework provides a point of reference for IT management, IT project managers, developers and operations staff frequently charged to *do more with less* in these lean times. These groups will find that the framework can offer them a much needed *lifeline*.

Authors

Phil Robinson and Floris Gout have consulted to a number of organisations and across a range of industry sectors. They have worked together on various IT-related planning projects. This paper represents a major collaborative effort that organises not just the ideas of the two authors but also the many inspiring people they have worked with.

Introduction

The framework we are about to describe has evolved slowly, over a period of time. It has been heavily influenced by the consulting assignments we have undertaken for clients across a range of industry sectors. Along the way we have of course, been influenced by other author's ideas about IT Architecture. We acknowledge that we have learnt much from the best thinkers in the field. In presenting *yet another framework* it is our hope that we have not *reinvented the wheel* but rather borrowed the best ideas and combined them with a healthy dose of experience in order to create a practical tool that is really useful.

We have called our framework an *extreme framework*. We use the word *extreme* as a relative term in relation to other architecture frameworks. We believe that we are justified in calling our frame work extreme because in relation to other architectures:

- It advocates a pragmatic *middle path* between the extremes of perfection and chaos;
- It is easy to describe;
- It encourages an agile approach to architectural work products;
- It unifies a number of disparate disciplines; and
- It offers a simple, consistent view to the various parties involved in the management of IT resources.

Buildings

In his well known essay on software architecture, Frederick Brooks (p42) cites the building of the cathedral at Reims as a triumph of architectural vision [1]. According to Brooks, eight generations of builders sacrificed their own ideas in order to maintain the design integrity of the cathedral. In describing the cathedral, Brooks states:

> *…the joy that stirs the beholder, comes as much from the integrity of the design as from any particular excellences.*

Brooks offers the triumph of Reims as a sort of *Holy Grail* for software developers. However, in a less well known article titled *Big Ball of Mud Pattern* [2], the authors describe what they claim is the most frequently encountered software architecture. They use the metaphor of a *shanty town* to describe the architecture and point out that shanty towns are:

> *…built from common, inexpensive materials and simple tools, using relatively unskilled labour.*

The authors list a number of reasons why this architecture is common:

- Pressure on project schedules and cost;
- Inexperience, lack of skills and high staff turnover; and
- The complexity and rate of change associated with the problem domain.

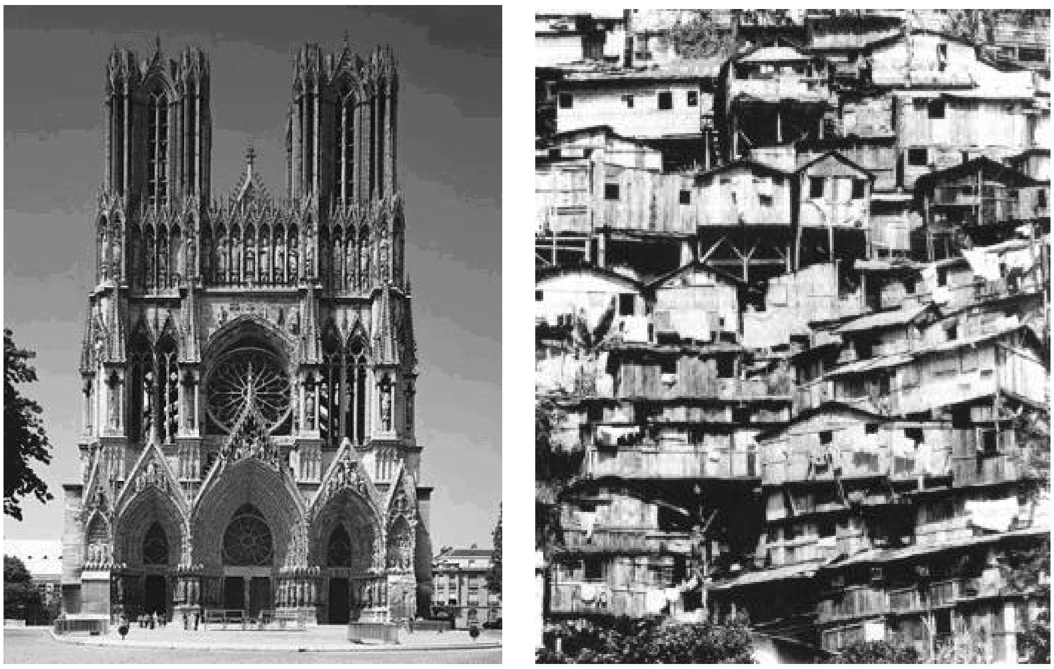

Figure 1: Software - Reims Cathedral or Shanty Town?

When the photographs of Reims cathedral and a shanty town are placed side-by-side it is obvious that they represent two quite extreme views of architecture. Confronted by these two extremes, we find ourselves naturally drawn to what many call the *middle path*. In doing so we are in good company. The Buddhist faith advises that:

> *…avoiding the extremes, gives vision and knowledge and leads to calm, realization, enlightenment…*

Systems

Although useful at a superficial level, the building-software metaphor can only be taken so far. Buildings are static structures that interact minimally with their environment. In contrast, software systems are by very nature, dynamic and interactive. The design integrity of a building is determined by factors such as geometric proportion, structural strength and the materials used for construction. In comparison, the design integrity of software depends on factors that are more abstract and more difficult to measure.

In addition to its inherent complexity, software is often part of a much larger, and potentially more complex human activity system. A detailed understanding of the human activity system is nearly always a prerequisite for a proper understanding of software requirements.

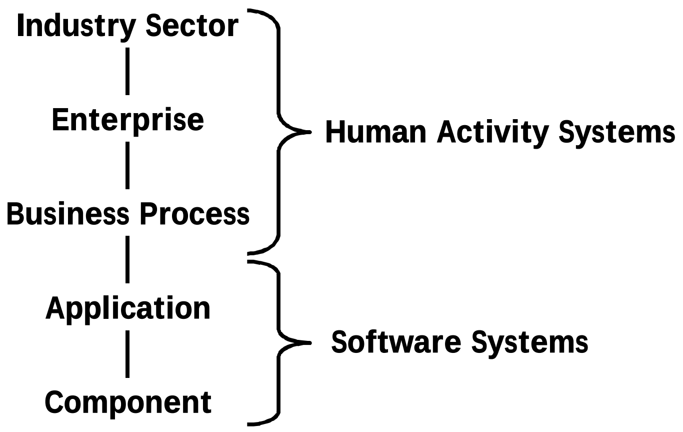

Figure 2: Hierarchy of Subsystems

Figure 2 shows how, human activity systems and software systems can be organised into a hierarchy. Industry sectors sit at the top of the hierarchy. These consist of a group of enterprises that are all engaged in a similar type of activity. At the next level, individual enterprises perform a range of activities that are often grouped into a number of business processes. Software applications are designed to automate and support these business processes. This is reflected by their placement at the next level of the hierarchy. Finally, the software components used to construct applications, are at the very bottom of the hierarchy.

Systems organised into a hierarchy such as the one shown in Figure 2, are often nested one inside the other. Nested systems are only able to interact with the system that contains them and any other systems that share the same container. This implies that all interactions between say, business processes must be restricted to the enterprise that owns them. Applying similar reasoning to software systems would require a software application to support just a single business process. It would also mean that only those people working for the enterprise would be able to interact with the application.

While some of these restrictions may have been valid in the past, most do not apply in a modern business environment. For example:

- It is already common for business processes to be outsourced. When this happens it means that a number of enterprises are involved in a single business process.
- It is becoming common for people who are not employed by an enterprise, such as customers, agents and suppliers, to interact directly with its software applications.
- Modern software development techniques emphasise the creation of *reusable components*. The goal is to reuse a single component a number of times in different software applications.

In view of the above, a more realistic way to view human activity and software systems is to regard them as a number of independent, overlapping systems. This view as illustrated in Figure 3 below.

Independent, overlapping systems are able to interact with each other at all levels of the hierarchy. For example, a software application might automate aspects of a number of business processes. Some of the business processes might span a number of different enterprises. The same software application may be required to integrate the information flows of a number of different enterprises. A software component may be designed to provide generic services used throughout one or more industry sectors.

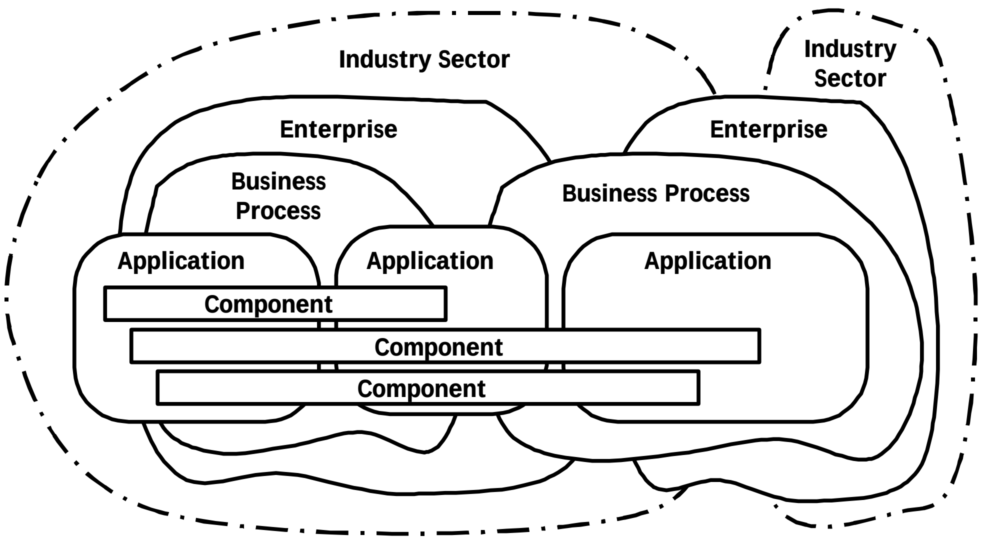

Figure 3: The Independent and Overlapping Nature of Human Activity and Software Systems

This view of human activity and software introduces a strong requirement for *interoperability* between individual systems Interoperability can be defined as:

> *…the ability of a system to successfully interact with other, specified systems.* [6]

Interoperability is one of the major factors that can be used to assess the design integrity of a software system. Other attributes of design integrity include factors such as:

- extensibility;
- scalability;
- re-useability;
- portability;
- security;
- reliability; and
- performance.

Architecture

High levels of design integrity are unlikely to be found in software systems that are developed in the same fashion as shanty towns. As the builders of Reims knew, to achieve good design integrity, it is necessary to have an architectural plan and stick to it.

A clear and simple definition of what constitutes an architecture can be difficult to achieve. We like this definition of building architectures that was provided by Ean MacDonald[3], a retired architect.

> *Architectural design is the simultaneous resolution and solution of the various architectural problems; that can include location, aspect, and prospect, sun, wind, and weather, materials and method, finance, function, and form, to which may be added a dash of flair that can make a structure work of art.*

An equally appealing definition of IT architecture was offered by Jim Sinur of the Gartner Group at the 1996 ZIFA Annual Forum.

> *If you can implement a drawing in more than one way, you have architecture. If you can implement a drawing in only one way, then the drawing contains exact specifications for instantiation, and you have a design.*

A more formal definition can be found in legislation. In February 1996, the US Congress passed a piece of omnibus legislation; The National Defense Authorisation Act for Fiscal Year 1996 [4]. Division E (the last 20 of 500 pages) of the legislation is known as the *Information Technology*

Management Reform Act of 1996 also known as the Clinger-Cohen Act. It offered a formal definition of IT Architecture that has now passed into law.

> *An integrated framework for evolving or maintaining existing information technology and acquiring new information technology to achieve the agency's strategic goals and information resource management goals.*

The purpose of the Clinger-Cohen Act was to ensure that the US taxpayer received value for money with respect to IT purchases.

The Act specifies that the agencies must support the goal of maximising value by assessing and managing the risk associated with IT acquisitions. It defines the roles and responsibilities of agency Chief Executive Officers (CEOs) and stipulated that each agency had to nominate a Chief Information Officer (CIO) with direct access to the CEO and other agency management.

A key aspect of the Clinger-Cohen Act's definition of IT Architecture is its reference to an *integrated framework.* The word framework is frequently used in relation to IT architectures. A framework describes an underlying structure that can be used as the starting point for developing an IT Architecture.

Perhaps the best known architecture framework is the *Zachman Framework for Enterprise Architecture* proposed by John Zachman in 1987 [5]. The framework is presented as a matrix of six rows and six columns. Each row of the matrix describes a perspective of the architecture as seen by various roles in the enterprise (*planner, owner, designer, builder, out-of-scope, and the functioning enterprise*). The column headings consist of the six imperative questions (*what, why, when, how, where, and who*). Each cell of the matrix contains a *primitive model* that describes a single aspect of the architecture. The Zachman Framework is a comprehensive, rich and intellectually appealing approach to classifying architectural models but it is not always easy to understand and apply.

Another noteworthy framework is *The Open Group's Architectural Framework (TOGAF)*[6]. As well as describing the framework, TOGAF also includes a detailed methodology that can be applied to the development of an IT Architecture. TOGAF was in fact based on yet another framework, the *Technical Architecture Framework for Information Management (TAFIM).* TAFIM was originally developed by the US Department of Defense.

Following the enactment of the Clinger-Cohen Act, the US Office of Management and Budget (OMB) issued a directive that also included a brief description of an architecture framework. We have chosen this framework as the starting point for our framework because we appreciate its simplicity.

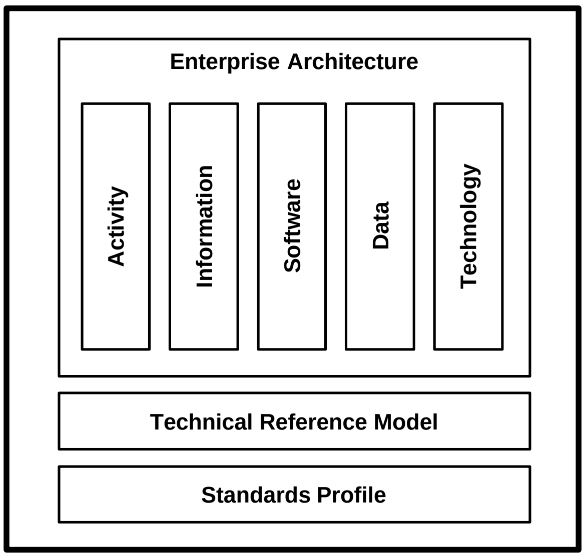

Figure 4: The OBM Architecture Framework

As the diagram above illustrates the OMB architecture framework has three major components:

- an *Enterprise Architecture* that defines the relationships between an enterprise's business activities, information systems and its information technology infrastructure;
- a *Technical Reference Model* that provides a generic description of the services provided by an enterprise's information technology infrastructure; and
- a *Standards Profile* that is a collection of information technology standards that precisely define the services identified in the Technical Reference Model.

The Enterprise Architecture is further decomposed into five *sub-architectures*.

- *Activity Architecture*[1]. This architecture describes an enterprise's high-level business activities and workflows.
- *Information Architecture*[2]. This architecture describes the information required to support the business activities described in the Activity Architecture.
- *Software Architecture*[3]. This architecture describes the software that is required to support the Activity and Information Architectures.
- *Data Architecture*. This architecture describes the logical and physical structure of the enterprise's software-maintained data stores[4].
- *Technology Architecture*. This architecture describes the technical environment in which software executes. It includes components such as hardware platforms, operating systems, database management systems, networking software and the communications infrastructure.

The *Technical Reference Model* and *Standards Profile* influence all aspects of the *Enterprise Architecture*. They provide the enabling forces for a high level of design integrity in the areas of interoperability, extensibility, scalability, re-useability, portability, security, reliability, and performance.

The *Technical Reference Model* comprises of a generic and comprehensive list of all the IT services available to an enterprise. The list includes items such as data interchange, data management, graphics, directory management, and network and operating system services. *The Technical Reference Model* groups the services into logical classifications rather than listing specific products or solutions. Figure 5 provides an example of the TOGAF Technical Reference Model (see TOGAF [6]for a detailed explanation of the logical classifications).

The *Standards Profile* is a collection of standards that precisely define the services identified in the *Technical Reference Model*. Standards are fundamental to the achievement of interoperability between systems. The standards can be grouped into the same categories as the services identified in the *Technical Reference Model*. Internally developed guidelines, de facto standards and formal international standards can all be included in the *Standards Profile*.

[1] In the OMB memorandum, this architecture is called the *Business Process Architecture*. We have changed the name because we have already used the phrase *business process* to describe something else.

[2] It is common practice to either combine the Information Architecture with the Activity Architecture and call it a *Business Architecture* or to include information requirements in the Data Architecture. We do not like the first approach as the phrase *business architecture* sounds rather vague and nebulous. As far as the second approach is concerned, we feel it does not highlight the fact that information is related to business activity while data is more closely related to information technology. In addition, there are issues associated with an Information Architecture such as the management of non-electronic records that are not accommodated well in the Data Architecture.

[3] In the OBM memorandum, this architecture is called an *applications architecture*. We have changed the name because we have already used the word *application* to describe something else.

[4] The Software Architecture and Data Architecture together could be viewed as the definition of an *Information Systems Architecture*. In fact, this composite architecture would appear to be a much better candidate to be named *Applications Architecture*.

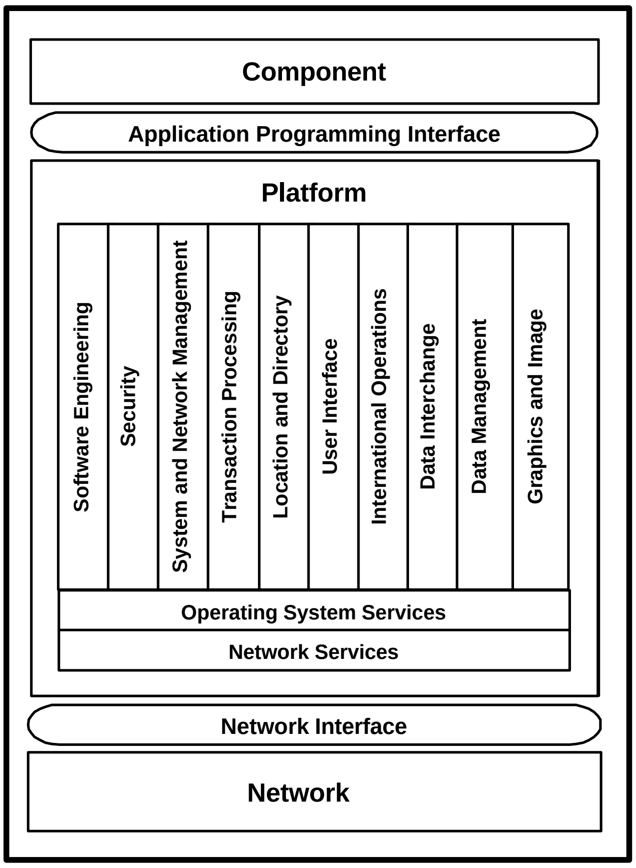

Figure 5: The TOGAF Technical Reference Model

The Extreme Architecture Framework

In summarising our discussion of Reims cathedral and shanty towns, we boldly stated that our inclination was to follow a *middle path*. How then, can this inclination be reconciled with the title of this paper – *extreme architecture*?

Firstly, we believe that the discipline of architecture is in constant danger of being hijacked by fanatics. With the perfectionist cathedral builders and the builders of chaotic shanty towns eyeing each other across an uneasy dividing line, adopting a middle path is of itself an extreme point of view. However, we emphasise that following the middle path is not the same as *sitting on the fence*.

We believe our position is best illustrated by introducing a third building metaphor – the suburban house. As shown in Figure 6 below, when a photograph of a suburban house is placed between those of Reims and a shanty town, it becomes quite obvious that there is in fact, a very attractive alternative to the two extremes.

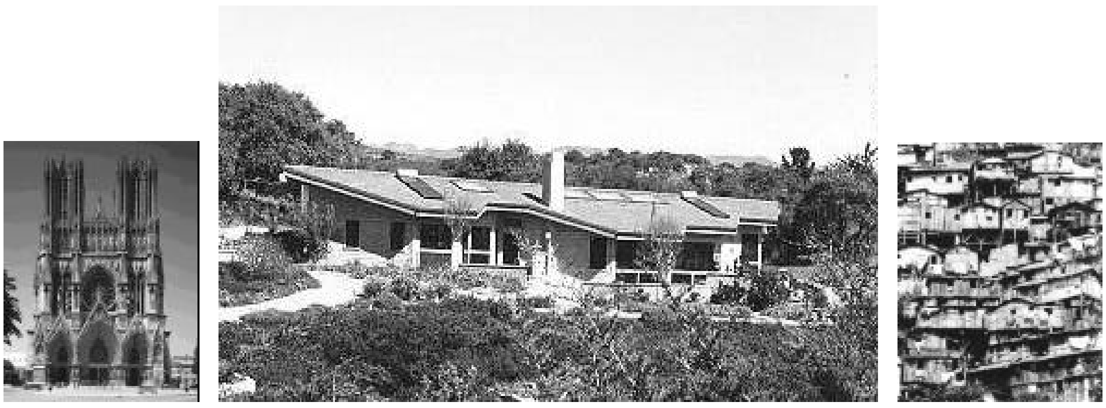

Figure 6: Suburban House and the *Middle Path*

We argue that in most cases, the self-reliance, affordability and pragmatism of the suburban house has a much greater appeal to most when it is compared with the abundance, privilege and order of Reims or the disenfranchisement, poverty and chaos found in a shanty town.

As well as being extreme by avoiding the two extremes, we argue that our framework is extreme because it exaggerates the best aspects of other architecture frameworks. Specifically our framework:

- **Is easy to describe.** The framework is based on a simple five by five matrix. The body of the matrix is populated with just nineteen architectural elements. We like to describe the framework with a single, colourful reference card that we distribute – everyone seems to want one.

  In contrast, many architecture frameworks are complex and difficult to describe. For example, the documentation for TOGAF Version 8 is 313 pages while the TAFIM documentation runs to 8 separate volumes. Try grabbing people's interest with those!

- **Encourages an agile approach to architectural work products.** Each of the nineteen architectural elements can be described using a simple bullet-point list or a detailed UML model as well as many other possibilities in between.

  In contrast, many architecture frameworks advocate the creation of large number of elaborate and detailed models. For example, the Zachman Framework identifies no less that thirty-six, *primitive* models.

- **Unifies a number of disparate disciplines.** We know of architecture groups that work in splendid isolation. Their elaborate models never make one iota of difference to the business managers, business analysts, software developers or IT infrastructure groups. In our framework, the nineteen architectural elements can be grouped into four different areas. Each area is focused on a particular group but retains the context of its relationship to the other elements. The areas are:

  - Business modelling,
  - Requirements definition,
  - Software construction; and
  - IT infrastructure management.

- **Offers a simple, consistent view to the various parties involved in the management of IT resources.** This encourages shared understanding between IT groups and their clients by presenting an area of common ground that can be understood by everyone. This leads to increased participation and ownership by all parties.

  In contrast, some frameworks organise models according to various roles and disciplines. This tends to encourage redundant descriptions of architectural elements at different levels of detail. For example, the Zachman Framework answers the questions what, why, when, how, where, and who from the perspective of five different roles.

  The framework can provide a means to explicitly acknowledge the responsibility that some groups have for certain architectural elements. From the opposite perspective, it also serves as an encouragement for people to acknowledge the responsibility that others have for architectural elements.

It is now time to introduce the framework. As we have already stated, the framework, is based on a matrix as shown in Figure 7 below. We chose this format because the Zachman Framework has demonstrated the simplicity and universal appeal of presenting an architecture framework as a matrix. The rows of the matrix are labelled with the five overlapping, independent types of system described in Figure 3. The columns of the matrix are labelled with the five sub-architectures of the Enterprise Architecture described in Figure 4.

The individual cells of the matrix are used to organise architectural content. For example:

- The cell at the intersection of the *Sector* row and the *Activity* column contains content that describes the activities performed within an industry sector;

- The cell at the intersection of the *Process* row and *Data* column contains content that describes the data associated with a business process;
- The cell at the intersection of the *Application* row and *Software* column contains content that describes the requirements for an individual software application;
- …and so on.

	Activity	Information	Software	Data	Technology
Sector					
Enterprise					
Process					
Application					
Component					

Figure 7: Extreme Architecture Framework

The content of each individual cell is further classified by a number of architectural *Elements*. These *Elements* refine the coarser classification scheme provided by the rows and columns of the matrix. They also offer a comprehensive checklist for architecture content.

Architecture Elements

Figure 8 below shows the architecture framework complete with the nineteen architectural *Elements*. Notice how some of the cells have been grouped together where they share similar content across a number of rows of columns. The most obvious examples are the grouping of *Sector*, *Enterprise* and *Process* into a single row and the grouping of the *Technology* column into a single cell. The completed framework is followed by a brief tour that introduces each of the architectural *Elements*.

- **Activities -** describe the business activities performed within a sector, enterprise or business process.
- **Workflows -** describe the flow of physical objects and information between the business activities.
- **Subject Areas -** classify and group *Information Requirements* having a common theme, *Subject Areas* can also used to group *Businesses Objects* and *Storage Requirements*. A database[5] is a special case of subject area that can actually be implemented and deployed.
- **Information Requirements -** identify and describe the information required in order to successfully perform an activity as well as any information that is generated as a result of performing the activity.
- **Functional Areas -** used to classify and group *Functional Requirements* having a common purpose. *Functional Areas* can also used to group *Non-functional Requirements*, *Interface Requirements*, *Use Cases* and *Test Cases*. An application is a special case of a *Functional Area* that can actually be implemented and deployed.

[5] Terms used in the database world can be mapped to those used in the framework. A logical data model is equivalent to the Business Objects. A physical data model (or database design) is equivalent to Storage Requirements. Data Manipulation language (DML) is Code. Data Definition language (DDL) defines a database Schema. A DBMS such as Oracle is Technology.

	Activity	Information	Software	Data	Technology
Sector					
Enterprise	Activities Workflow	Subject Areas Information Requirements	Functional Areas	Business Objects	
Process					
Application	Use Cases	Interface Requirements	Functional Requirements Non-Functional Requirements	Storage Requirements	Networks Platforms Frameworks
Component	User Interface		Architecture Code Test Cases	Schemas	

Figure 8: Full Framework Contents

- **Business Objects**[6] **-** represent the concepts of interest within the *Sector*, *Enterprise* or *Process*. Concepts of interest include business-related events and time periods; the roles of people, organisations, places and things; the actual people, organisations, places and things; and classifications of any of the above.
- **Use Cases -** describe the usage of a software application by identifying interactions between the user and the software. Each step in the interaction either provides some direct value to the user of the application or indirect value to the application's stakeholders. Use case steps provide value by validating business rules, permanently storing data or providing information to the user.
- **Interface Requirements -** a special type of *Information Requirement* that provides a detailed description of either a user or software interface. *Interface Requirements* should include a full definition for each data element included in the interface.
- **Functional Requirements -** describes the mandatory capabilities, actions and behaviour of a proposed software application.
- **Non-Functional Requirements -** describes the requirements of a proposed software application that are not related to its capabilities, actions or behaviour. These requirements include areas such as the quality of service provided by the application; external constraints associated with the environment in which the application is deployed; requirements associated with the life-cycle of the application; and design guidelines that should be considered during the development of the application.
- **Storage Requirements -** a special type of *Business Object* that describes data that will be permanently stored. *Storage Requirements* should include a full definition for each of the attributes of all the *Business Object*.
- **User Interfaces** – the physical screens reports and web pages that the user interacts with.

[6] Strictly speaking, the *objects* of object-orientation have relevance in three places in the framework; The persistent *Business Objects* described here belong in the *Data* column; business objects that have behaviour (as well as state) belong in the *Activity* column (we actually don't recommend that activities are modelled in this way but some users of the framework may prefer this approach); and software classes and components belong in the *Component* row.

- **Architecture -** various high-level views of a software system that describe its underlying conceptual organisation, the modules from which it is constructed , the organisation of the source code and the run-time deployment of the software.
- **Code -** the human readable source code which defines the software and the binary code which is executed by a computer.
- **Test Cases -** specific ways of executing software that is designed to identify errors and validate requirements.
- **Schemas** – defines an electronic data store in terms of the records (or tables) and the relationships between the records.
- **Networks -** the mechanisms that are used to interconnect hardware and software platforms to permit the transfer of data and invoking of remote of services.
- **Platforms -** the hardware and software required to execute a software application.
- **Frameworks** – a standard component model or reference software architecture such as J2EE or .Net.

Types of Architecture Content

The architecture framework consisting of the matrix and architecture *Elements* provides a classification scheme for architecture content. In an actual architecture based on the framework, the body of the matrix may contain a variety of different types of content. For example the architectural content might consist of (but is not restricted to) any of the following:

- **A model**, list or definition of any the of the actual architectural *Elements.* For example, a list of the core business processes performed by an enterprise; a list of key business objects that are relevant to a sector; or a data model for an software application.
- **An assessment** or SWOT analysis of the current state of an architectural *Element*. For example, an assessment of data quality associated with a database; or a SWOT analysis of the user interface of an application. Assessments might also refer to potential risks and rewards associated with the current state of the element.
- **A vision** of some desirable, future state of the architectural *Element*. The vision should describe architectural elements will contribute to business strategies and goals. For example, a vision that data will be seamlessly transferred between business processes.
- **An underlying principle** associated with an architectural *Element*. A principle is a short statement that guides or constrains some aspect of the architectural element. For example, the principles of minimising data redundancy and duplication or the principle of data entry at the point of data capture.

  Principles tend to define fundamental aspects of an architecture that are infrequently changed or amended. Principles that actually guide the development and implementation of the architecture can also be defined.
- **A potential risk** associated with an architectural *Element*. For example, low customer satisfaction associated with a complex business process.
- **A potential reward** associated with an architectural *Element*. For example, a reduction in procurement costs associated with effective data interchange with suppliers.

Framework Areas

In addition to the initial grouping of cells shown in the framework diagram above (Figure 8) cells of the matrix can be grouped in a number of other ways. Grouping of cells are useful for highlighting focus areas of the framework. A number of obvious groupings are shown in Figure 9 below.

The **Business Modelling** area describes the business as a whole. It includes the activities performed, workflows between activities, information requirements, business objects and a high-level grouping of software functions into a number of functional areas.

The **Requirements Definition** area describes the various requirements and use cases that define a single software application.

The **Software Construction** area describes the physical artefacts that implement a single software application.

The **Infrastructure Management** area describes the hardware and software platforms together with the networks that interconnect them.

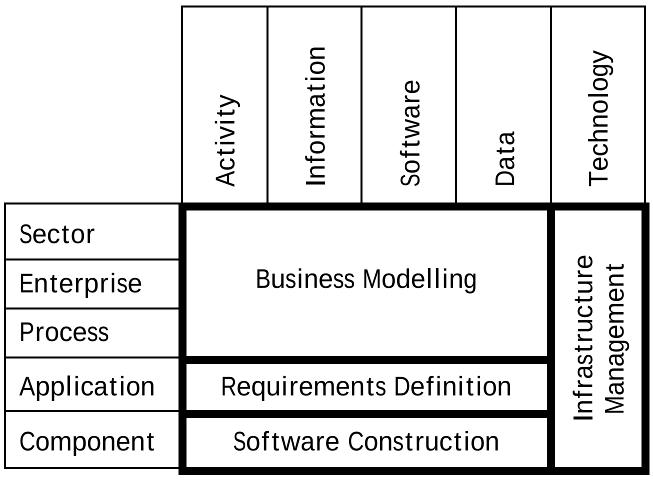

Figure 9: Areas of Architecture Content

As well as these frequently used areas, the cells of the matrix can be grouped into any number of arbitrary areas. This approach can be used to highlight special areas of interest.

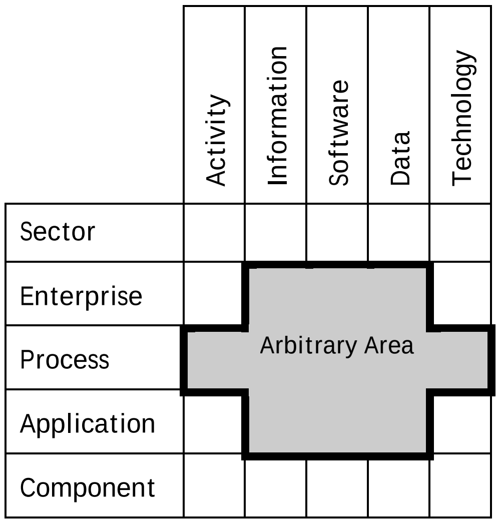

Figure 10: Arbitrary Areas

Using the Framework

We developed this framework over many iterations. At the end of each iteration we reflected on the aspects of the framework that stood out. As the framework developed, we learnt something about:

- The scope of activities performed by the business planning, operations and IT staff; and
- The relationship between an IT provider and its clients.

Our lessons are best described by referring to distinct areas of the framework.

What is produced when?

The *Business Modelling* area of the framework can be regarded as defining the scope of an information planning project. The work products of this exercise become the enterprise and technology context for individual information system development projects.

The *Requirements Definition* area of the framework describes the scope of an information system project. The work products of this project become the specification for an information system.

The *Software Construction* area describes the work products produced during the construction of an information system. That is during its design and implementation.

The *Infrastructure Management* area describes the technology assets of the organisation. Frequently, information system projects do not need to establish a technology infrastructure since the existing infrastructure is used. However, if new infrastructure is required, the development environment is normally established prior to construction while the production environment is established in parallel with construction.

Who is responsible for defining what?

Areas of the framework can also be used to help identify responsibilities. The *Activity* and *Information Architecture* columns identify the responsibilities of the business planning and operations staff. The architectural *Elements* in these columns are defined by the business. Even though the IT provider does not define the things found in these columns, it is important for the IT provider to understand them.

The *Data* and *Technology Architecture* columns identify the responsibilities of the IT provider. The IT provider defines these architectural *Elements* in response to those defined by the business. However, the business planning and operations staff must verify that the data and technology elements are suitable for their needs.

The middle column is interesting. Business planning and operations staff together with the IT provider share the responsibility for the software application portfolio. This is the boundary between the two groups. It is where they collaborate and ideally have joint ownership of the solution. Many projects fail or succeed in this column, due to an *assumed* scope of the functional requirements. For many projects, this column will come to represent either an *axis of joy* or an *axis of sorrow*. The quality of the relationship between the two groups involved (collaborative or antagonistic) will have an enormous influence on the outcome of *joy* or *sorrow*.

These areas of the framework are highlighted in Figure 11 below.

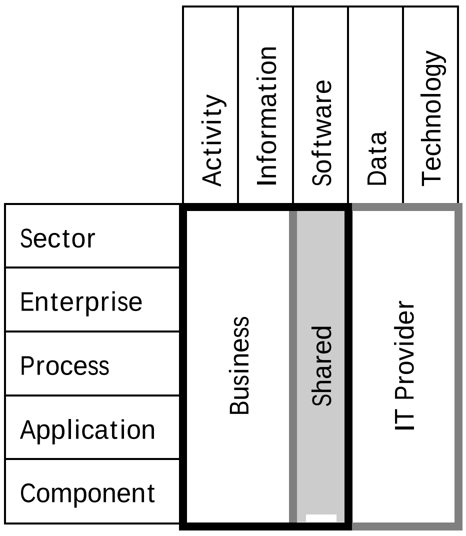

Figure 11: The Overlap of Business and IT Providers

Conclusion

As consultants, we developed this framework in response to a question asked by an IT provider we worked with:

What do I need to know about and manage on behalf of my client?

We think that our response, in the shape of the framework, offers IT providers a practical tool that highlights precisely what they need to manage on behalf of their clients. It also provides an opportunity for the provider's clients to see a clear relationship between their responsibilities and those of the provider. The framework presents a simple and consistent view to client and provider alike.

In elaborating our response, we have presented the elements of a framework, that links human activity systems with software systems. We have shown how this framework can be used to explore and classify an understanding of both types of system. We have also demonstrated how the framework can be used to focus attention on specific issues and to clarify the roles played by various parties.

In conclusion, we would like to remind the reader of the image of the suburban house nestling between Reims cathedral and a shanty town. We would like to think that the framework invites participants to be extreme by avoiding extremes! We hope the result is vision, knowledge, calm realization and enlightenment for all involved.

[1] Brooks Jnr, Frederick P., *The Mythical Man-Month*, 1995, Addison-Wesley.
[2] Foote, B. and Yoder, J., "Big Ball of Mud", *Pattern Languages of Program Design 4*, 1999, Addison-Wesley.
[3] MacDonald, Ean, Notes on a conversation about building architecture. Perth, Western Australia. October 2003.
[4] *National Defense Authorization Act For Fiscal Year 1996*, 1996, US Congress Act
Available http://thomas.loc.gov/cgi-bin/toGPO/http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=104_cong_public_laws&docid=f:publ106.104.pdf [2002, 1 September]
[5]Zachman, John A. "A Framework for Information Systems Architecture.", *IBM Systems Journal*, vol. 26, no. 3, 1987, IBM.
Available http://www.zifa.com/ [2002, 1 June].
[6] *The Open Group Architecture Framework (TOGAF), Version 8, "Enterprise Edition"*, 2002, The Open Group
Available at http://www.opengroup.org/products/publications/catalog/i912.htm