Table of Contents for Chapter 1

<<< … >>>

Part 1. Architecture 101 – 'The Language We Speak'

This part of the book concentrates on main concepts that give you a foundation for the overall understanding of this elusive beast we call IT Architecture. Yes, and we too will define an Architecture for you (any excuse will do).
We make an emphasis on immediate pragmatic needs of our fellow comrades IT Practitioners out there in the trenches.

Part starts with Key Concepts of architectures like business transactions, application life cycle, security etc. – just to get us started and to establish a common vocabulary of IT Architects.

Enterprise solutions are complex and ever changing infrastructures. So, next we summarise the Quality Measures for the Enterprise Architectures. Some Quality Measures can be well quantified, some not. These are important Quality Measures nevertheless, and you should keep them in mind.
You can always attach some weight or rank to Quality Measures if required, and make them somewhat quasi-quantative for the purpose of your evaluation or feasibility assessment.

One of the many beauties of Information Technology is that it is largely domain-agnostic. IT can find its useful application in any business, or any kind of human activity for that matter. In this part, we provide an overview of main Application Domains of the information technology.

As a foretaste for the rest of the book, we demonstrate how major technologies and concepts come together on the practical architecture example in its dynamics. We walk with you through the major considerations in building a feasible and scaleable enterprise solution as it evolves.

Chapter 1. Business Goals for the Enterprise Architecture - 'what is the big idea?'

This chapter describes high-level business goals, motivations and forces that compel Business to turn to the Information Technology for help.

We do not dwell here on the point of view of the Business Analyst or other expert in the Application Domain where Information Technology happen to find its purposeful implementation. This ground is sufficiently covered for you by vast literature from the business prospective.

We give an Enterprise Architect's spin to the brief discussion of the Business Goals and set a broader context for the following more technical topics.
This way, view on the Business Goals becomes somewhat philosophical. However, this is no surprise to the IT person who routinely deals with abstractions representing the real-world artefacts.
Even the most militant pragmatist requires a clear higher-level vision or broader frame of reference.

Business Challenges of the Enterprise Architect

Information Technology is Application Domain-agnostic by nature. Enterprise Architect is dealing with the Application Domain artefacts on the higher level of abstraction. This is the blessing in disguise, however.

Enterprise Architect should be able to cross the boundaries or dimensions and to maneuvre himself or herself freely in two different paradigms – Business Application Domain and the Information Technology that is employed to assist the Business.
He or she cannot afford to be a backroom techo anymore and play with some mysterious IT toys of questionable to the rest of mankind value. And the opposite extremity does not work for the Enterprise

Architect either – it is not enough to know business, you still have to be a techo, and a high-flier at that.

In order to prime himself for such a challenge, Enterprise Architect must find a fine balance between the broad understanding of technologies and the deep technical knowledge and skills.
Sounds impossible? That's not all yet. You will have to possess the knowledge and ability in the Business side of the story, in order to guide the complex integration project towards the solution that business feels comfortable and satisfied with.
But wait, there is more. Enterprise Solution implies that you build it for the people and by the people, and a lot of them. Every person may have own perceptions, pre-conceptions and idiosyncrasies. Therefore, managing expectations and perceptions is of high importance for the Enterprise Architect. Public Relations, leadership and ability to sell benefits of the technical solution to all stakeholderts (sometimes overcoming mutual misunderstandings, apathy or opposition) require some special abilities that historically often neglected by technocrats.

So, who are these Super Persons called Enterprise Architects? Do they exist in nature? How to become one?
Let's not claim that you can get all answers here. But by the end of the book you will have a good appreciation of the Enterprise Architect challenges, and a good platform for your further advancement in your well-chosen area.

IT Architecture as a Model of the Real World – 'mirror, mirror …'

We live in the infinitely complex real world. Real world has no limits in time, space, or matter. And even if there are limits, they are beyond our ability to reach them.
Universe consists of infinite number of entities, or things with their own distinguishable properties and boundaries that separate them from other entities or environment. Some of these entities are known to us, some – not, and possibly never will be. Every entity in itself is as infinitely complex as the universe itself.
Every entity exists in some context, and relates to other entities through multitude of various complex relationships and processes. These inter-dependent relationships could be electromagnetic by nature, or chemical, thermodynamic, biological, social, financial etc. We cannot even be sure that modern science is able to enumerate all kinds of relationships and processes that entities in the universe can be in, let alone comprehend them fully.

Everything tied to anything else in many different ways at the same time… Are you scared yet? Don't be, this was not the intent, but just a reminder what Creator has given us.
Appreciation of the world infinity and its infinite complexity (outward, inward, and whichever way you look) will help to illuminate the purpose and limits of the human activity in the scheme of things, and to achieve the necessary humility and broader vision in any of your endeavours, including IT and Enterprise Architecture.

Business Processes, and Human Activities in general, are influenced by the intricate combination of forces and factors. We are struggling to understand these intertwined forces to the sufficient level of depth, so that we can explain processes, comprehend what happened or happening to us, and make a reasonable guess about what is going to happen.
We are even attempting to control these processes based on our limited comprehension and, sometimes, sheer bravery. In some instances, making important decisions in a hurry, based on limited understanding of entities, relationships, and processes, may be more appropriately called 'stupidity'. But even this may be not that simple, when politics of different human agendas and higher considerations are involved.
We are trying to achieve our goals of wellbeing and progress by striving to understand, predict and influence processes in the real world. Another extreme – inaction or dropping out of the technology race – is not an option for the modern enterprise either.

So, what would be the practical lesson for the Enterprise Architect from this broadest philosophical frame of reference? In short, putting things into prospective helps you stay focused on achieving your immediate goals. And, at the same time, good grasp of the bigger picture helps you in articulating better the longer-term goals, and ways to achieve them.

We need models that capture only major comprehensible entities, processes and patterns from the real world – simple enough to be manipulated by the man, and complex enough to be sufficiently precise for the specific purpose of the man.

In essence, modelling is the process of simplification, *abstraction* from the entities, properties, relationships, and processes in the real world that are perceived as irrelevant (possibly, mistakenly) for the solution to the problem of immediate concern.
Model introduces structure and categorisation of the entities that we are going to deal with, within the *scope* or pre-defined boundaries of our problem domain.

We model real world by systematically and recursively applying techniques of *decomposition*, *layering*, *separation of concerns*, and determining the multiple orthogonal complementary *views* that summarily give us a picture of required detail and breadth of the problem domain.
These general techniques are as old as mankind itself. We can devise variations and specializations of these techniques, and label them differently, but the pattern remains the same. At large, this is how human brain works, in contast with other representatives of the animal kingdom. We apply these cognitive techniques intuitively in the routine course of our lives, and we consciously refine and improve these techniques as our accumulated knowledge and understanding grows.
Wisdom and intelligence (as in *Homo Sapiens*, also see Chapter 6) is not only about having the tools – it is also about what you do with them.

During the analysis of the problem domain, we capture the structure of our model as a first step towards building the system architecture. Having the structure implies that we are able to identify some discrete structural elements (and, more importantly, their types), or entities, components, and their interrelationships.
By applying the principle of *separation of concerns*, we achieve *decomposition* in the architecture and, conversely, the *composition* of the larger structural elements from the parts.

Layering of the architecture introduces different levels of abstractions, or layers. Every consequent layer will drill down to the greater level of detail. Architectural decisions may be different for every layer.
Multiple architectural views will be applied on every layer.

Architecture solution to the business problem may be described in various notations and methodologies. But common between all architecture description languages will be the systematic and recursive application of *decomposition*, *layering* and *views*.
Enterprise Architect may have a say in how this architecture description grid of layers and views will be constructed and populated (possibly, skipping or glossing over some views), within the guidelines of binding enterprise or industry standards for the architecture description.
Not every layer or prescribed view needs to be vigorously captured and documented on every project - large or small project, ‘green fields’ project or mundane repetition of well-known patterns. Latter is a very fortunate scenario indeed, good litmus test of the architecture quality, meaning that we can reap rewards of good architecture work that was done earlier, or due to our good expertise in building the Enterprise Architecture.

Our models of the real world, and IT models not an exception, are by necessity intentionally dummied down views, our way of dealing with complexity for the purpose of surviving, thriving, and conducting our human activities towards achieving our humble human goals.

IT is the tool that helps the man to build complex models of the real world and to control its processes. Models, that otherwise would be far beyond our physical capability of mere mortals.

Anatomy of the Technology Insight

There are more trusted and proven techniques of gaining the insight into technologies and their future evolution, other then gazing into the crystal ball.
We do not count the sixth sense, or intuition, as a viable technique for the technology insight either. Although, if we define the intuition as an informal opinion based on the wealth of experience and

understanding of the processes, but not presented in a well-structured or exhaustive fashion, we could count it too.

In mathematics, extrapolation is the foundation for conclusions known in layman terms as predictions or forecasts.
In general, we build the model (in this instance, the mathematical model) as a result of the thorough analysis of the domain in question. In order to assess how trustworthy and precise the model is, we feed the known observations into the model and play with it. Estimates for some unknown values of some observed features could be given with some probability only. But at least it is good to know how much you can rely on your estimates. Really, you would not get too excited if model just confirms back to you values that you know already (however, this is exactly what you do when you test the quality of your model).

After we satisfied ourselves with ability of the model to capture processes in the domain of our analysis, we turn the model towards the time span (usually in the future) or the unchartered part of the domain that we do not know enough about yet. We expect model to produce for us values of the observations in some time in the future (or, in some other part of the domain, or both), that otherwise would not be available to us. Good model will provide also indications of the probability, precision, and range of predicted (extrapolated) values, complemented by well-documented assumptions that brought you there.

Figure puts together basic concepts of the art of prediction.

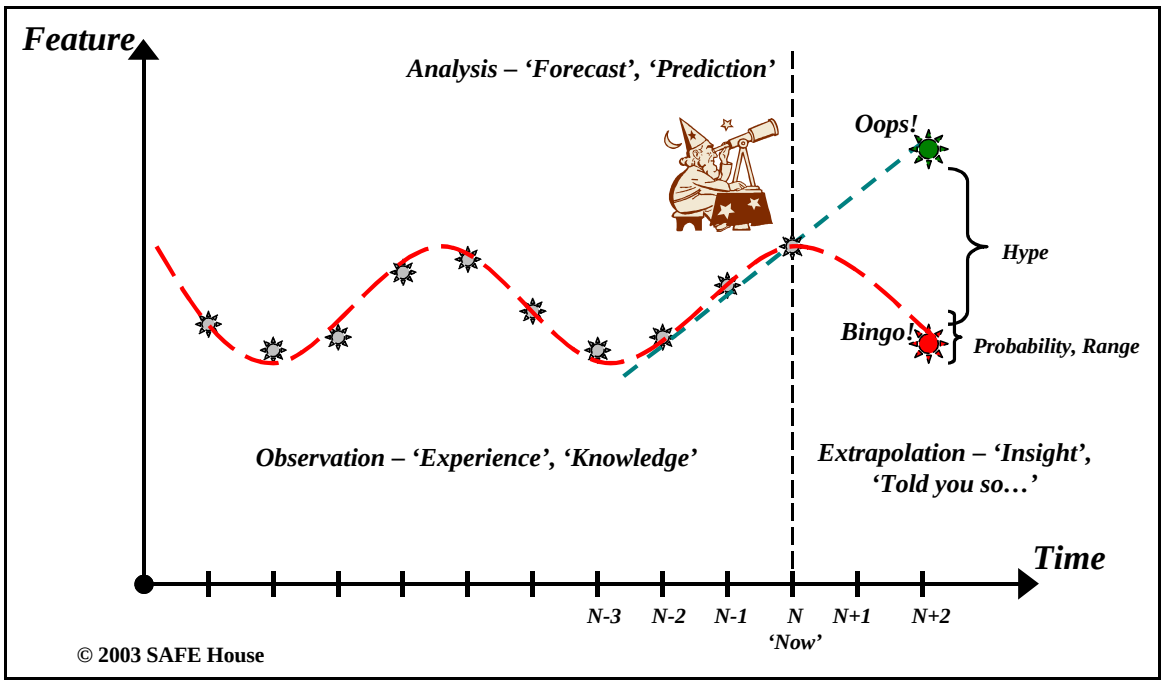

Figure 1.1 Anatomy of the Insight

Yes, let's not make mistake about it, extrapolation is still an art, especially in the complex application and technology domains that Enterprise Architects are dealing with.
Figure presents a very simplistic view of the world, just to explain concepts in the idealistic scenario. However, these are concepts that will guide you in any complex analysis nevertheless.

Figure depicts two different models that are built on two observation samples of vastly different depth. Presumably, observation samples relate to the two different levels of understanding of the domain. Resulting predictions from the two models are vastly different too.

Let's see what are the deviations from this idealistic scenario that you are going to encounter in the real-life technology assessments.

Our simplistic model is one-dimensional (if we do not count the time dimension that will be common in the analysis of any feature), i.e. we analyse observations for one feature only. Real-life products and technologies can be dissected into many separate features or parameters. Every feature may be independent (orthogonal), thus domain will be represented as a multi-dimensional view. However, features may (and will) be dependent on each other to some degree. And, features will be of different importance to the extrapolation exercise, with different impact on the result of analysis.

Furthermore, you will encounter features that are largely *qualitative*, and not easily *measurable*. We may try to *quantify* such features by introducing some levels or marks ourselves.
For instance, Level of Intelligence that may be measured by often misleading Intelligence Quotient (IQ).

Good technology insights and predictions are of high value in building the Enterprise Architectures precisely because they are hard to do, and due to the immense importance to the core business that relies on the Enterprise IT Architecture to succeed.

Usually, high quality predictions and technology forecasts fall back on the vast knowledge base and extensive observation samples from various domains over the relatively long period of time.
If history of technology dynamics was being observed over several iterations of analysis, we can judge the quality and trustworthiness of the analysis methodology, and of the analysts.
Or, in other words, on a lighter note - wisdom comes with age, but sometimes age comes alone.

There are companies that build their whole business on providing to the clients services of timely and high quality assessments of markets, technologies and products. These companies pride themselves on being independent and un-affiliated analysts of the industry trends.
Examples of such companies in no particular order include Gartner, Burton, Meta, Ovum, Forrester, IDC, Giga, Cutter, Yankee, Jupiter, just to name a few.

Enterprise Architect may come across conflicting reports on some technology from the different sources. Apart from the genuinely different opinions from different independent analysts, you may decide to take some vendor-sponsored reports with the grain of salt.

Business Requirements

"It's just what we asked for! But it's not what we want."
Dilbert, Programmer's Shortlist

Business Requirements is the layman term for the desired features of IT system that has to achieve certain business goals while complying with certain criteria.
We define reasonable boundaries within the Application Domain where we focus our attention in building the IT solution for the specific business problem. We determine what entities, their relationships, and processes will be included in our consideration. Also, we determine what transformations or additions will happen in the area when we start moulding it to achieve the stated goals.
We call this chunk of the Application Domain within the defined boundaries, and its intended transformations towards new goals, the *scope* of our project.
We hope that such a chunk can be easily de-coupled from (or, loosely coupled with) the rest of the world so we can concentrate on our particular problem, as we think we understood it. This is never the case, as we can de-couple our scope from the context only to the certain degree of granularity, complexity, or level of abstraction.

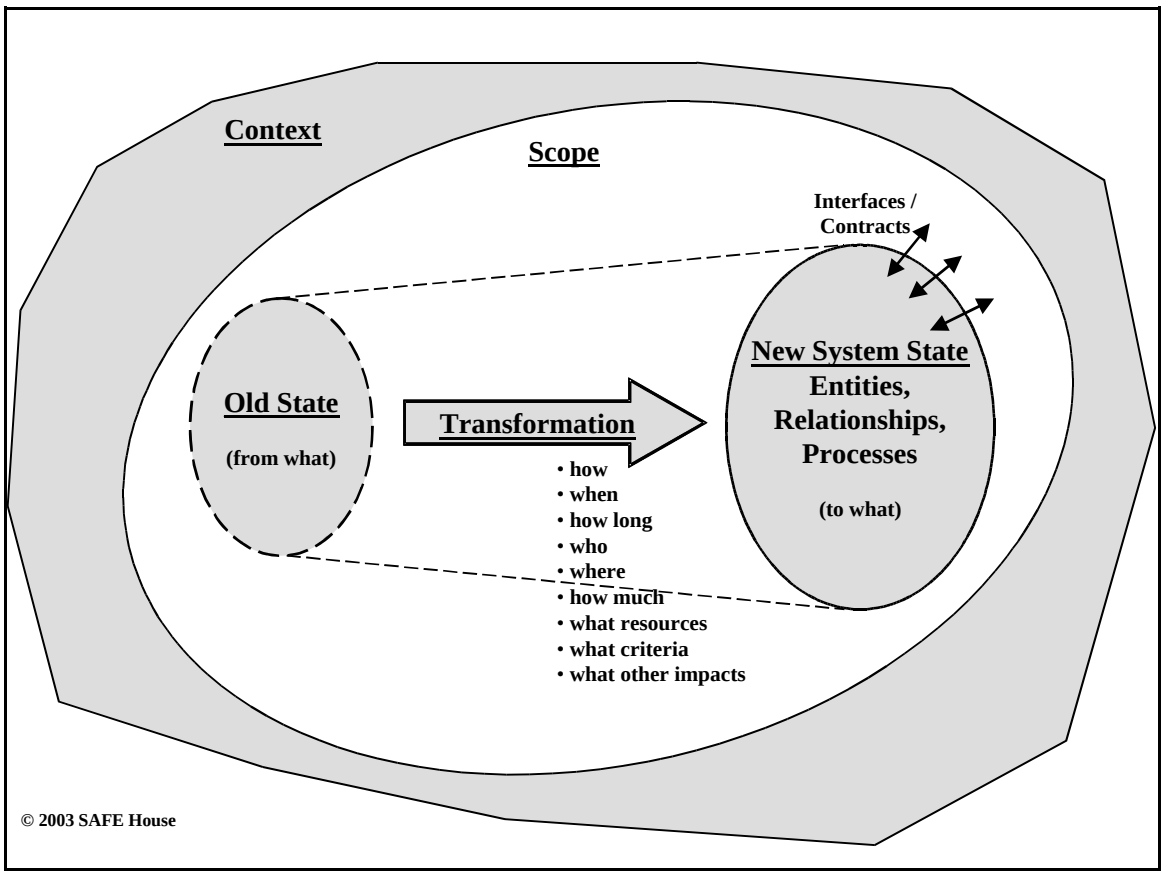

Figure 1.2 Business Requirements and Scope

Our first intuitive intent in the building the complex Enterprise Architecture is to extricate this chunk of the real world from all its bonds and dependencies on the surrounding environment. Then we would hug it, observe it from all angles in our own pace and at leisure. Then we would apply to it some well-known tools and wonderfully familiar procedures, and, oh miracle, it quietly does for us something we really wanted... We wish.
Still, we need to define the manageable scope for our project with clear boundaries showing what is in, and what is out.

Defining requirements and scope for the stand-alone project may be a challenge as it is. Enterprise Architecture will deal with multiple projects at the same time, likely with clashing goals and competing for the same limited resources. These projects may be part of the bigger project, container for other smaller projects, or overlap in some requirements with other projects. One project's external context and interfaces could be well within the other project's scope.

We learned to capture Business Requirements so that our goals for the solution we build become tangible and measurable. Business Requirements become the guiding light and goal for the build process of the Enterprise Architecture. And when we finished building, Business Requirements that we captured in the beginning become our test cases and the ultimate criteria of success.
Of course, we never expect Business Requirements and the business context of our project to change on us, while for several months we keep ourselves busy building the system, do we? …No, we are not that naïve, but can sometimes be caught off-guard or underestimate the ferocity of change.
Surely, every Enterprise Architect has own horror story to tell, with projects in the full flight cancelled or re-positioned. Dynamicity of the core business and its requirements keep IT Architecture on its toes.

Enterprise Architect should not expect to find a cosy comfort zone, not for long anyway.

Enterprise Architect is armed with good tools and techniques. However, rest assured that your tools, as well as the context and requirements of your problem domain, and, hopefully, you will change. Possibly, this change will happen often and rapidly.

See yourself rather as an agile, attentive and open-minded intellectual worker, but not as someone sitting in the ivory tower. Even if you are the king of your domain and own your IT infrastructure, you will greatly benefit from listening to the target market and from keeping in touch with IT capabilities, trends and limitations.

Functional and Non-Functional Requirements – 'do this, do that…'

Functional Requirements define specific business processes and entities that are exposed to the business user of the solution we build. These processes and entities are the artefacts, tangible to the Application Domain expert – his or her tools of trade in conducting the business transactions. Functional Requirements determine intended behavior of the system, or what the system is expected to do to achieve user goals.
We capture Functional Requirements in the form of functions, services, tasks, workflows, use cases – all are variations of some definition of the business processes.

We structure Functional Requirements so that sum of defined processes covers the whole scope of the system, consistently on the same abstraction level.
Higher-level processes may represent an aggregation of smaller processes by including them as parts into the larger whole. This is an example of a part/whole, or containment relationship between processes.
Otherwise, higher-level processes may be in 'kind of' type of relationships with more granular or detailed processes. We call such relationships specialization/generalisation, depending if you look from the more specific or more abstract point of view of the system. We say then that more specific process *extends* the more generic parent process, and provides a more specific variation of it.
The *extends* relationship is yet another term for the *inheritance* relationship (or, 'kind of', 'type of'), as opposed to the *containment* relationship ('part of') between entities and processes.
These relationships will be discussed in more detail in the context of the Object-Oriented methodology and UML. Use Cases are the intrinsic part of UML concepts and notation.

If *Functional Requirements* determine *what* behaviours system will provide, *Non-Functional Requirements* define *how*, or, rather, *how well* system will deliver these behaviours or services. Non-Functional Requirements complement system functions and services with some parameters or criteria for the quality of service, as well as timeliness, costs, resources, cohesion, technologies. Examples if Non-Functional Requirements are transaction latency, performance, availability, capacity etc.

Non-Functional Requirements define technical, operational and overall business context for the Enterprise Architecture. Non-Functional Requirements are our constraints that may not be at all tangible for the Application Domain expert (at least while the solution operates in the 'sunny day' scenario).
However, in the end of the day, it is not enough to perform a service somehow – service needs to be delivered in a viable fashion so that user can actually benefit from service in achieving his or hers business goals. For instance, if system is able to deliver a complex graph on the user's PC with analysis of the manufacturing process in the real-time, but takes forty minutes to do that, the benefit of the service may be lost completely.

Conventional categorisation of system or business requirements to *Functional* and *Non-Functional Requirements* has been convincingly challenged in [Bass 1998]. Term *Function* is the widely abused notion, and creates a lot of confusion in captuting the requirements.
Ask yourself, is *Performance* a *Non-Functional Requirement*, if poor performance does not allow reasonable access to the built functionality in the first place? How many times you gave up on website because the poor response time, or because it asks too much private information on registration, or you cannot read the web page for some reason?
Paul Clements in [Bass 1998] states that "non-functional requirement is a dysfunctional term". [Bass 1998] offers an alternative categorisation of requirements and features – *Observable* via the execution, and *Not Observable*.

Either way, methodology of capturing the requirements, being the bridge between the business and IT domains, should be well understood and adopted by all stakeholders in your Enterprise. We can strive to perfect our methodology, provided at no time we damage that bridge.

Service Level Agreement (SLA) – 'promise is promise'

Service Level Agreement is the business term for the documented and agreed upon interface between two systems or stakeholders in the complex Enterprise Solution who are responsible for its different parts.
SLA includes mutually agreed and guaranteed service quality measures, possibly with penalties for the services not delivered, or delivered with degraded quality or delay.

SLA identifies service provider(s) and service consumer(s), or vendors and clients. Each party will have their own expectations and responsibilities. SLA may have the scent of the legal document, depending on the scale and mission-criticality of services rendered.

SLA may enumerate Functional Requirements that customer expects to be delivered from the vendor. However, main emphasis in SLA is on Non-Functional Requirements.
SLA defines the most important for the service quantifiable and measurable quality criteria, basically the same quality criteria that we discuss in the Chapter 3.

And, do not forget the exit or cessation clause for the services in the agreement, as well as disclaimers for the deteriorated quality for reasons beyond your control – things may not be amicable after awhile.

Quality of Service (QoS)

Quality of Service defines some measurable quality criteria, and obligations by stakeholders to adhere to them in maintaining the required level of service. SLA documents the expectations of parties involved for the *Quality of Service*.
If the deviation from the agreed quality metrics takes place, some mitigating actions need to happen, or, agreed upon penalties apply.

Defining and maintaining metrics for the *Quality of Service* in a complex enterprise application is not a trivial task.
Enterprise Architect, together with the business analyst – application domain expert, must understand the notion of failure in delivery of the complex service to the diverse customer base.

For instance, ask any business analyst about desirable availability, and you will get likely stock-standard request for '*24x7x365*'. You may have to manage expectations, raise awareness of the impact of inflated expectations on the costs and efforts involved. Managing expectations may mean presenting the business with options that will help to find practical compromise (not necessarily perfect). Possibly, it is not viable, or not achievable through reasonable efforts and expense, or not really called for by true business imperatives, to have 100% system availability. Furthermore, enterprise environment does not fit well into 'all or nothing' scenario. Partial service, with scaled down performance, to the reduced customer base, in more important timeframes, may be sufficient just enough to keep business up and running, and to perceive service as 'available'.

Availability of service may be achieved by brute force of fault-tolerant, redundant, and highly available components of the architecture, at a cost.
Conversely, intimate knowledge of business processes and of the enterprise customer base may help in maintaining the high level of customer satisfaction and overall perception of high availability, at a lesser cost (in addition, these costs are one-off, incurred in the development phase). In other words, we can enhance the perception of availability through partitioning the business logic by application designers with special attention to the robust delivery of at least some of more critical services.

Reliable complex system can be built from unreliable components, depending on how these components are put together.

It pays to understand the mechanics of this magic. There are simple basic rules of thumb for availability of the complex system consisting of separate interconnected components or parts. Any system recursively applies two patterns of component connectivity – sequential and parallel.

Sequential control flow through components increases probability of failure somewhere in a flow. If we imagine such system as links in a chain, strength of the chain (think *Quality of Service*) is *weaker* than the weakest link. More sequential links (even if they are more reliable) only increase overall probability of failure.
For instance, assume that 'strength of the link' is availability, and every sequential link has 99% availability (by the way, this equates to 15 minutes downtime daily). Then overall system availability can only be 99% or less, if different components have their 15 minutes of fame at different times of the day each. Eg, two components may cause up to 30 minutes of system downtime a day, if their downtimes did not overlap.

Parallel connection of interchangeable redundant components increases overall availability of the cluster of these components. *Redundant* and *Interchangeable* are keywords here. Parallel components should be able to provide alternate control flow path, or replace failed component for the purpose of carrying on the interrupted service. Parallel connectivity of unreliable components into cluster achieves greater overall reliability of the cluster than the separate components it is built of.
In terms of Enterprise Architecture, clustering of parallel redundant components gives as load balancing and failover capability, towards higher overall availability and fault-tolerance.

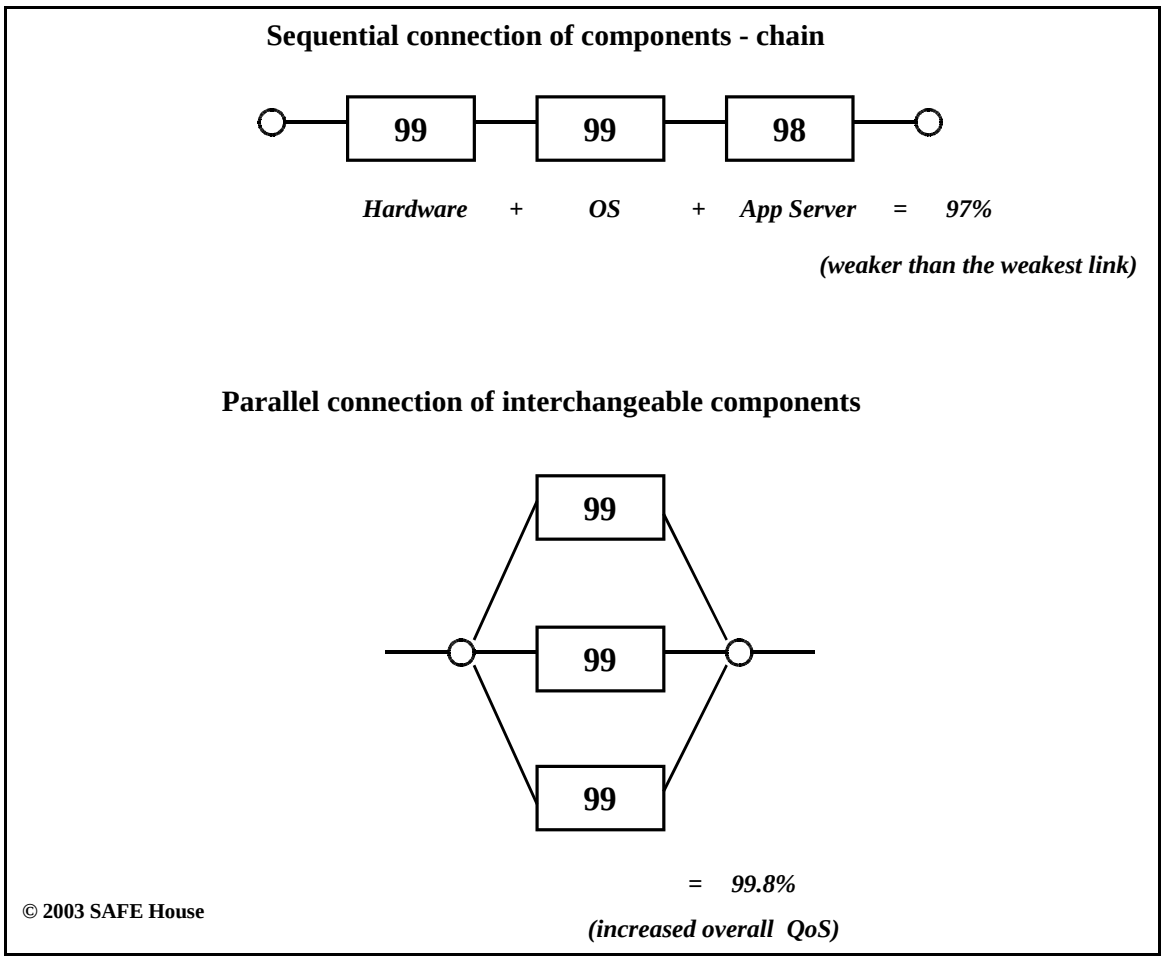

Figure 1.3 Quality of Service in the complex system

We can always pull apart complex architecture and find that component after component, and layer after layer, these two basic patterns of sequential and parallel connectivity are applied recursively. As soon as we recognised these patterns, we can consider impacts of changes in the strusture on the overall *Quality of Service*.

Business Transactions – 'let's make a deal'

Business Transaction is a fundamental high-level concept that defines tangible discrete Application Domain process performing the business function or task. Business Transaction is visible to the user as it directly represents some business process and achieves user's business goals, in the language of the business domain expert.
Business Transaction can represent the delivery of some service, or making the deal. In turn, deal may consist of several smaller Business Transactions in their own right. For instance, one withdrawal transaction and one deposit transaction may be part of the single payment transaction.

Business Transactions can differ in complexity, parties involved, volumes, and the timespan.
For example, Business Transaction could be as simple as logging-on into system (at least, simple from the user's point of view, hopefully), or as complex and prolonded as chasing the bad debt on the credit card that may involve litigation and can span years.

Notion of Transaction is rather overloaded and very important throughout IT.
Transaction may mean something different in various parts and layers of the Enterprise Architecture. Also, Transaction may mean something different in the context of specific technologies like Database Management Systems, Transaction Processing Monitors, Object Request Brokers, Message Queues, Application Servers, or Communications.

Business Transaction may have many different incarnations and compositions, as it percolates through the components of the Enterprise Architecture.
Business Transaction is the good place to start building the understanding of the overarching concept of Transaction, and the good thing to keep in mind while designing and building other related transactions in various components of the Enterprise Architecture. We shall have a closer look at the transactional mechanics in following chapters.

We capture Business Transactions by analysing the Business requirements, and faithfully implement them by mapping onto underlying transactions in the IT solution that we build.

<<< … >>>