Extreme Architecture - Part 2*, Architecture Components

By Phil Robinson, Lonsdale Systems, and Floris Gout, Independent Consultant

*part 1 was printed in Oracle Scene, Issue 19, Autumn 2004

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

The IT Architecture Papers

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

In the previous article, we offered a comparison between buildings, the traditional roots of architecture, and the more complex world of business and software systems. During the discussion we introduced the basic building blocks of our architecture framework.

In this article, the complete IT Architecture framework is presented as a single, uncluttered diagram. This approach reflects our belief that the framework is simple to describe and easy to recall. However, the diagram is not trivial. lt includes nineteen different elements that, taken together define an IT architecture.

The third article in the series will describe how the framework is actually used. We will also describe how our ideas are related to other ideas, methods and models. Using the framework, an IT Architecture can either be fully defined in single planning project, or gradually fleshed out over a period of time. We hope that the framework will provide a point of reference for business areas supported by IT, IT management, IT project managers, developers and operations staff, frequently charged to do more with less in these lean times. These groups may find that the framework offers them a much-needed lifeline.

Flashback!

In the previous article, we used the metaphors of cathedral and shanty towns to discuss the extremes of perfection and chaos in IT systems. We implied a comparison between building, an activity undertaken by humankind for several millennia, and software systems development, something that has only been performed for the last few decades.

We highlighted the differences between human activity systems and software systems and classified these systems (and their sub-systems) into a hierarchy. We noted that, although the hierarchy is a convenient way of classifying systems, the true nature of business and software systems is to be independent and overlapping.

This led to the notion of interoperability as one of the key architectural issues. Interoperability can be defined as:
'…the ability of a system to successfully interact with other, specified systems.' [1]

Drawing once again on the building metaphor, we attempted to define architecture. In doing so, we referred to a formal definition of IT architecture found in legislation passed by the US Congress; the Information Technology Management Reform Act of 1996 also known as the Clinger-Cohen Act,
'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 Extreme Architecture Framework

It is now time to introduce the framework. As we have already stated, the framework, is based on a matrix as shown in Figure 1. 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. The columns of the matrix are labelled with the five sub-architectures of the Enterprise Architecture. These were described in Part 1.

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					

Fig. 1: 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 2 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 be used to group Businesses Objects and Storage Requirements. A database[2] 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 be 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.
- **Business Objects**[3] - 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.
- **Architecture** - various high-level views of a software system that describe its underlying conceptual organisation,

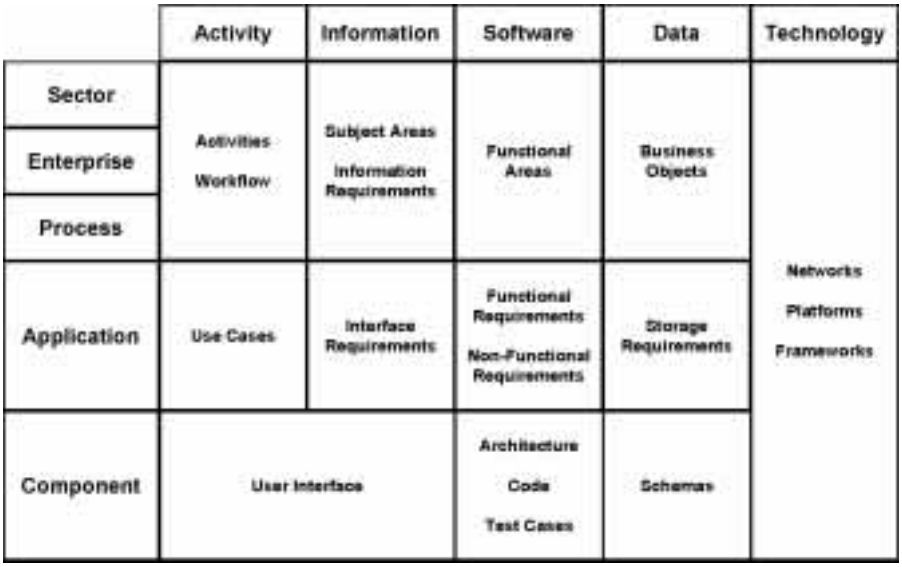

Fig. 2: Full Framework Contents

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 that 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 services.
- **Platforms** - the hardware and software required to execute a software application.
- **Frameworks** – standard component models or reference software architectures 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 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 a 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 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.
- **A vision** of some desirable, future state of the architectural Element. The vision should describe architectural elements that will contribute to business strategies and goals. For example, a vision that data will be seamlessly transferred between business processes.
- **A strategy** or course of action to achieve the future state of the architectural Element. For example, a strategy for Functional Areas maybe to create a 'spoke and hub' application architecture in order to transfer data between applications, to support the seamless transfer of data 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.

So what! Who cares?

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 up 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. When the image of a typical suburban house is placed between the extremes of Reims and a shanty town, it becomes quite obvious that there is in fact, a very attractive alternative to the two extremes.

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 eight separate volumes. Try grabbing people's interest with those!

- **Encourages an agile approach to architectural work products.** Each of the 19 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 a large number of elaborate and detailed models. For example, the Zachman Framework identifies no less than 36 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 19 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.
  - 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.

So in this article we have presented our framework, which we use to undertake our consulting assignments. The benefits have been stated; simple and easy to describe, as well as unifying disciplines and easing communications between groups. The framework is used as the basis for creating work products at each stage of the development lifecycle. In our next article we will draw together other ideas and show how this framework can work in harmony with concepts of business planning and project management. This will be featured in a future issue of Oracle Scene, and/or in the library on the UKOUG web site at `http://www.ukoug.org/lib/`

1 *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*

2 *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.*

3 *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.*

About the Authors

Floris Gout gained his Bachelor of Applied Science (Information Science) at Edith Cowan University in Perth, Australia. Whilst studying at ECU he worked at the University of Western Australia, building research databases for epidemiological studies. Floris was then employed at the Department of Justice from 1991 till 1999. He became Data Administrator and was Project Manager for its first data warehouse. Floris has been an independent contractor since 1999 and he is still enjoying new and creative challenges. He can be contacted at floris@floris.com.au

Phil Robinson has been involved in the planning, analysis and implementation of a diverse range of business, scientific and technical information systems. Phil is an experienced workshop facilitator and has led numerous workshops in the course of his consulting assignments. Phil has presented training courses for organisations in Australia, Thailand, Hong Kong, Singapore and Indonesia. As well as presenting courses, Phil has authored numerous courses for industry and three University units. He has also had two books published on programming Apple computers. The books were published in a number of countries including the USA, UK and as translations in Germany and France. More recently, he co-authored two award-winning articles describing an original organisational theory. He can be contact at Lonsdale@iinet.net.au

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.

This article was originally printed in the AUSOUG's Foresight magazine.