Improving Architectural Design Decisions

A. B. Boake
ABSA Group IT
268 Republic Rd, Randburg
andrew.boake@absa.co.za

Abstract. The value of architecture in a corporate information technology environment lies in guidance on technology choice and system design. Underlying this is the ability of the architecture team to research the relevant architectural domains well, to formulate informed strategies, to document these explicitly, and to guide projects in their application. To do this, architects must make decisions between competing directions, and on difficult trade-offs in their application. These decisions are often based solely on experience, gut-feel, even bias. They are typically arrived at using implicit reasoning such as rules of thumb, and are often poorly articulated. This results in poor corporate technology decision-making, and unclearly documented architectural direction. This paper investigates what can be done to improve this decision-making. It positions architectural decision-making as an exercise in balancing design forces, and the role of solution architect as facilitator between the representative stakeholders. We describe an experience in building a team which has followed this approach in a particular corporate environment, and document the lessons learnt.

I. INTRODUCTION

Architecture is a loaded word in corporate IT. For some it brings to mind a position of power and insight: architects as a group of strategic thinkers, visualizing the future of business and its supporting technology, laying out the desired landscape, and guiding the pieces of that landscape into place. For others, "architecture" conjures images of a tall white tower, with sages looking out high above the plains of ordinary experience and pronouncing what things ought to look like without giving any indication of how to get there. The difference lies in the level of involvement of architecture in the trenches – not just prescribing strategic direction, but being involved in the painful details of how to actually make things work, design decision by design decision.

In a corporate IT environment, architecture must play a vital dual role. Under conditions of increasing complexity, scale and urgency, there is a need for a group of technology people with a strategic mandate, to look ahead and steer the ship around the dangers, towards the next port. But, equally important, is the need for people who understand how to guide the design decisions made in projects so that they align with strategic architectural direction. These are the two faces of architecture – strategy and application.

Some of the problems we face are due to the expectations we place on architects. As architects we generally make decisions using experience or rules-of-thumb. Apart from a computer science education which gives us the basics of our field, we do not often have the benefit of formal training in architecture. A current lack of technically skilled resources propels us into positions of power before our appropriate time, and we are faced with making important strategic decisions, ill-equipped to do so. Our experience is often limited to a particular technology, or to projects in a particular area. The rules of thumb we use are consequently a collection of commonly held beliefs from those areas: you must layer your application; you shouldn't use object-oriented databases because they are complex and proprietary and you can't get the required business reports out of them; Microsoft technologies are not scalable; you shouldn't use Java for financial transaction processing. These beliefs form the dubious basis of many an architecture, and substantiate many project design decisions.

It is not to say that these beliefs do not have a sound basis – it is just that we don't test them well enough before using them as a basis of design decisions in particular situations.

The upshot is implicitly reasoned and poorly articulated design decisions, leading to sub-optimal, or in some cases, incorrect system designs.

The hypothesis of this paper is that, using the concept of design forces, we can pose architectural design problems as an explicit trade-off between competing forces, and more specifically as a conversation between representatives of these forces. Further, we can embed this notion into architectural design artifacts and governance, thereby improving architectural design decision-making.

The way in which we will approach this is by case study. Over the past three years, ABSA has grown an IT Solution Design team, and positioned it as the solution facilitation arm of IT Architecture. A system of design governance has also been put into place, which increases the visibility of design decisions, and puts the different representative stakeholders in a position to influence those decisions. These measures have experienced a certain degree of success in improving the quality of architectural design decisions.

In this paper, we describe the positioning of that team, examine the lessons learnt through that particular experience, and attempt to extract general principles that might be helpful to others in a similar situation.

The paper is structured as follows: Firstly, we position how different representative software development methods see architecture. This is in order to position architecture with respect to software development projects, so that we may

understand how architecture can influence project decision-making.

We then look at the process of designing software solutions, and in particular how designing these solutions inevitably equates to trading off various forces. We also look at how modeling helps us to visualize different aspects of a software design.

We then go further than particular projects and position architecture in the enterprise, showing its different constituent parts. We build up a model of the enterprise, showing the different components of an Information Technology division, and how these provide the necessary aspects of business system development and support. Within this context, we examine the role of the IT Architecture department, and show how it can be enhanced by the addition of an IT Solution Design function, with design governance in the form of an Enterprise Design Authority.

To illustrate the value added by IT Solution Design, we use the concept of design forces discussed earlier, and walk through an example of a conceptual design, showing how the different forces can be balanced and documented explicitly, so that the best overall solution is selected.

During this process, various tools are necessary to allow solution designers to visualize and control aspects of the design – we discuss a selection of tools that we have found useful.

Finally, to conclude, we discuss some challenges faced by an IT solution design team, and suggest further work in this space.

II. Positioning Architecture – In Software Development Methods

Software development methods are divided on the usefulness of architecture. The Rational Unified Process (RUP), as an example of an iterative-incremental method which emphasizes the importance of documentation, sees the architecture of a system as a group of decisions that should be made early, and proven early, in the process of developing that system. These are typically decisions that affect the qualities of the system, such as security, performance and robustness. These decisions may include, for example, dividing the system into independent layers, and replicating the components in a layer for the purposes of scaling and fail-over.

RUP believes that such decisions would be difficult to introduce into the system at a later stage, necessitating much disruptive change, so it is better to make these decisions early. The wisdom of such decisions should also be proven early - in the Elaboration phase of a RUP project, it is recommended that an "executable architectural skeleton" be built that demonstrates the viability of the architecture, and which is used as a framework around which to build the rest of the system. We share this sentiment of proving the architecture "where the rubber hits the road" before basing large-scale development on it.

Extreme Programming (XP), one of a group of agile methods that emphasizes communication above documentation, is rather disdainful of the concept of a group of clever people who prescribe overriding structure to the rest of the project. XP'ers rather see the architecture of a system as the common understanding of system structure that the team shares. They caution against complex up-front designs, supposedly put in place to enable the system to absorb changes. The danger, which is all too often experienced, is that the complexity of the framework complicates the initial system development, and when change does come, it is of an unexpected kind, is not handled by the framework, and much change to the system needs to happen anyway [Fowler]. In a similar vein, Fred Brooks, in a chapter of his famous "Mythical Man-Month" cautions against an architect's second system, in which he is now rearing to add all the bells and whistles, with all of the accompanying complexity. XP, therefore, is wary of architectural complexity without due value.

XP'ers also maintain that it is not necessary to make all important decisions up front. A combination of test-first programming (rigorously defining and testing the expected behavior of system components), continuous integration (ensuring that the parts of the system are continually compatible) and refactoring (being able to change the design of the system without changing its behavior) means that it is possible to start with only as much of a framework as is absolutely necessary to deliver the initial parts of the system, and to add complexity as it is needed.

Here we believe a word of caution is necessary. As shown in Figure 1 below, in system design, there are two kinds of simplicity: naïve and informed. Naïve system design is how one starts out. As you add functionality, the design becomes more and more complex in order to cater for that functionality. It takes real skill to take a step back and discern a simple design that handles all of the current functionality, simply. Here is where experience such as design patterns and architectural styles give the system a coherent structure. It is also debatable whether one can introduce the design disciplines necessary for such an informed simplicity design only later on in the project.

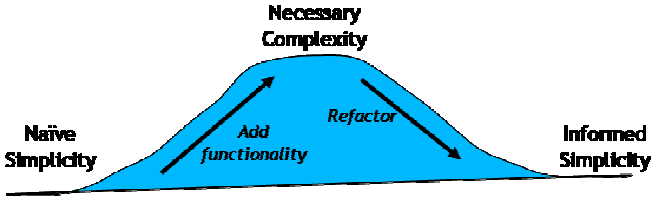

Figure 1: *Moving from Naive to Informed Simplicity*

III. Design Forces

We are told that the essential difference between science and engineering is that science observes the world and attempts to understand and explain it, while engineering wants to change

the world, to create something that was not there before, for the benefit of people. Science is therefore chiefly concerned with analysis, while engineering is concerned with synthesis.

Synthesis involves design. Synthesis involves putting together different components, creating something in an existing environment that will achieve an end. Christopher Alexander, in his book "Notes on the Synthesis of Form" describes this process using an example of a kettle. The intended outcome is something that makes hot water, but designing the kettle involves balancing many different forces. The size of the kettle is a balance between the amount of water to be boiled, the speed of boiling it, and the ability to carry it safely. The materials used are a balance between taking advantage of their properties (the handle must insulate the user from the heat, the body must contain heat and be easy to clean), ease and cost of manufacture (the more materials there are, the more complex and therefore the more expensive the manufacturing process), aesthetics (depending on the target market, the kettle must appeal to perceptions of style), and safety (the design must take into account the dangers of combining electricity and water, avoiding burns and spillage). Alexander says that this typifies design problems: there are requirements to be met, and there are interactions between the requirements that make the requirements hard to meet.

In the same way, design of software solutions involves the balancing of many forces. Users want features which are easy to learn and use. Project managers want these developed using minimal time and resources. Architects want system quality and compliance to standards. Unfortunately, most of these are in opposition, and need to be traded off in the design.

These trade-offs bring to mind physical forces that work in opposition to one another. As seen in Figure 2, for an aircraft to fly, several forces need to be in balance: the thrust of the engines overcoming the drag of friction, and the lift from the wings overcoming the aircraft's weight.

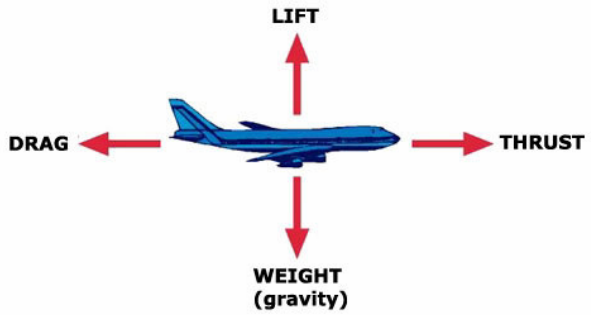

Figure 2: *Physical Forces*

In a similar way, we may envisage design forces on a software system as weights acting on a ball through a system of pulleys (see Figure 3).

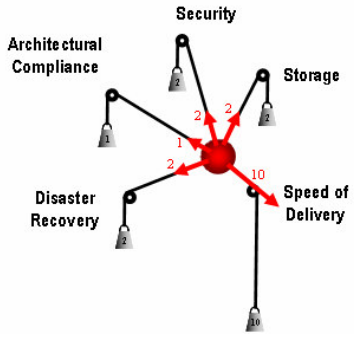

Figure 3: *System Design Forces*

Each design force seeks to pull the final solution in a particular direction, and it is the balance of these forces that positions where the ball (the final system design) ends up.

An example might clarify these forces and different ways they may be balanced. Imagine a situation where some software is required for our business. There are two options: we can buy an existing package and install it, or we can build a new piece of software. The two options are depicted in Figure 4 below.

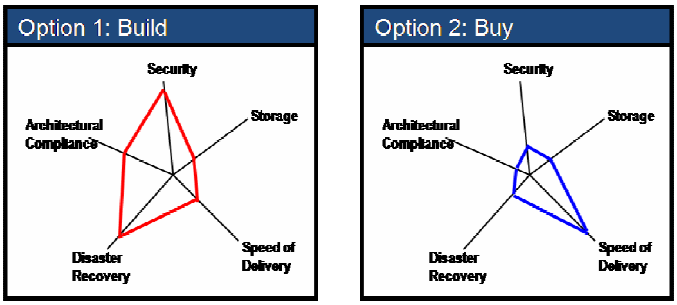

Figure 4: *Two Options*

Firstly, a word on the diagram type being used. This is known as a 'radar' diagram – several dimensions are shown as arms radiating out from the centre. The centre is 'bad' and the outer ends are 'good' for each dimension. The overall 'shape' of the solution can be seen in how much area is covered.

In these two options, we can see how different 'design forces' are balanced in different solutions to the same problem. In the right hand option, the package vendors will have their own architecture, chosen for a packaged solution, so compliance to our architectural standards will not be good. Also, their choice of security and disaster recovery will not be to our required levels. However, seeing as the functionality is already written, the speed of delivery will be excellent.

In the left hand option, we can build the system according to our architectural standards, providing our desired levels of security and disaster recovery, and optimize storage. But, we will take longer to deliver.

Presenting these two 'shapes' of the solutions on offer, we can more easily reason through their advantages and disadvantages.

But, how do we come up with solutions that balance the forces experienced? It is here that design patterns prove useful. A design pattern can be seen as an explanation of how, in a particular situation, the acting forces can be balanced in a way that provides a good design, or a desired outcome. As an example, Christopher Alexander, in his book "A Pattern Language" sets out a system of design patterns for the built environment: rooms, homes and cities. In that system, he describes typical situations where design forces oppose one another, and suggests ways in which they may be balanced.

A particular example might illustrate this principle. Alexander describes someone coming into a room with a window. He stands at the window looking out, until he gets tired. He then finds somewhere to sit. Rested, he feels the need to look out again, until he tires. The forces of looking out and sitting are not in balance. The solution is to introduce a place where he might sit and look out, called a Bay Window.

Figure 5: *Bay Window*

Originally inspired by Christopher Alexander and his building patterns, there is extensive software design patterns literature suggesting ways of achieving decoupling, guaranteed message delivery, portability, scalability etc. These patterns range from code-level principles right the way to architectural styles.

IV. MODELING

Models are important to architecture because they allow us to visualize aspects of different solutions. Modeling is of course a mechanism of abstraction – we use modeling elements to depict important aspects of a system to particular stakeholders. For instance, those concerned with the functionality of a system might want to see a model of the system's business domain (a class diagram), those concerned with performance might want to see a model of the executing processes and the flow of data between them, and those concerned with the hardware requirements and configuration might want to see a physical view of the machines and networks.

This is the crux of the concept of architectural views, for example as described by Kruchten [1]. Architectural views are depicted using models, at a certain level of abstraction, to answer particular questions (eg what, where, who, how, when, why) (see Zachman [6]). A model we have found useful to understand the different dimensions of architectural views is depicted in Figure 6.

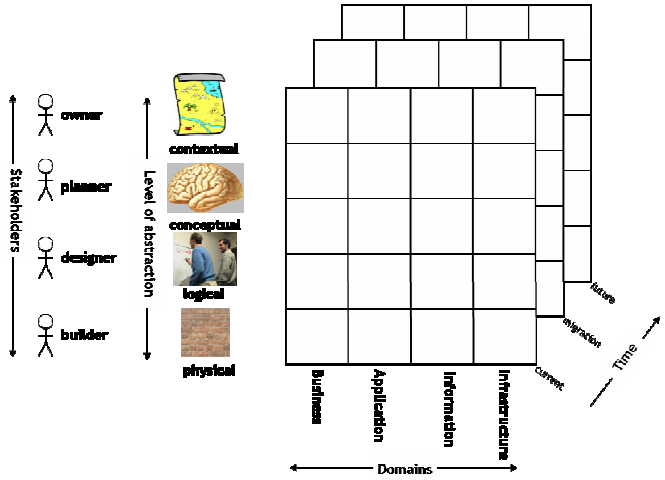

Figure 6: *Architectural Views*

Firstly, let us define an architecture describing a domain as broadly consisting of the main components in that domain, their responsibilities, their relationships, and constraints on these. To describe a typical business-supporting IT, we can then define four main kinds of architecture: how the business functions, the application components that support the business, the information being manipulated, and the IT infrastructure that the applications and information are deployed on. This forms the 'domains' dimension of our model.

The stakeholders can be arranged, depending on the level of detail required by each. For example, the system owner or sponsor is really only interested in overviews, to place the value of the system in context. The planner is interested in more detail, but still only at a conceptual level. The designer would be interested in a logical level of detail, whereas the builder would be interested in detailed information. This forms the 'levels of abstraction' dimension of our model.

Finally, we might be interested in what the picture looks like right now, before we start changing anything. We might then also be interested in what it should look like in the future, and in a sequence of steps that will allow us to get there. This is the 'time' dimension.

We will use this model later in our walk-through of a conceptual design to position the different architectural views being used.

It is interesting to note that XP and RUP also differ in their approaches to modeling. While RUP emphasizes the importance of models in facilitating the design thought processes and documenting these explicitly, XP sees models more as transient diagrams used on a white board to describe aspects of the system, as a medium to facilitate tacit system

knowledge exchange between the team members, with the code itself being the source of ground truth.

As the materials of architecture consist of documents and models, we are more aligned with the RUP thinking. Models allow us to visualize aspects of the system, reason through design rationale, and document it.

V. Positioning Architecture – In the Enterprise

Up to this stage, we have dealt mainly with aspects of architecture within a project. In smaller software development efforts, where the project is the whole, and in software development project houses, where each project provides an architecture to the client, it is easy to confuse architecture with the project's system design.

In larger enterprises, where the architecture is the state of the supported systems, and it is worked on by a sequence of projects, it is necessary to position architecture beyond projects, in the enterprise as a whole.

In this section, we will build up a model of a typical business enterprise. We start with a simple situation (Figure 7) where business units with supporting systems give through requirements to projects, which deliver systems to them.

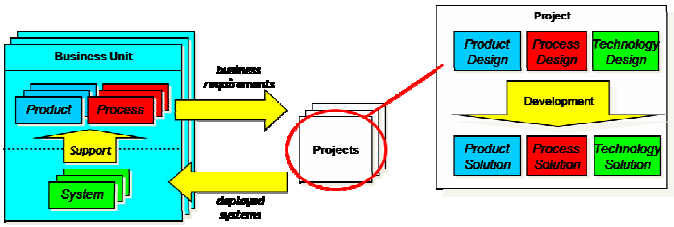

Figure 7: *Business Units' Projects*

To prevent current project staff from having to spend more and more of their time running the systems they have developed before, these systems are handed over to people dedicated to IT operations, who become responsible for their operation.

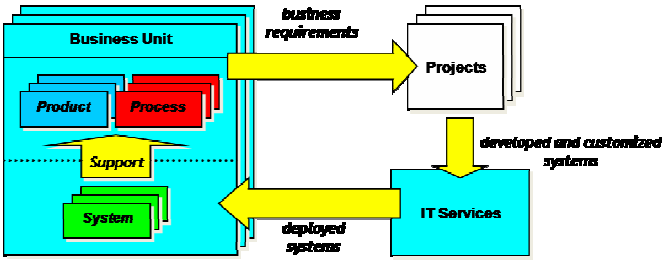

Figure 8: *IT Operations*

As projects come and go, people are assigned to these projects from resource pools (Figure 9).

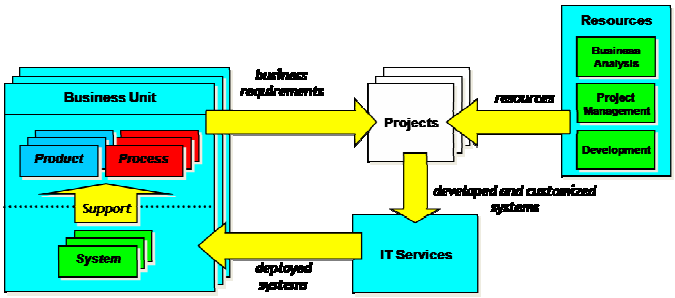

Figure 9: *Project Resources*

A Project Office is established to coordinate the efforts of many projects. Also, a function called Business Change becomes responsible for managing and controlling the changes to the business, and becomes the conduit through which IT projects are executed (Figure 10).

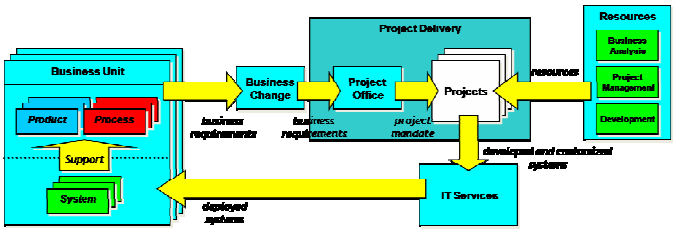

Figure 10: *Business Change & Project Delivery*

As the number of projects increases, it becomes necessary to guide the total effort strategically. IT Architecture sets standards and guides the projects and IT infrastructure support (Figure 11).

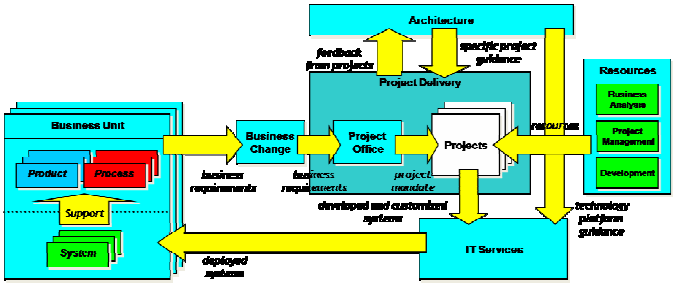

Figure 11: *IT Architecture*

In a small company, it is possible for an architect to be responsible for the architecture as a whole. However, it can quickly become a large responsibility. Quite often, at this stage, architecture gets split into three: *business architecture* (responsible for strategic business direction, including the development of new products and processes), *systems architecture* (responsible for strategic application component direction, including build or buy decisions) and *infrastructure architecture* (technology infrastructure support, including platforms and networks).

For larger enterprises, these functions are looked after by teams of people, and the divisions are more granular. An example breakdown is shown in Figure 12.

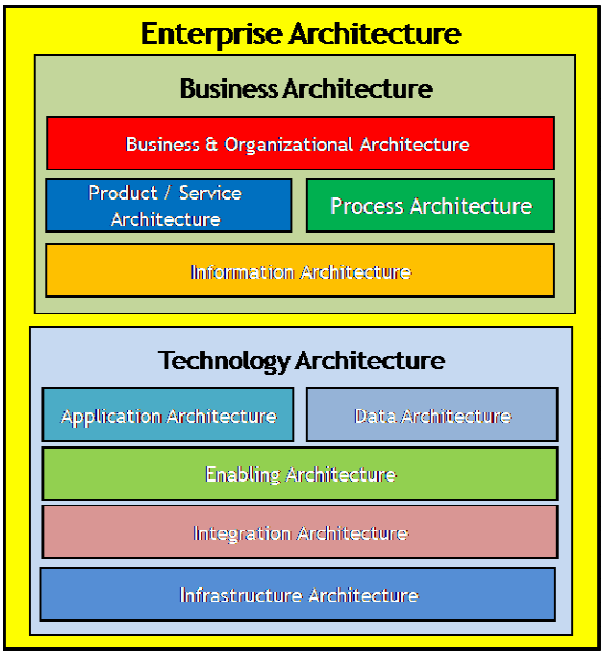

Figure 12: *Enterprise Architecture*

The whole is now known as Enterprise Architecture [7]. This consists of business architecture, which looks after organizational structure, the products and processes, and the information (eg the data warehouses), and technology architecture, which is split into those that look after the application components, the data stores, the deployed packages (eg SAP), the integration middleware and the platforms and networks.

At this stage, Architecture becomes an organization in its own right, with much coordination required for it to deliver a coherent service.

It is interesting to note that, at this granularity, the different forces in the software development lifecycle are represented by different organizational units. A newly formed project, trying to deliver a new piece of functionality, means the project manager and his development resources are faced with trying to satisfy an intimidating array of stakeholders.

VI. IT Solution Design

Each of these stakeholders is charged with looking after the strategic direction of a particular part of the solution, but there is no one trying to bring them together tactically, in the context of a particular project. Projects try to satisfy the most obvious stakeholders, but mostly follow tactical solutions in the interests of delivery. This results in the following problems in solution delivery:

- Too little coordination of technology direction in projects: projects make individual tactical decisions based on development expediency.
- Too little understanding of inter-relationships between individual projects. Projects often work against one another rather than cooperating, and often get held up by unexpected dependencies on each other.
- Duplication of work across projects.
- Too little guidance and governance of standards and compliance, because architecture is not explicitly involved in project work.
- Too little operational and performance consideration in project designs. A focus on development expediency has the result of forgetting important non-functional considerations in system design. Consequently, new systems begin to suffer from, for example, performance and operational management problems.
- Difficulties in reliable deployment and management of systems.

It is at this point that a new role is often introduced to architecture – that of Solution Architect. The solution architect is charged with walking along with a set of projects, and guiding them towards a comprehensive solution. In other words, while the project manager has as his mandate the coordination of time and resources, the solution architect's mandate is solution quality, making sure that all of the aspects necessary for a well-designed solution have been addressed.

The solution architecture team (called IT Solution Design in ABSA) provides design direction and review during the entire development lifecycle (see Figure 13).

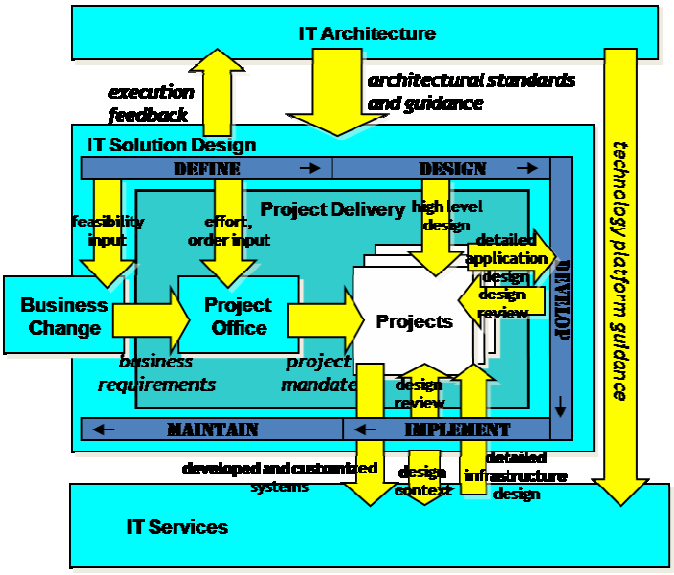

Figure 13: *IT Solution Design*

Early in the business change lifecycle, the team is expected to provide input into the technical feasibility of proposed projects. When the projects are constituted in the project office, the team again provides assistance in the estimation of the effort required to complete the project, the impact the project will

have on the current application landscape, and the options available to pursue. In the Design phase, a solution designer facilitates and documents the high level design decisions, and in subsequent phases, acts as consultant, reviewer and mediator to ensure that these decisions are followed.

Another view of this process is provided in Figure 14. The outer ring of process steps documents the typical activities undertaken in a system development lifecycle, starting from normal business operations at the left. Analysis of the business problem and its proposed solution is followed by architectural direction, analysis of systems enhancements to support the new solution, development, testing, and implementation of the solution. Change management in the business operationalizes the solution, and it is incorporated into normal business operations.

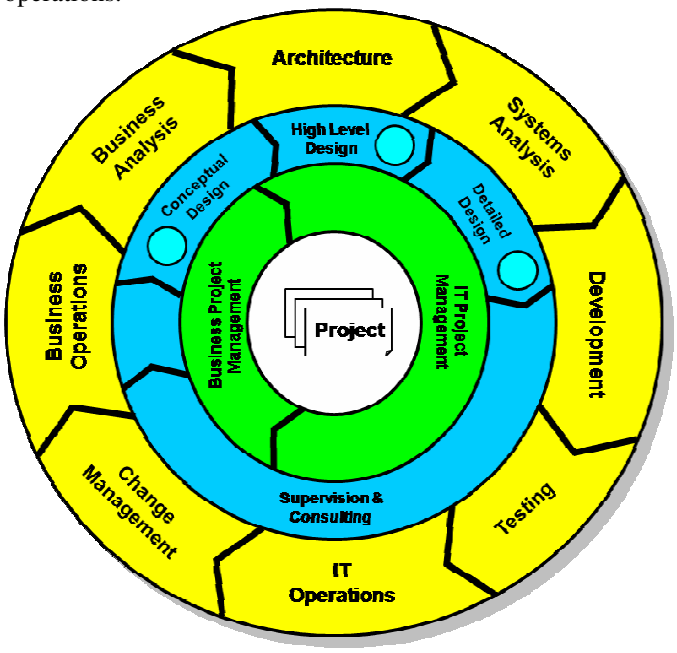

Figure 14: *Solution Design Process*

The inner ring of project management activities shows when business and IT project management takes the lead in coordinating these activities.

Between them is a ring of activities of the solution design team. During the business analysis stage, a solution designer prepares a conceptual design, which enumerates the different options available, and their trade-offs. Having chosen one of the options, in consultation with the different architecture stakeholders, the solution designer prepares a high level design, which in essence is the blueprint for subsequent development and infrastructure, each of which use this blueprint as a basis for their own detailed designs. During the rest of the lifecycle, the solution designers act in a supervisory and consulting capacity.

There are two primary aspects to solution design – facilitation and governance. The facilitation aspect has to do with how solution designers approach their task. A solution designer need not be the expert in any particular field, but needs to know enough of all of the fields to be able to facilitate the necessary trade-offs. In the preparation of a design (we shall see an example in the next section), this facilitation is a difficult task – each of the experts needs to be consulted, and their input consolidated into an overall design. For this reason, the people hired for this role needed to be experienced architects in their own right. One of the major factors of success in a designer, though, is being able to work with people and convince them to make concessions in what they might consider imperative, to cater for the concerns of others.

The second primary aspect is governance. For the overall success of the solution design initiative, it was vitally important to set up a senior decision-making body that would oversee its governance. This body (called the Enterprise Design Authority, or EDA) would consist of senior representatives of the stakeholder community: programme and project managers, resource managers, IT services managers, information management, and senior architects from the different areas (networks, platforms, security, disaster recovery and applications to name a few). Meeting every week, a working committee would sit to hear presentations of designs to be approved. This would give the senior IT decision-makers a view into the project pipeline, and real influence into what would be allowed to continue beyond conceptual or high level design stage. Each stakeholder would have the right to veto if desired, but would rather be encouraged to state the case for a higher consideration of the particular design force that they represent.

What this achieved, more than anything else, was an explicit documentation and ratification of design rationale and decisions. Design decisions were no more implicit, and on the shoulders of an individual – the EDA would officially give a stamp of approval to the design direction, or would guide the direction until it could give approval. The designer would prepare the ground for difficult conversations by facilitating a preliminary solution to be brought before the EDA, and the members representing the different design forces would participate in conversations at the EDA that negotiated the necessary trade-offs.

VII. An Example

Let us walk through an illustrative example to see how this would work in practice. For our chosen example, we are going to add workflow to a particular banking product system to allow the business to better measure the effectiveness of their processes.

The term “workflow” refers to a kind of system that allows one to represent, execute and measure the performance of a business process, such as claiming on short term insurance. The different steps are modeled, and the participants, either human or system, are identified. When these processes are stepped through, the system measures the time taken for each step, producing reports that can be examined for process improvement. Taken a step further, the system can initiate

corrective action, for example escalating notifications to managers if service levels are about to be exceeded.

A document management system is often used in concert with workflow. This kind of system allows users to capture digitally all documentation associated with a process (such as forms, certificates and correspondence), and sends these along with each step in the process, so that participants in the process steps are able to view all of the relevant documentation in order to make decisions.

In preparation for this conceptual design, the reader is reminded of our modeling framework (seen here in Figure 15).

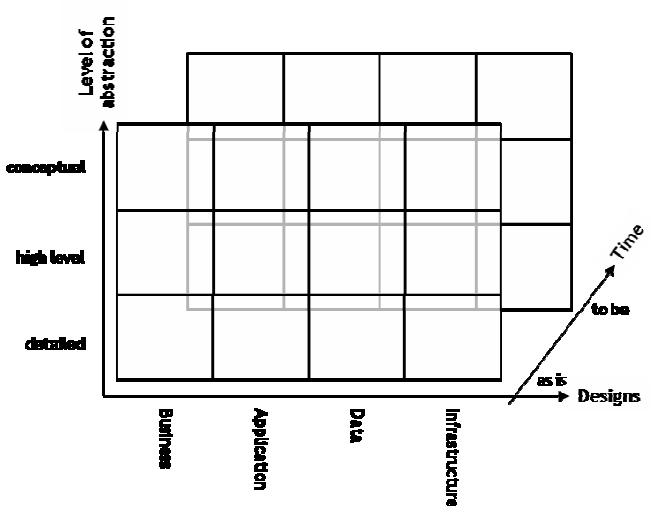

Figure 15: *Model Framework*

We will use this to position each of the models in our conceptual design.

Firstly, we need to document which steps in the business process we will need to change to add workflow and document management. Figure 16 highlights the additional steps where documents are captured digitally.

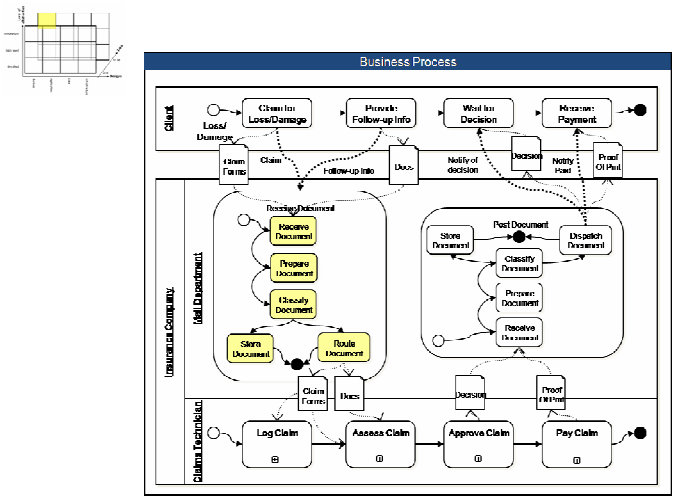

Figure 16: *To Be, Conceptual Business View*

Next, we need to model the information that we will need to store. This is shown in Figure 17.

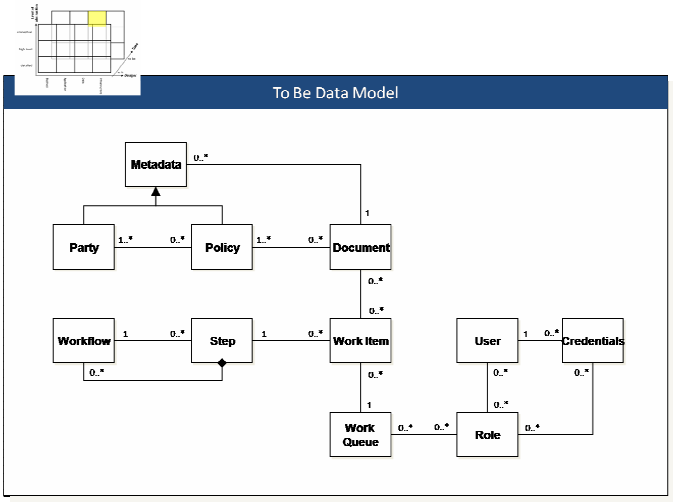

Figure 17: *To Be, Conceptual Data Model*

Figure 18 shows what application components will be needed in the new application landscape.

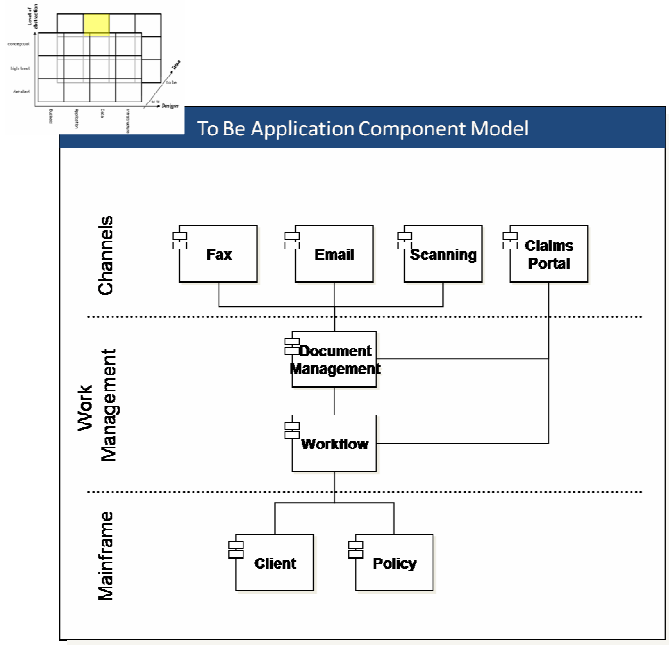

Figure 18: *To Be, Conceptual Application Component Model*

Finally, Figure 19 shows one possible infrastructure configuration to support these applications. This particular configuration places the capture and storage of digital documents in the branches, a decentralized model. Another option we could consider is a centralized solution, where all storage and processing of data happens in the central data centre.

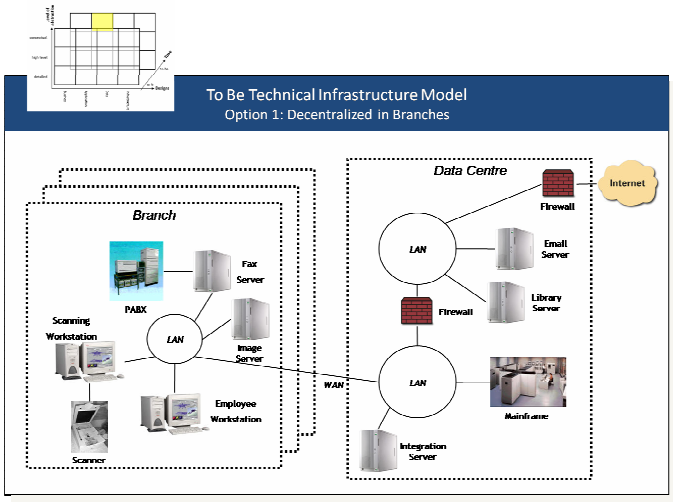

Figure 19: *To Be, Conceptual Infrastructure Model*

The first three diagrams give context to the EDA about the business problem, and the application landscape involved. However, the real options in this conceptual design are the trade-offs involved in different infrastructure deployments.

The radar diagrams shown in Figure 20 show summaries of the different design forces inherent in these options, after consultation with the experts in the different fields.

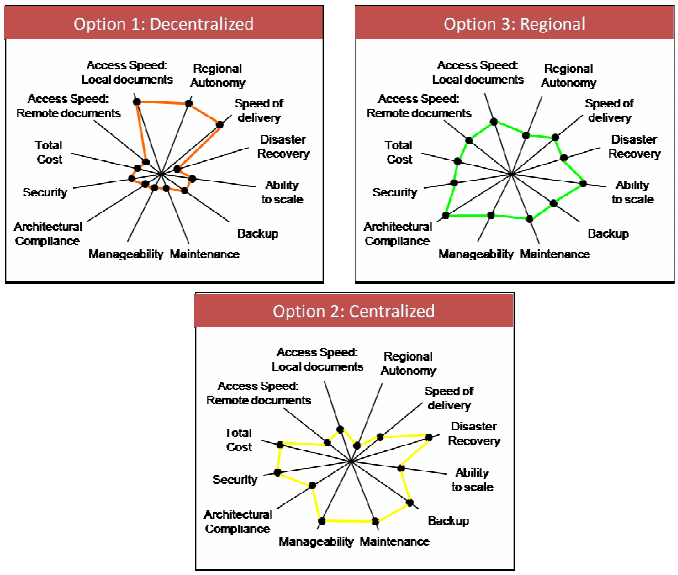

Figure 20: *Radar Diagrams*

Briefly, the decision rationale is as follows: The decentralized option, although it allows users to very quickly access their local documents (they are stored locally), access to remote documents would be slow. This option gives them regional autonomy (because they are not dependent on centralized deployment of hardware). However, the total cost of the solution is prohibitive because of the duplicated distributed hardware, and the manageability of the solution leaves much to be desired.

On the other hand, a centralized solution makes local image access unacceptably slow. It also slows down delivery because of a dependency on centralized deployment. However, the management of the solution is enhanced, as is architectural acceptance of the solution, and the total cost is considerably lower.

After deliberation at the EDA, it was decided that a hybrid solution, deploying the hardware at regional hubs, would give a measure of the advantages of both solutions, without the severe disadvantages of each.

This simplified example is illustrative of the conversation that regularly occurs during conceptual design presentations at the EDA, and illustrates the kind of reasoned design decisions that can be reached.

While the EDA is also involved at the next level of detail, high level design, we will not cover that in this paper, save to mention that a high level design involves the more detailed design decisions for each of the design forces (including security, disaster recovery, storage capacity, network capacity and platforms), signed off by the representative parties, so that a unified blueprint guides the further development, infrastructure procurement and installation.

VIII. TOOLS

ABSA's solution design team grew from around 6 people originally, to around 25 at its largest. As is normal, growth implies the need for increased coordination and automation. This section describes some of the tools we introduced to this end.

Project Tracking

Projects that engaged the services of a solution designer needed to go through a project tracking tool. This was primarily to keep track of the time spent by team members, as it was expected that our time would be recovered from those projects.

Design Effort Estimation

As we went through different projects, we started to understand the effort it would require to do the designs for different kinds of projects, and we became able to estimate more accurately as we went along. We devised a spreadsheet that would ask for certain key properties of the project in question (such as how many business units were involved, how much integration, security issues such as external access, and reliability expectations), and from that information would estimate the design effort required.

Quality Attribute Gathering

We realized that much of the information we needed to gather had to do with the non-functional requirements. It was difficult to remember quite an extensive list, so we developed a checklist with descriptions of the requirement types and the

implications, both in terms of design considerations, and who we would have to talk to.

Peer Reviews
As more team members came aboard, we needed to give guidance to the less experienced designers, as the kinds of decisions we were facilitating were significant. So, we set up a peer review panel which met twice a week, to be of assistance at various stages of the design process. Solution designs would be brought to the panel at the 20% stage, which really just amounted to a conversation about direction and options, and also at the 80% stage, which was a dress rehearsal for the presentation to the EDA. The more experienced designers knew the kinds of questions the EDA was likely to ask, and were able to help the less experienced designer be prepared for these.

The real value of the peer review is and continues to be exchange of design knowledge. Although intended for the purposes of mentoring, this shared conversation went much further, into the area of knowledge management. More about this later.

Landscapes
As a long-standing member of the peer review panel, one of the advantages that I experienced was to have a bird's eye view of all designs that came across our desks. We were able to begin to piece together a map of the ABSA application and infrastructure landscape, which, it appeared, was not available anywhere else. We started documenting this map, so that designers that were new to an area would not have to gather the information from scratch. It turned out that the maps were extremely useful to many other people, and we added different 'scales' of maps to our product range: a one-pager, which would allow CIO's responsible to the different business sectors to explain impacts of proposed projects to their business clients, and a five-pager, which helped both business units and designers to understand what was there before they started changing anything, and to visualize the context and constraints of a design.

Repository
As the numbers of designs increased, we needed to manage our documents. We started by putting them on a shared drive, but also needed version control and metadata-based searching, so we put them onto a Sharepoint portal, in a searchable document library. We are busy with an exercise in consolidating all of the designs and the landscapes into a single searchable repository.

Corporate System Patterns
As the designs came across our desks, we also started to notice certain patterns. There were a number of recurring kinds of designs, which needed to address particular aspects. For example, we saw a series of what were called 'White-labelling' projects, where we would provide the back-end administration of banking products for different brands. These designs almost always involved re-branding the channels (internet site, call centre, autobank), labeling the information in the common client base, and integrating to core banking applications such as accounts, payments and credit-scoring engines. We started documenting these patterns, which began to be a vocabulary for project characterization. Much more work is necessary in this area.

IX. CHALLENGES

Solution design is not without its challenges. It is by its nature demanding and difficult work, and the expectations on designers to know more about the ABSA landscape, and to produce more designs in a shorter time, is increasing. This section enumerates some of the challenges we face.

Knowledge Management
Although we keep a repository of our designs and landscapes, one of the real difficulties of solution design is to distil the right information relevant to a design problem. Much of it seems to be in people's heads, and much of that, when gathered, is inconsistent and inaccurate. This leads to us having to expend tremendous effort on acquiring, checking and documenting knowledge, an overhead not always planned for, and which extends delivery times. Using incorrect information as a basis of our designs can lead to serious credibility issues, so knowledge management is key to our continued success. An investigation is under way into more advanced knowledge management tools. A promising direction is a model of explicit design rationale, documented in an ontology, as described by Akerman [4].

Project Delivery Bottlenecks
Sometimes success can be your biggest enemy. Although the solution design team has added tremendous value through the delivery of improved designs in projects, a single design team, single peer review panel and single Enterprise Design Authority constricts production and evaluation of design artifacts into a bottleneck.

There is currently tremendous pressure on increasing IT delivery to the business. Part of the approach being taken to achieve this is to dedicate resources to particular sectors of the business. Designers are allocated to a particular business area, with the intent that they build up relationships, flesh out the relevant landscape, and more readily understand the context of designs in their area. A Principle Designer oversees the portfolio of designs and landscapes in a sector.

Another important initiative is to prioritize designs – the more important ones, where we can add more value, need more attention. We are working on a prioritization model based on novelty, size and complexity, which should cause run-of-the-mill designs to pass through the Design Authority for information only, allowing them to concentrate their efforts on the more challenging design issues.

Coordination with other Design Authorities

Being part of a larger corporate also brings its challenges. Being part of Barclays, we are participating in an increasing number of projects that cross corporate boundaries. This means that design in these projects is subject to the scrutiny of different Design Authorities. Conversations on this collaboration are challenging, and ongoing - inter-company collaboration requires both coordination of their efforts and consensus of their decisions.

X. Conclusions and Further Work

In this paper, we noted that architectural decision-making is often made implicitly and not documented well. We introduced and illustrated the concept of design forces, and suggested that phrasing architectural decisions in terms of trading off different design forces might improve the articulation of such decisions. We then described the workings of a solution design team that applied these concepts in improving the architectural design decisions in projects.

Our conclusions are that we have proven the basic concept in practice, but we face some scaling challenges.

More work is required in the documenting of technology landscapes, especially in depicting different views for different stakeholders, and the use of better tools to enable a less manual extraction of relevant model elements.

More work is also required in characterizing and articulating enterprise system design patterns, especially relating them to system qualities.

Acknowledgments

This work was carried out as part of my duties as a Solution Designer at ABSA's Republic Road IT campus. Many thanks to Diane Skinner for her leadership and foresight in building this team. Many thanks to my colleagues in the design team, especially to Alistair, Ruben, Rod, Paul and Lukas – for interesting, sometimes robust, and always challenging conversations.

References

[1] Kruchten P., "Architectural Blueprints – The 4 + 1 View Model of Software Architecture", *IEEE Software* 12 (6), November 1995, pp. 42-50.

[2] Kruchten P., "A Taxonomy of Architectural Design Decisions", *Software Architecture Workshop*, Groningen, December 2004.

[3] Brooks F, *The Mythical Man-month - Essays on Software Engineering,* 20th Anniversary edition, Addison-Wesley, 1995.

[4] Akerman A & Tyree J., "Using ontology to support development of software architectures", *IBM Systems Journal*, Vol 45, No 4, 2006.

[5] Abrams S., Bloom B., Keyser P., Kimelman D., Nelson E., Neuberger W., Roth T., Simmonds I., Tang S. & Vlissides J., "Architectural thinking and modeling with the Architect' Workbench", *IBM Systems Journal*, Vol 45, No 3, 2006.

[6] Zachman J., "A framework for information systems architecture", *IBM Systems Journal*, Vol 26, No 3, 1987.

[7] Goethals F., "An overview of enterprise architecture framework deliverables", *SAP Leerstoel*.