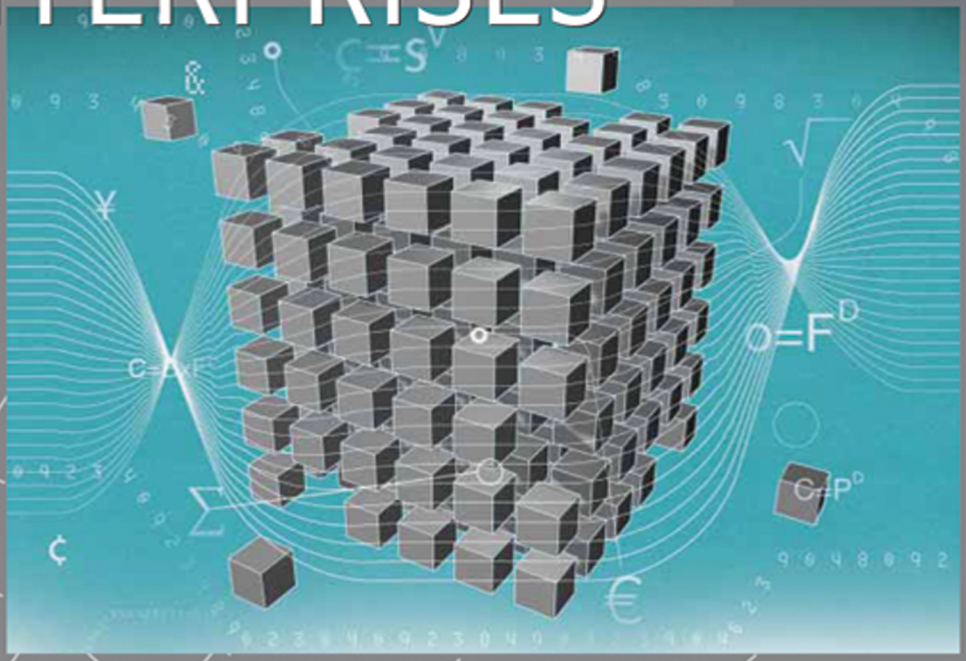
Microsoft

BEST PRACTICES

SIMPLE ARCHITECTURES FOR COMPLEX ENTERPRISES



Roger Sessions

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2008923658

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWE 3 2 1 0 9 8

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Acquisitions Editor: Ben Ryan
Developmental Editor: Devon Musgrave
Project Editor: Lynn Finnel
Editorial Production: Interactive Composition Corporation
Cover Illustration: John Hersey

Body Part No. X14-71550

'Tis the gift to be simple, 'tis the gift to be free,
'tis the gift to come down where we ought to be…

—Shaker hymn

Everything should be made as simple as possible, but not simpler.

—Albert Einstein

Contents at a Glance

Table of Contents

Acknowledgments

This book has benefited from the help of a large number of people, ranging from reviewers to photographers to production staff.

First and foremost is Beverly Bammel. Beverly is the President of ObjectWatch and has worked closely with me on developing the ideas of complexity management and on the production of this book.

Early drafts of this work have greatly benefited from reviewer feedback. My primary reviewers include

- Paulo Rocha, Manager of Enterprise Architecture for Fronde Systems Group, LTD of New Zealand
- Darko Bohinc, Principal Consultant, Strategy Services, Fronde Systems Group LTD of New Zealand
- Dan Schwartz of EDS
- Matt Peloquin, Chief Technology Officer at Construx

It has been delightful working with Microsoft Press. I especially thank Ben Ryan (my product planner), Lynn Finnel and Devon Musgrave (my editors), and Roger LeBlanc (my copy editor).

A number of photographers have graciously allowed me to reproduce their photos. The photo credits are as follows:

- The photo of the replica of Thoreau's cabin in Chapter 1 is by Laura Raney, my adopted niece, who made a special trip out to Walden's Pond just to get me this photo.
- The photos of the Rubik's Cubes in Chapter 2 are used by permission of Seven Towns Ltd, *www.rubiks.com*.
- The photo of Captain John Boyd in Chapter 2 is used by permission of his daughter, Mary Ellen Boyd, who also reviewed the material relating to Captain Boyd.
- The photos of the cockpits in Chapter 2 are used by permission of Richard and Susie McDonald of MIG Jet Adventures (*www.migjet.com*).
- The photos of the orchids in Chapter 4 are courtesy of Orchids of Wickford, N. Kingston, RI. (*www.wickfordorchids.com*).
- The photo of the Jewelweed in Chapter 4 is courtesy of Bruce Marlin, Red Planet, Inc.

Finally, there are a number of people who have contributed to this book that do not fit into any category

- Rodrigo Estrada of Neoris, for many discussions on the nature of complexity in IT systems
- Kevin Drinkwater, CIO of Mainfreight, for allowing me to recap some of our discussions on complexity
- John DeVadoss and Simon Guest of Microsoft for supporting early work relating to iteration through their sponsorship of several of my white papers
- Al Summers of Wiltshire England for allowing me to use one of our chess games as an example of partitioning
- The baristas of Starbucks in Brenham, Texas for their endless supply of Doppio Espresso Machiatos, one sugar, extra foam, preheat the cup please.

My thanks to all of you.

Legal Notices

Introduction

> *It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us…*

So begins Charles Dickens's *A Tale of Two Cities*. Dickens was writing about London and Paris in 1775. But Dickens could have been writing about the field of enterprise architecture, the science of aligning business needs and IT solutions, as it exists today.

It is the best of times. The goal of enterprise architecture is to maximize the business value delivered by IT investment. For most enterprises, large and small, nonprofit and for-profit, public and private sector, the need to maximize the return on IT investment and help IT work more effectively with the business has never been greater. No wonder interest in enterprise architecture is at an all-time high.

It is the worst of times. Enterprise architecture is supposed to ensure that IT systems deliver business value. Too often, it doesn't. Executives are losing confidence that enterprise architecture can make a real difference to IT. This crisis in confidence spans enterprise size, scope, and type. In October 2007, Gartner predicted that 40 percent of all existing enterprise architecture programs will be shut down by 2010. In their highly influential book, *Enterprise Architecture as Strategy* (Harvard Business School Press, 2006), authors Ross, Weill, and Robertson say that fewer than 5 percent of firms use enterprise architecture effectively. In my studies of enterprise architectures and their implementations, I see a common pattern of costly enterprise architectural efforts followed closely by costly IT failures. No wonder confidence in the ability of enterprise architecture to deliver value is at an all-time low.

Enterprise architecture takes a high-level view of the enterprise, focusing on the relationship between an organization's IT architecture and its business architecture. IT architectures describe IT systems. Business architectures describe business processes. IT systems that do not meet the needs of the business are wasteful. Business processes without good IT support are inefficient. Enterprise architectures describe how these two architectures complement each other, ensuring that IT systems effectively support the business processes of the organization.

Clearly, this is a good idea. And yet, enterprise architecture is failing.

What is going wrong? In my experience, there are three basic problems with existing approaches to enterprise architecture. First, existing approaches are too expensive to execute. Second, they take too much time to complete. Third, there is no way to validate their results. So, we have long, expensive processes to create architectures whose effectiveness can be tested only by building large, expensive implementations. Not only is there no way to

evaluate whether a given architecture is good or bad, most enterprise architecture methodologies don't even have a standard criteria for what "good" and "bad" mean.

How do you know whether a typical enterprise architecture is good or bad? Simple. You try to implement it. You build the IT systems that support your business processes. *If* you successfully deliver these IT systems and *if* they meet the needs of the business, then you must have had a good enterprise architecture. If not, you didn't. Better luck next time.

In any other field of science this approach would be considered absurd. Nobody would send a rocket to the moon without first testing the planned trajectory against mathematical models for planetary motion; without first testing the planned fuel levels against models for gravity and thrust. Nobody would think of building a bridge without first testing the architecture against models for stress, load, and fluid flow.

Why do we implement large, expensive enterprise architectures without first testing them against mathematically grounded models for effectiveness? The answer is simple: we don't know how. We lack a mathematical understanding of "good." We lack the models for testing for "good." We lack even a basic definition of what "good" means!

Without such models (and definitions), there is no way to validate an enterprise architecture. There is no way to predict its cost. There is no way to ensure that it will deliver business value. There is no way to know if it will even be deliverable at all. *This* is why the field of enterprise architecture is in so much trouble. *This* is why it is increasingly common to hear of massive IT failures; projects that are over budget, late, poorly aligned to business needs, or all of the above.

I was recently talking about enterprise architecture with two high-level architects in a large, highly respected, public sector IT organization. I asked them how often their IT projects came in on time, on budget, and on the mark. One of the architects looked at the other and said, "On time, on budget, and on the mark. I can't think of a single project that we have ever done that met that criteria. Can you?" The second architect only shook his head sadly. I have had similar conversations with architects, chief information officers (CIOs), and chief technology officers (CTOs) in dozens of organizations.

A recent article in *IEEE Spectrum* included this gloomy assessment:

> *Looking at the total investment in new software projects—both government and corporate—over the last five years, I estimate that project failures have likely cost the U.S. economy at least $25 billion and maybe as much as $75 billion. Of course, that $75 billion doesn't reflect projects that exceed their budgets—which most projects do. Nor does it reflect projects delivered late—which the majority are. It also fails to account for the opportunity costs of having to start over once a project is abandoned or the costs of bug-ridden systems that have to be repeatedly reworked.*[1]

[1] "Why Software Fails" in *IEEE Spectrum* (September 2005) by Robert N. Charette.

What do we do about this state of affairs? Do we give up on enterprise architectures, as Gartner predicts so many will do? No. We don't give up on the field. The goals of enterprise architectures are too important. Instead, we figure out how to do enterprise architecture right.

By *right*, I mean five things. First, we define what we mean by a *good* enterprise architecture. Second, we use this definition to build up a mathematical understanding of *good*. Third, we extend this mathematical understanding into a formal model for what a *good* enterprise architecture looks like. Fourth, we create a process for developing a *good* enterprise architecture based on that model. Fifth, we validate our resulting architectures against the model *before* we implement them.

This all starts with a good definition of *good*. So here is my definition. A *good* enterprise architecture is a *simple* enterprise architecture. Of two architectures that generally align business needs and IT capabilities, the better of the two is the simpler of the two. The worst of the two is the one that is more complex.

Now it is important not to confuse the complexity of the problems we are trying to solve with the complexity of the solutions we are trying to create. The problems on the business side are certainly complex. Businesses are struggling to adopt new technologies, deal with increasingly stricter regulatory requirements, and trade in a world that is shrinking rapidly. All of these are complex problems, and only getting more so. On the IT side, too, complexity is also the norm. Software systems are becoming more distributed, more heterogeneous, more connected, more critical to the organizations. All of these are also complex problems, and they, too, are only getting more so.

As both business and software systems become more complex, the relationships between them become harder to keep in alignment. Those working on the two sides become more specialized. They develop their own languages, even their own culture. They have less time to relate to those who do not share their overwhelming concerns. A growing separation develops between the business and the IT organizations.

In most organizations, the chasm between the IT and the business organizations is increasing. This will not be news to most readers. Most are painfully aware of the chasm. Few, if any, understand why this chasm exists. IT blames the business side. The business side blames IT. Distrust becomes widespread. Finger-pointing becomes the norm. The business people are making unreasonable demands on IT, preventing them from getting their increasingly stressful jobs done. The IT people are slowing down the business, impeding sales in an increasingly competitive environment.

But the problem is neither IT nor business. The problem is a more fundamental issue that is common to both IT and business. The real problem is complexity. And complexity is everybody's problem.

So yes, the problems are complex. But complex problems do not *ipso facto* require complex solutions. Au contraire! The basic premise of this book is that simple solutions are the only solutions to complex problems that work. The complex solutions are simply too complex.

The antidote to complexity is simplicity. Replace complexity with simplicity and the battle is three-quarters over. Of course, replacing complexity with simplicity is not necessarily simple. But this book will tell you how to do it.

The first thing you need to do to achieve simplicity is focus on simplicity as a core value. We all discuss the importance of agility, security, performance, and reliability of IT systems as if they are the most important of all requirements. We need to hold simplicity to as high a standard as we hold these other features. We need to understand what makes architectures simple with as much critical reasoning as we use to understand what makes architectures secure, fast, or reliable. In fact, I argue that simplicity is not merely the equal of these other characteristics; it is superior to all of them. It is, in many ways, the ultimate enabler.

Take security, for example. Simple systems that lack security can be made secure. Complex systems that appear to be secure usually aren't. And complex systems that aren't secure are virtually impossible to make either simple or secure.

Consider agility. Simple systems, with their well-defined and minimal interactions, can be put together in new ways that were never considered when these systems were first created. Complex systems can never be used in an agile way. They are simply too complex. And, of course, retrospectively making them simple is almost impossible.

Yet despite the importance of simplicity as a core system requirement, simplicity is almost never considered in architectural planning, development, or reviews. I recently finished a number of speaking engagements. I spoke to more than 100 enterprise architects, CIOs, and CTOs spanning many organizations and countries. In each presentation, I asked if anybody in the audience had ever considered simplicity as a critical architectural feature for any project on which they had participated. Not one person had. Ever.

The quest for simplicity is never over. Even systems that are designed from the beginning with simplicity in mind (rare systems, indeed!) will find themselves under a never-ending attack. A quick tweak for performance here, a quick tweak for interoperability there, and before you know it, a system that was beautifully simple two years ago has deteriorated into a mass of incomprehensibility. This book is not, therefore, just about how to create simple systems, but also how to keep those systems simple.

This book is not for everybody. If your organization's systems are typically on time, on budget, and successful in meeting the business needs, you don't need this book. You are either building systems that are much simpler than those that I am discussing, or you have already found a way of managing complexity. Either way, I congratulate you. You are in a lucky minority.

I recently met one such person, Kevin Drinkwater. Kevin is the CIO of Mainfreight, the largest freight company in New Zealand with more than a half billion U.S. dollars per year in revenue. Kevin is widely recognized for his innovative approach to IT and for the cost effectiveness and agility of his solutions. He literally made front-page news in New Zealand by throwing out a $13 million JD Edwards ERP implementation at a company purchased by Mainfreight and replacing it for $25,000 with a home-grown system almost overnight. He was a ComputerWorld CIO-of-the-year finalist and is a well-known speaker. Kevin is also a trusted advisor to his business units, a position that very few CIOs enjoy.

In a round-table discussion sponsored by Fronde and covered by ComputerWorld New Zealand, Kevin and I traded notes on simplicity in enterprise architectures. As I drew my pictures of an ideal simple architecture and Kevin drew his, we were both struck by their similarities. Kevin does not need me to evangelize simplicity. He and his entire IT organization eat, drink, and breathe simplicity every day. It is the core architectural requirement of everything they do. It is the primary reason that when Kevin delivers an IT solution, that solution is typically on time, on budget, and spot on the mark with regard to the business requirements.

If you are like Kevin, you don't need this book. However, if your organization's systems are typically late and over budget, and you sense a growing rift between the technical and business sides of the organization, your organization *does* need this book. If you are an IT executive, IT manager, software architect, or business analyst involved in a project whose complexity seems to be growing exponentially, you might find this book transformative.

I say this book might be transformative because it just might transform your understanding of enterprise architectures. It might change why you think we want them, how we can create them better, how we can implement them more effectively, and how they can provide greater business value.

It all comes down to simplicity. Simplicity as a core value. Simplicity as an enabler. Simplicity as a business asset. As one chief architect of a major airline recently told me, "I have been talking to many organizations about enterprise architecture. They all tell me the same thing. None of it sticks. You are the first one to discuss enterprise architecture differently. And you are the first one to make any sense." It isn't *me* that makes sense. It is *simplicity*.

How do you make things simple? Simple. Get rid of complexity. Understand it, recognize it, eliminate it, and banish it. By the time you finish this book, you will know how to do this. You will understand the mathematics of complexity, the models that govern complexity, the processes that eliminate complexity, and the validation approach that ensures complexity is no longer haunting your enterprise architectures. Your life and your architectures, then, will be so much simpler.

So while this book is ostensibly about enterprise architecture, it is really about something even more basic: simplicity. The approach to controlling complexity presented in this book

can be applied successfully to either business architectures or IT architectures. But this approach is most effective when applied at the level that includes *both* business *and* IT architectures. This is the level of enterprise architecture.

The Organization of This Book

This book starts by giving an intuitive understanding of complexity, moves to a more formal understanding, and then finally moves to a more process-focused discussion. The particular process that I advocate is called SIP, for simple iterative partitions. SIP is the only enterprise architectural methodology that specifically focuses on the problem of complexity.

Part I, "The Question of Complexity," gives a basic understanding of the issue of complexity in enterprise architectures.

Chapter 1, "Enterprise Architecture Today," gives a general introduction to the field of enterprise architecture, including an overview of the major methodologies used today and where they stand on the issue of complexity.

Chapter 2, "A First Look at Complexity," introduces in a nonmathematical way the main concepts of partitioning, iteration, and simplification, and the relationship of these three ideas to complexity control. As you will see in this chapter, you can learn quite a bit about enterprise architectures by looking at executive lunches, emergency rescues, and even chess games!

Chapter 3, "Mathematics of Complexity," gives a formal introduction to the mathematics of complexity. No mathematical background is assumed, so don't worry. We are looking at very simple dice throwing, partitioning, and Boolean math. These concepts, which are all explained from the ground up, are the basis for our model for complexity. This model helps us better understand how complexity changes as we manipulate partitions of our enterprise.

Part II, "The Quest for Simplicity," describes the specific methodology that I advocate to address complexity in enterprise architectures.

Chapter 4, "The ABCs of Enterprise Partitions," introduces the concept of an autonomous business capability (ABC). The ABC is the enterprise equivalent of a partition subset. Understanding the nature of ABCs and how they relate to each other sets the stage for the methodology we will use to create enterprise architectures that embrace simplicity as a core value.

Chapter 5, "SIP Process," describes the methodology of simple iterative partitions (SIP) in detail. This is our methodology for controlling complexity. It is grounded in the mathematics of complexity and is based on identifying, manipulating, repartitioning, and reorganizing ABCs.

Chapter 6, "A Case Study in Complexity," looks at an actual case study of a highly complex project, the National Programme for IT, part of Britain's National Health Care System. If you think you have seen complexity before, just wait. This system has already cost billions of dollars, brought several companies to the brink of financial disaster, and, most likely, will end up with the dubious distinction of being the world's largest IT failure. This chapter discusses what went wrong and how the SIP methodology could have helped save this project.

Chapter 7, "Guarding the Boundaries: Software Fortresses," looks at the software components of ABCs and discusses some of the special challenges they face in maintaining the integrity of the boundaries separating autonomous systems. I'll describes a pattern called *software fortresses* that allows you to apply the simplification algorithms of SIP to software systems.

Chapter 8, "The Path Forward," reviews the main points of this book and describes how you can take your new understanding of complexity and use it to drive a corporate culture that embraces simplicity.

The book then concludes with an appendix, "This Book at a Glance," which gives a concise description of the main mathematical rules, the SIP methodology, and the software fortress model. After this, you will be a bona fide member of the Anti-Complexity League, ready to defend the simplicity of your enterprise architecture against every insidious attack.

Find Additional Content Online

As new or updated material becomes available that complements your book, it will be posted online on the Microsoft Press Online Developer Tools Web site. The type of material you might find includes updates to book content, articles, links to companion content, errata, sample chapters, and more. The Web site will be available soon at *http://www.microsoft.com/learning/books/online/developer*, and will be updated periodically.

Support for This Book

Microsoft Press provides support for books and companion content at the following Web site: *http://www.microsoft.com/learning/support/books/*.

Questions and Comments

If you have comments, questions, or ideas regarding the book or the companion content, or if you have questions that are not answered by visiting the sites previously listed, please send them to Microsoft Press via e-mail to

mspinput@microsoft.com

Or via postal mail to

Microsoft Press
Attn: *Simple Architectures for Complex Enterprises* Editor
One Microsoft Way
Redmond, WA 98052-6399

Please note that Microsoft software product support is not offered through the above addresses.

Chapter 1

Enterprise Architecture Today

This book is about how to do enterprise architecture better. This immediately brings up the question, better than what? Better than we do things today. I will address issues that I think are important that are not addressed by today's methodologies. The most important of these issues is, of course, complexity.

But before I can discuss how I think things need to be improved, you need to understand the current state of the art. What are the methodologies that I think need improvements? How do these methodologies address complexity, if at all?

Most enterprise architects have some experience with one of these methodologies, but few enterprise architects have a broad perspective on the field. In this chapter, I will give some background about the field of enterprise architecture. I will discuss why the field exists and what the field looks like today. I will compare the major enterprise architecture methodologies in use and their relationship to each other.

Each of these methodologies has important contributions to make to the practicing enterprise architect's tool chest. And although most people treat these methodologies as mutually exclusive (you can use Zachman, TOGAF, or FEA), in reality they are complementary and you should have an awareness of what each can contribute to solving the problems at hand.

But just as you should be aware of what each of these methodologies can contribute, you should also be aware of what each lacks. None of these methodologies provides a complete solution to creating an enterprise architecture. Even all of them combined do not offer a complete solution. This is my reason for writing this book: to fill in the missing piece.

The missing piece is a way to manage complexity. These methodologies can help you understand your business processes and how to better serve those processes with technology. But they can only do so effectively if you have first brought some order to the enterprise. This book will show you how to tame your enterprise to the point where these methodologies can be brought into play effectively.

So this chapter is really about the current state of the art of enterprise architecture. What works, what doesn't work, and what is needed to complete the picture.

Why Bother?

Creating an enterprise architecture is a significant undertaking for an organization, requiring time, resources, and cultural change. Why should an organization bother?

I'll go through some of what are, in my experience, typical concerns that lead enterprises to consider creating an enterprise architecture and how a successful enterprise architecture can deliver value by addressing those concerns. If one or more of these enterprise concerns seems applicable to your organization, you are a good candidate to consider implementing an enterprise architecture. If not, consider yourself lucky.

Issue: Unreliable Enterprise Information

Enterprises are dependent on reliable information about their operations to make good business decisions. An enterprise that either cannot access or trust its information will, at best, be constantly second-guessing its decisions and, at worst, make decisions based on inaccurate information. Either is a serious problem.

Unreliable information is frequently a result of data duplication across multiple information technology (IT) systems that span multiple uncoordinated business processes.

An enterprise architecture can help an organization understand what information is unreliable, how it affects the organization, and what steps are necessary to solve the problem.

Issue: Untimely Enterprise Information

Enterprises are dependent on not only reliable information (as previously mentioned) but on information being presented in a timely fashion. Enterprises need timely information to make agile business decisions. Enterprises that do not have access to timely information end up making business decisions based on stale information, which is like playing chess without being allowed to know your opponent's last move. These enterprises will find it difficult to compete against enterprises that are making decisions based on what is happening now.

Untimely information is frequently a result of highly human-driven operations. Human-driven operations is a sign that IT is not well aligned with the business needs.

An enterprise architecture can help an organization understand how to better use technology and reduce the dependencies on human operations.

Issue: New Complex Projects Underway

Enterprises that are preparing to undertake highly complex IT projects are often concerned about managing that complexity and, if they are not concerned, they should be. Building a new, highly complex IT project without fully understanding its relationship to the business processes is unlikely to be successful.

An enterprise architecture can be critical to helping the IT department understand exactly what the business needs are before it begins a new project, greatly increasing the odds that the project will be successful.

Issue: New Companies Being Acquired

When one company acquires another, it can be very difficult to merge the two sets of business processes and IT systems.

An enterprise architecture for both organizations can be a great help in seeing how the business processes and IT systems complement each other and how they can be merged together.

Issue: Enterprise Wants to Spin Off Unit

Sometimes a company wants to sell off some unit of the business. The value of any business unit is greatly increased if it is autonomous from the rest of the business and easily integrated into another organization's operations. That autonomy also helps ensure that the enterprise that remains is minimally affected by the spinoff.

An enterprise architecture can help an organization understand the impact on the business processes and IT systems of the spinoff.

Issue: Need to Identify Outsourcing Opportunities

Frequently, enterprises decide to focus on their core strengths and outsource support functions. This business strategy requires an understanding of how the core IT systems and business processes relate to the support IT systems and business processes.

An enterprise architecture can help an organization understand where the opportunities for outsourcing exist, and how it can be accomplished with minimal disruption of operations.

Issue: Regulatory Requirements

Governments around the world are taking a hard line on how enterprises manage their information. Privacy regulations require companies to prove that only authorized individuals can access various types of information. Auditing regulations require that organizations can trace back data changes to specific business process events. Many enterprises are faced with trying to meet these regulations with highly convoluted software systems wherein data is randomly shared in often unexpected and undocumented patterns.

An enterprise architecture can help business managers understand the data usage patterns and how those patterns relate to business functions.

Issue: Need to Automate Relationships with External Partners

The clear trend in the business world is to automate relationships between partners. The line between retailers and suppliers is becoming increasingly blurred, with suppliers sometimes having access to inventory information in ways that would have been unthinkable a decade ago. Much of this automation makes use of industry-standard Web services for passing messages between partners.

Enterprises that seek to participate in these relationships need to have well-defined business processes that are closely aligned with their IT systems.

An enterprise architecture can help define those business processes and pinpoint opportunities for automation.

Issue: Need to Automate Relationships with Customers

Today's customers expect online access to search for merchandise, place orders, check the status of orders, and look for product support information. From the customer perspective, such capability is convenient. From the business perspective, such capability is highly cost-effective. This is a win-win situation.

An enterprise architecture can help determine how customers can access business systems without compromising necessary protection of data and protected business functions.

Issue: Poor Relationship Between IT and Business Units

In many enterprises, we can see the alarming trend of creating a separation between IT groups and business groups. I discussed this problem in the Preface. The IT group sees the business groups as unreasonable. The business groups see the IT group as unable to deliver the desired functionality. The IT group does more and more without consulting the business groups. The business groups try more and more to circumvent IT. Distrust between the groups becomes normal and even expected. Clearly, this is an unhealthy situation for an organization.

An enterprise architecture can provide a neutral watering hole at which both the business and IT groups can meet and discuss how best to work together.

Issue: Poor Interoperability of IT Systems

Many enterprises have a large and rapidly evolving collection of IT systems that were developed and/or acquired independently and built on incompatible platforms. The IT group is left with the challenge of getting these systems to coordinate their work and share information. Frequently, these systems are glued together in a patchwork fashion. Often the juncture

points are poorly documented, highly fragile, and unreliable. The result is systems that are tied together in random ways, with failures in one system propagating in unpredictable ways to other systems. The IT backbone of an enterprise is only as strong as its weakest link. For far too many organizations, there are far too many of these weak links.

An enterprise architecture is critical to understanding how to improve the interoperability of these systems.

Issue: IT Systems Unmanageable

As I mentioned in the last section, IT systems are frequently built up piecemeal and patched together haphazardly. In addition to creating the interoperability problem that I just discussed, this cobbling together of IT systems also often results in what I call *pinned architectures*—that is, architectures in which one system can't easily be changed because any changes could affect other systems in unacceptable and sometimes unpredictable ways. When changes must be made, it becomes very expensive, and very risky, to do so.

An enterprise architecture is the starting point to understanding how IT systems are related to each other.

The Value of Enterprise Architecture

Do any of these issues seem familiar? If so, your organization can probably benefit from creating an enterprise architecture. Are any of these problems your problems? If so, at least part of your job is the role of an enterprise architect, regardless of the title that might be on your business card.

Later, you will see how finding the solutions to these problems can be greatly aided by having a defined approach to managing complexity. But let's start by seeing where most methodologies are today.

Common Definitions

Before I get too far into discussing enterprise architecture, I need to define some terms. These definitions are especially important in comparing methodologies, because different methodologies sometime use similar terms to mean different things.

For example, we have two popular methodologies that describe themselves as *enterprise architectural frameworks*: the Zachman Framework for Enterprise Architectures and The Open Group Architectural Framework (TOGAF). Yet these two methodologies share little in common other than the words *enterprise*, *architecture*, and *framework*. Even using the term *methodology* to describe these two approaches is questionable. As John Zachman himself has

reminded me on more than one occasion, his approach is a *classification scheme* for organizing systems, not a *method* for doing anything.

So I will start by defining the terms as I will use them in this book:

- **Architect** One whose responsibility is the design of an architecture and the creation of an architectural description.
- **Architectural artifact** A specific document, report, analysis, model, or other tangible item that contributes to an architectural description.
- **Architectural description** A collection of architectural artifacts that collectively document an architecture.
- **Architectural framework** A skeletal structure that defines suggested architectural artifacts, describes how those artifacts are related to each other, and provides generic definitions for what those artifacts might look like.
- **Architectural methodology** A generic term that can describe any structured approach to solving some or all of the problems related to architecture.
- **Architectural process** A defined series of actions directed to the goal of producing either an architecture or an architectural description.
- **Architectural taxonomy** A methodology for organizing and categorizing architectural artifacts.
- **Architecture** The fundamental organization of a system, including how that system is related to its environment and what principles guided its design and evolution.
- **Enterprise architecture** An architecture in which the system in question is the whole enterprise, especially the business processes, technologies, and information systems of the enterprise.

What Is Enterprise Architecture?

The definition just given of an enterprise architecture is pretty high level. Because enterprise architecture is the topic of this book, let's look at the term in a bit more depth.

According to Carnegie Mellon University (home of some of the thought leaders in this field), an enterprise architecture is defined as follows:

> *A means for describing business structures and processes that connect business structures.*[1]

Although it's succinct, this definition does not capture the business justification for trying to build an enterprise architecture.

[1] Carnegie Mellon University, www.sei.cmu.edu/ architecture/glossary.html.

Wikipedia goes further with its definition of enterprise architecture:

> *The practice of applying a comprehensive and rigorous method for describing a current or future structure for an organization's processes, information systems, personnel and organizational sub-units, so that they align with the organization's core goals and strategic direction.*[2]

The Wikipedia definition gives a better hint of the exhaustive nature of so many enterprise architectures, and even contains a hint as to their value, but it still focuses on the *how* rather than the *why*.

Here is my definition of enterprise architecture, one that focuses on the benefits of an enterprise architecture:

> *An enterprise architecture is a description of the goals of an organization, how these goals are realized by business processes, and how these business processes can be better served through technology.*

In fact, this definition could be simplified even further: enterprise architecture is the art of maximizing the value of IT investments. As I will discuss, the ability to maximize the value of IT investments is largely dependent on our ability to manage the most fundamental impediment to realizing value. But this is getting ahead of the story.

The goal of an enterprise architecture should not be to document every business process, every software system, and every database record that exists throughout the organization. It should be about adding business value.

If adding business value is not the bottom line of an enterprise architecture, the energy put into creating that enterprise architecture has been badly misplaced. If one can achieve this goal without going through a costly, time-consuming process, then I say, so much the better. It is the ends that are important, not the means.

Some of the confusion about enterprise architectures begins with the term *architecture* itself. The word "architecture" implies blueprints. Blueprints are known for their completeness, specifying everything from how the roof connects to the walls, to how the pipes are laid, to where the electrical sockets are located, and so on. Although many enterprise architecture methodologies attempt to capture this level of detail, the effort rarely pays off.

When looking at how to use technology to add business value, we need answers to these questions:

- What are the overall goals of the business?
- How is the business organized into autonomous business processes?

[2] Wikipedia, http://en.wikipedia.org/wiki/Enterprise_architecture.

- How are those business processes related to each other?
- Which business processes (or relationships between processes) seem particularly amenable to improvement through technology?
- What is the plan for making those improvements?

There is no such thing as a finished enterprise architecture. Instead, an enterprise architecture should be seen as a living set of documents that guides the use of technology. It is actually much more analogous to a city plan than to a building blueprint.

Using the analogy of a city plan to describe an enterprise architecture was a comparison first made by Armour in 1999[3] and is particularly relevant for today's highly complex organizations.

A city plan addresses different issues than do building blueprints. City plans address issues such as the following:

- What type of buildings will be allowed in which zones (for example, business or residential)?
- How do buildings connect to the city infrastructure (for example, in terms of plumbing and electrical)?
- What impact will buildings have on others of their ilk (for example, on air quality and traffic flow)?
- Are the buildings built to a standard that will not endanger their inhabitants (for example, are they fire and earthquake resistant)?

Imagine a city that included in its city plan a detailed blueprint for every building that would ever be built in the city. Such a plan would be extremely expensive to create, and, if it was ever completed, would be inflexible and stifling. Which, come to think of it, is not unlike some enterprise architectures I have seen.

Complexity in Enterprise Architectures

This field of enterprise architecture was inaugurated more than 20 years ago to address two major problems in the field of information technology that were already becoming apparent. The first problem was managing the increasing complexity of information technology systems. The second problem was the increasing difficulty in delivering real business value with those systems.

[3] Arm - A big-picture look at enterprise architectures by Armour, F.J.; Kaisler, S.H.; Liu, S.Y. in IT Professional Volume 1, Issue 1, Jan/Feb 1999 Page(s):35–42.

As you can imagine, these problems are related. The more complex a system, the less likely it is that it will deliver maximum business value. As you better manage complexity, you improve your chances of delivering real business value.

As systems become more complex, they generally require more planning. It is easy to see this in buildings. When Henry David Thoreau built his little cabin on Walden's Pond (shown in Figure 1-1), he embraced simplicity and needed no architects. If you are building New York City (shown in Figure 1-2), simplicity is out of the question and you will need many architects.

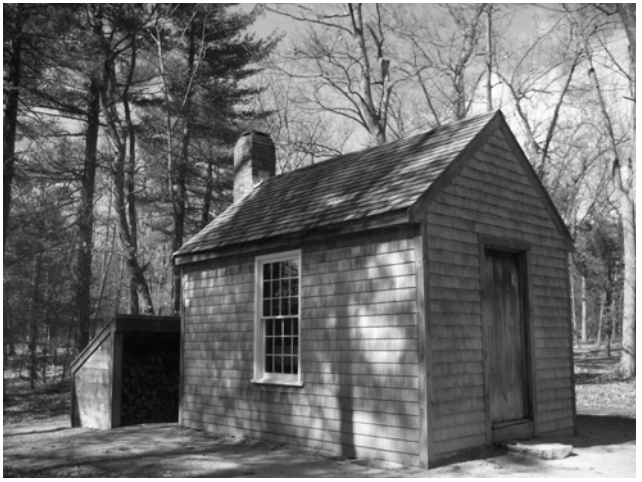


FIGURE 1-1 Replica of Thoreau's cabin at Walden Pond.

FIGURE 1-2 New York City.

The relationship between complexity and planning for buildings and cities is similar for information systems. If you are building a simple, single-user, nondistributed system, you might need no architects at all. If you are building an enterprisewide, mission-critical, highly distributed system, you might need a database architect, a solutions architect, an infrastructure architect, a business architect, and an enterprise architect.

This book concerns the responsibilities of the enterprise architect. This is the architect who specializes in the broadest possible view of architecture within the enterprise. This is the architect's architect, the architect who is responsible for coordinating the work of all the other architects. Do you need such an architect? It all depends on what you are building: Thoreau's cabin or New York City.

Building a large complex IT system without an enterprise architect is like trying to build a city without a city planner. Can you build a city without a city planner? Probably. Would you want to live in such a city? Probably not.

Of course, having a city planner does not guarantee a livable city will be built, it merely improves the chances of that happening. Similarly, having an enterprise architect does not guarantee a successful enterprise architecture will be built. There are many examples of failed enterprise architectures in the world today, and all of them had enterprise architects (probably dozens!). But there is one thing that these failed enterprise architects didn't have, and that is a methodology for controlling complexity.

This seems like an odd statement, given that I said that the field of enterprise architecture was started in part, to address the very issue of complexity. However, as I present the major enterprise architectural methodologies in use today, you will notice that none define what complexity looks like, how it should be controlled, or how one can validate that one has successfully eliminated complexity. In fact, most methodologies have become more focused on process rather than deliverables.

And yet, the problem of complexity has never been greater. Over the last decade, the cost and complexity of IT systems have exponentially increased while the chances of deriving real value from those systems have dramatically decreased. The bottom line: more cost, less value. These problems, first recognized 20 years ago, have today reached a crisis point. Large organizations can no longer afford to ignore these problems. The warnings about overcomplexity that 20 years ago seemed quaintly quixotic today seem powerfully prophetic.

Enterprise architectures can be a tremendous asset in finding effective ways to better use technology. You can't afford to ignore the potential of a well-done enterprise architecture. These benefits include decreased costs, improved processes, more agile business solutions, and expanded business opportunities.

But you also can't afford to ignore the risks of getting mired in a bad enterprise architecture. These include astronomical expenses, technological gridlock, and even further diminished

IT credibility. They can also be a huge counterproductive drain on precious organizational resources. All too often, it is this final case that is realized.

What differentiates successful enterprise architectures from unsuccessful ones? In my experience, success in enterprise architecture is almost entirely correlated to complexity. The more complex the enterprise architecture, the less likely the enterprise architecture is to be successful. In other words, the more you need an enterprise architecture, the less likely it is to actually be successful.

As a good example of such failures, we need look no further than the U.S. federal government. It is likely that no organization in the world has dedicated more money, time, and effort to creating and leveraging an effective architecture. How has the U.S. government done?

Apparently, not too well. Hardly a month goes by in which the Government Accountability Office (GAO), an independent watchdog branch of the U.S. government, does not issue a scathing report on the information technology practices of at least one agency. It seems that the more crucial the government agency is, the more likely it is to have major IT failures.

In November 2005, the GAO noted these IT problems with the IRS:

> *The lack of a sound financial management system that can produce timely, accurate, and useful information needed for day-to-day decisions continues to present a serious challenge to IRS management. IRS's present financial management systems...inhibit IRS's ability to address the financial management and operational issues that affect its ability to fulfill its responsibilities as the nation's tax collector.*[4]

The Department of Defense has come under repeated criticism. For example, in June 2005, the GAO issued a report saying

> *DOD's substantial financial and business management weaknesses adversely affect not only its ability to produce auditable financial information, but also to provide accurate, complete, and timely information for management and Congress to use in making informed decisions. Further, the lack of adequate accountability across all of DOD's major business areas results in billions of dollars in annual wasted resources in a time of increasing fiscal constraint and has a negative impact on mission performance.*[5]

---

4 GAO Report to the Secretary of the Treasury November 2004 FINANCIAL AUDIT IRS's Fiscal Years 2004 and 2003 Financial Statements.

5 Testimony Before the Subcommittee on Government Management, Finance, and Accountability, Committee on Government Reform, House of Representatives; DOD BUSINESS TRANSFORMATION - Sustained Leadership Needed to Address Long-standing Financial and Business Management Problems (June, 2005).

The highly visible Department of Homeland Security has had many problems. In an August 2004 report, GAO had the following to say:

> *[DHS] is missing, either in part or in total, all of the key elements expected to be found in a well-defined architecture, such as descriptions of business processes, information flows among these processes, and security rules associated with these information flows, to name just a few.... Moreover, the key elements that are at least partially present in the initial version were not derived in a manner consistent with best practices for architecture development.... As a result, DHS does not yet have the necessary architectural blueprint to effectively guide and constrain its ongoing business transformation efforts and the hundreds of millions of dollars that it is investing in supporting information technology assets.*[6]

The list goes on and on. The FBI has sustained heavy criticism for squandering more than $500 million in a failed effort to create a virtual case filing system. FEMA spent than $100 million on a system that was proven ineffective by Hurricane Katrina. Other federal government groups that have been the subject of GAO criticism include the Census Bureau, Federal Aviation Authority, National Air and Space Administration, Housing and Urban Development, Health and Human Services, Medicare, and Medicaid.

If the federal government is the most comprehensive case study that we have on the value of enterprise architectures, the field is in a pretty sorry state.

Although private industry failures are not as prone to make headlines, the private sector, too, is perfectly capable of bungling enterprise architecture. Private sector failures that seem largely attributed to failures in enterprise architectural methodologies include the following:

- McDonald's failed effort to build an integrated business management system that would tie together its entire restaurant business. Cost: $170 million.[7]
- Ford's failed attempt to build an integrated purchasing system. Cost: $400 million.[8]
- KMart's failed attempt to build a state-of-the-art supply chain management system. Cost: $130 million.[9]

Unfortunately, complexity is not a passing whim. There are three predictions that we can confidently make about the future of enterprise architecture:

- Complexity is only going to get worse.
- If we don't find approaches to managing complexity, we are doomed to fail.
- The existing approaches don't work.

---

6 GAO Report to the Subcommittee on Technology, Information Policy, Intergovernmental Relations and the Census, Committee on Government Reform, House of Representatives August 2004 HOMELAND SECURITY Efforts Under Way to Develop Enterprise Architecture, but Much Work Remains.

7 McDonald's: McBusted by Larry Barrett and Sean Gallagher in Baseline, July 2, 2003.

8 Oops! Ford and Oracle mega-software project crumbles by Patricia Keefe in ADTMag, November 11, 2004.

9 Code Blue by David F. Carr and Edward Cone in Baseline, November/December 2001.

As Richard Murch succinctly put it in a recent article in *InformIT*:

> *To let IT infrastructures and architectures become increasingly complex with no action is unacceptable and irresponsible. If we simply throw more skilled programmers and others at this problem, chaos will be the order of the day.... Until IT vendors and users alike solve the problem of complexity, the same problems will be repeated and will continue to plague the industry.*[10]

The problem, in a nutshell, is that while organizations have become much more complex in the last 10 years, the methodologies have remained largely stagnant. As The Royal Academy of Engineering and the British Computer Society noted in a 2004 large-scale study of IT complexity:

> *...current software development methods and practices will not scale to manage these increasingly complex, globally distributed systems at reasonable cost or project risk. Hence there is a major software engineering challenge to deal with the inexorable rise in capability of computing and communications technologies.*[11]

My goal in writing this book is to give the practicing enterprise architect some new strategies that are specifically focused on the problem of complexity. But before we discuss these new strategies, let's look at where the field is today, and how it got there.

The Zachman Framework for Enterprise Architectures

The first and most influential enterprise architecture methodology is the Zachman Framework, which was first introduced in 1987 by John Zachman.

The first thing we need to understand about the Zachman Framework is that it isn't a framework, at least by my definition of a framework. According to the American Heritage Dictionary, a framework is defined as

> *A structure for supporting or enclosing something else, especially a skeletal support used as the basis for something being constructed; An external work platform; a scaffold; A fundamental structure, as for a written work; A set of assumptions, concepts, values, and practices that constitutes a way of viewing reality.*[12]

A *taxonomy*, on the other hand, is defined as

> *The classification of organisms in an ordered system that indicates natural relationships; The science, laws, or principles of classification; systematics; Division into ordered groups or categories.*[13]

---

[10] Managing Complexity in IT, Part 1: The Problem in InformIT, Oct 1, 2004 By Richard Murch.

[11] The Challenges of Complex IT Projects: The report of a working group from The Royal Academy of Engineering and The British Computer Society, April, 2004.

[12] "framework." The American Heritage® Dictionary of the English Language, Fourth Edition. Houghton Mifflin Company.

[13] "taxonomy." The American Heritage® Dictionary of the English Language, Fourth Edition. Houghton Mifflin Company.

The Zachman "Framework" is actually a taxonomy for organizing architectural artifacts (that is, design documents, specifications, models) that takes into account both whom the artifact targets (for example, business owner, builder) and what particular issue (for example, data, functionality) is being addressed.

As John Zachman retrospectively described his work:

> *The [Enterprise Architecture] Framework as it applies to Enterprises is simply a logical structure for classifying and organizing the descriptive representations of an Enterprise that are significant to the management of the Enterprise as well as to the development of the Enterprise's systems.*[14]

Many proponents of the Zachman Framework see it as cross disciplinary, with influence extending far beyond IT. One popular book on Zachman, for example, says the following:

> *…in due course, you will discover that the Framework exists in everything you do, not only IT projects. When you thoroughly understand the Framework, you can become more effective in everything you do. This means* everything. *This statement is not made lightly.*[15] *[Emphasis in original.]*

John Zachman himself told me the following in an interview that I conducted with him:

> *…the Framework schema has been around for thousands of years, and I am sure it will be around for a few more thousands of years. What changes is our understanding of it and how to use it for Enterprise engineering and manufacturing.*[15]

Zachman originally explained his IT taxonomy using the building industry as an analogy. In that industry, architectural artifacts are implicitly organized using a two-dimensional grid. One dimension of the grid is the various "players in the game." For a physical building, some of these players are the owner (who is paying for the project), the builder (who is coordinating the overall construction), and a zoning board (who is ensuring that construction follows local building regulations).

A building architect prepares different artifacts for each of these players. Every player demands complete information, but what constitutes completeness differs for the various players. The *owner* is interested in a complete description of the functionality and aesthetics of the building. The *builder* is interested in a complete description of the materials and construction process. The owner doesn't care about the placement of studs in the walls. The builder doesn't care how the bedroom windows line up with the morning sun.

---

[14] The Framework for Enterprise Architecture: Background, Description and Utility by John A. Zachman, published by Zachman Institute for Framework Advancement (ZIFA) Document ID 810-231-0531.

[15] Enterprise Architecture Using the Zachman Framework by Carol O'Rourke, Neal Fishman, and Warren Selkow. Published by Thomson Course Technology 2003. ISBN 0-619-06446-3.

As Zachman said in his original article:

> *...each of the architectural representations differs from the others in essence, not merely in level of detail.*[16]

The second dimension for architectural artifact organization is the descriptive focus of the artifact—the what, how, where, who, when, and why of the project. This dimension is independent of the first. Both the builder and the owner need to know *what*, but the owner's need to know *what* is different than the builder's need to know *what*. What *what* is what depends on who is asking the question.

In his first paper and Zachman's subsequent elaboration in 1992[17], Zachman proposed that there are six descriptive foci (data, function, network, people, time, and motivation) and six player perspectives (planner, owner, designer, builder, subcontractor, and enterprise). These two dimensions can be arranged in a grid as shown in Figure 1-3.

Take the column describing data, as an example. From the business owner's perspective, "data" means business entities. This can include information about the entities themselves, such as customers and products, or information about relationships between those entities, such as demographic groups and inventories. If you are talking to a business owner about data, this is the language you should use.

From the perspective of the person implementing the database, "data" does not mean business entities, but rows and columns organized into tables and linked together by mathematical joins and projections. If you are talking to a database designer about data, don't talk about customer demographic groups; instead, talk about third-normal relational tables.

It's not that one of these perspectives is better than the other or more detailed than the other or of a higher priority than the other. *Both* of these perspectives on data are critical to a holistic understanding of the system's architecture. As Zachman said:

> *We are having difficulties communicating with one another about information systems architecture, because a set of architectural representations exists, instead of a single architecture. One is not right and another wrong. The architectures are different. They are additive and complementary. There are reasons for electing to expend the resources for developing each architectural representation. And there are risks associated with not developing any one of the architectural representations.*[18]

---

[16] A framework for information systems architecture, by J.A. Zachman in IBM Systems Journal, 26 3, 1987.

[17] Extending and formalizing the framework for information systems architecture, by J.F. Sowa and J.A. Zachman in IBM Systems Journal, 31 3, 1992.

[18] A framework for information systems architecture, by J.A. Zachman in IBM Systems Journal, 26 3, 1987.

As I mentioned earlier, the Zachman Framework consists of six functional foci, each considered from the perspective of a major player. The Zachman framework as it is portrayed today is shown in Figure 1-3.

As you can see in Figure 1-3, there are 36 intersecting cells in a Zachman grid, one for each meeting point between a player's perspective (for example, business owner) and a descriptive focus (for example, data). As we move horizontally (for example, left to right) in the grid, we see different descriptions of the system, all from the same player's perspective. As we move vertically in the grid (for example, top to bottom), we see a single focus but change the player from whose perspective we are viewing that focus.

There are three suggestions of the Zachman grid that can help in the development of an enterprise architecture.

The first suggestion of the Zachman taxonomy is that every architectural artifact should live in one and only one cell. There should be no ambiguity about where a particular artifact lives. If it is not clear in which cell a particular artifact lives, there is most likely a problem with the artifact itself.

As an organization begins accumulating artifacts in the development of an enterprise architecture, it can use the Zachman grid to clarify the focus of each of these artifacts. For example, artifacts relating to a service-oriented architecture live mostly in the third row (designer's perspective). They generally will not be of interest to the business owner.

The second suggestion of the Zachman taxonomy is that an architecture can be considered a *complete* architecture only when every cell in that architecture is complete. A cell is complete when it contains sufficient artifacts to fully define the system for one specific player looking at one specific descriptive focus.

When every cell is populated with appropriate artifacts, there is a sufficient amount of detail to fully describe the system from the perspective of every player (what we might today call a *stakeholder*) looking at the system from every possible angle (descriptive focus). So an organization can use the Zachman grid to ensure that appropriate discussions are occurring between all the important stakeholders of an enterprise architecture.

The third suggestion of the Zachman grid is that cells in columns should be related to each other. Consider, for example, the data column (the first column) of the Zachman grid. From the business owner's perspective, *data* is information about the business. From the database administrator's perspective, data is rows and columns in the database.

Enterprise Architecture - A Framework™

	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*	
SCOPE (CONTEXTUAL) *Planner*	List of Things Important to the Business ENTITY = Class of Business Thing	List of Processes the Business Performs Process = Class of Business Process	List of Locations in which the Business Operates Node = Major Business Location	List of Organizations Important to the Business People = Major Organization Unit	List of Events/Cycles Significant to the Business Time = Major Business Event/Cycle	List of Business Goals/Strategies Ends/Means = Major Business Goal/Strategy	SCOPE (CONTEXTUAL) *Planner*
BUSINESS MODEL (CONCEPTUAL) *Owner*	e.g. Semantic Model Ent = Business Entity Reln = Business Relationship	e.g. Business Process Model Proc. = Business Process I/O = Business Resources	e.g. Business Logistics System Node = Business Location Link = Business Linkage	e.g. Work Flow Model People = Organization Unit Work = Work Product	e.g. Master Schedule Time = Business Event Cycle = Business Cycle	e.g. Business Plan End = Business Objective Means = Business Strategy	BUSINESS MODEL (CONCEPTUAL) *Owner*
SYSTEM MODEL (LOGICAL) *Designer*	e.g. Logical Data Model Ent = Data Entity Reln = Data Relationship	e.g. Application Architecture Proc. = Application Function I/O = User Views	e.g. Distributed System Architecture Node = I/S Function (Processor. Storage. etc) Link = Line Characteristics	e.g. Human Interface Architecture People = Role Work = Deliverable	e.g. Processing Structure Time = System Event Cycle = Processing Cycle	e.g. Business Rule Model End = Structural Assertion Means = Action Assertion	SYSTEM MODEL (LOGICAL) *Designer*
TECHNOLOGY MODEL (PHYSICAL) *Builder*	e.g. Physical Data Model Ent = Segment/Table/etc. Reln = Pointer/Key/etc.	e.g. System Design Proc. = Computer Function I/O = Data Elements/Sets	e.g. Technology Architecture Node = Hardware/Systems Software Link = Line Specifications	e.g. Presentation Architecture People = User Work = Screen Format	e.g. Control Structure Time = Execute Cycle = Component Cycle	e.g. Rule Design End = Condition Means = Action	TECHNOLOGY MODEL (PHYSICAL) *Builder*
DETAILED REPRESEN-TATIONS (OUT-OF-CONTEXT) *Sub-Contractor*	e.g. Data Definition Ent = Field Reln = Address	e.g. Program Proc. = Language Statement I/O = Control Block	e.g. Network Architecture Node = Address Link = Protocol	e.g. Security Architecture People = Identity Work = Job	e.g. Timing Definition Time = Interrupt Cycle = Machine Cycle	e.g. Rule Specification End = Sub-condition Means = Step	DETAILED REPRESEN-TATIONS (OUT-OF-CONTEXT) *Sub-Contractor*
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

FIGURE 1-3 Zachman grid.

Although the business owner thinks about data quite differently than the database administrator, there should be some relationship between these perspectives. Somebody should be able to follow the business requirements and show that the database design is, in fact, being driven by those requirements. If there are business requirements that are not traceable down to the database design, we must ask if the business needs will be met by this architecture. On the other hand, it there are database design elements that do not trace back to business requirements, we might ask if we have included unnecessary design at the database level.

I see five ways that the Zachman grid can help in the development of an enterprise architecture. It can help

- Ensure that every stakeholder's perspective has been considered for every descriptive focal point.
- Improve the architectural artifacts themselves by sharpening each of their focus points to one particular concern for one particular audience.
- Ensure that all the business requirements can be traced to some technical implementation.
- Convince the business side that the technical team isn't planning on building a bunch of useless functionality.
- Convince the technical team that the business folks are including them in their planning.

But Zachman by itself is not a complete enterprise architectural solution. There are too many critical issues that Zachman does not address. For example, Zachman does not give us a step-by-step process for creating a new architecture. Zachman doesn't give us much help in deciding if the future architecture we are creating is the best architecture possible. For that matter, Zachman doesn't even give us an approach to show a need for a future architecture. And, most importantly, from our perspective, although the Zachman grid might help organize the architectural artifacts, it does nothing to address the complexity of the enterprise that we are trying to understand.

The Open Group Architecture Framework

The Open Group Architecture Framework is best known by its acronym, TOGAF. TOGAF is owned by The Open Group[19], which is a consortium including many vendors and customers. TOGAF's view of an enterprise architecture is shown in Figure 1-4.

[19] www.opengroup.org.

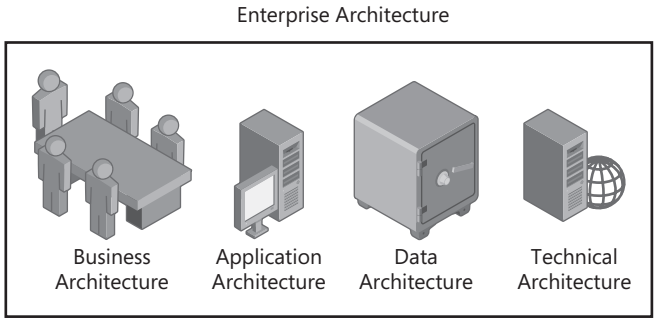


FIGURE 1-4 TOGAF's enterprise architecture.

As shown in this figure, TOGAF divides an enterprise architecture into four categories, as follows:

- **Business architecture** Describes the processes the business uses to meet its goals
- **Application architecture** Describes how specific applications are designed and how they interact with each other
- **Data architecture** Describes how the enterprise data stores are organized and accessed
- **Technical architecture** Describes the hardware and software infrastructure that supports applications and their interactions

TOGAF describes itself as a "framework," but the most important part of TOGAF is the Architecture Development Method, better known as ADM. ADM is a recipe for creating architecture. A recipe can be categorized as a *process*. Given that ADM is the most visible part of TOGAF, I categorize TOGAF overall as an *architectural process*. I thus reject both the description of TOGAF as either an *architectural framework,* as The Open Group describes TOGAF, or a methodology, as The Open Group describes ADM.

Viewed as an architectural *process*, TOGAF complements Zachman, which, you will recall, I categorized as an architectural *taxonomy*. Zachman tells you how to categorize your artifacts. TOGAF gives you a process for creating them.

TOGAF views the world of enterprise architecture as a continuum of architectures, ranging from highly generic to highly specific. It calls this continuum the *Enterprise Continuum*. It views the process of creating a specific enterprise architecture as moving from the generic to the specific. TOGAF's ADM provides the process for driving this movement.

TOGAF calls most generic architectures *Foundation Architectures*. These are architectural principles that can, theoretically, be used by any IT organization in the universe.

TOGAF calls the next level of specificity *Common Systems Architectures*. These are principles that one would expect to see in many—but perhaps not all—types of enterprises.

TOGAF calls the next level of specificity *Industry Architectures*. These are principles that are specific across many enterprises that are part of the same domain—such as, say, pharmaceutical enterprises.

TOGAF calls the most specific level the *Organizational Architectures*. These are the architectures that are specific to a given enterprise.

Figure 1-5 shows the relationship between the Enterprise Continuum and the Architecture Development Method.

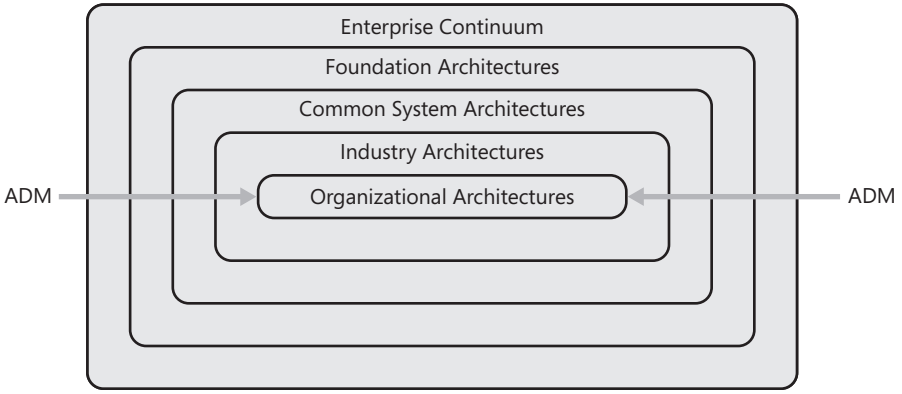


FIGURE 1-5 The TOGAF Enterprise Continuum.

TOGAF defines various knowledge-bases that live in the Foundation Architecture. Two that you might run into are the *Technical Reference Model* (TRM) and the *Standards Information Base* (SIB). The TRM is a suggested description of a generic IT architecture. The SIB is a collection of standards and pseudo-standards that The Open Group recommends that you consider when building an IT architecture.

TOGAF presents both the TRM and the SIB as suggestions; neither is required. In my view, both are biased toward application *portability* at the expense of application *interoperability* and application *autonomy*. I personally consider this an outdated view of technical architectures, but obviously not everybody agrees.

For an enterprise trying to build an enterprise architecture, TOGAF largely boils down to the Architecture Development Method (ADM). Individuals will be exposed to the Enterprise Continuum, the SIB, and the TRM (as well as a few other TOGAF features), which is why I discussed them. But the day-to-day experience of creating an enterprise architecture will be driven by the ADM, a high-level view of which is shown in Figure 1-6.

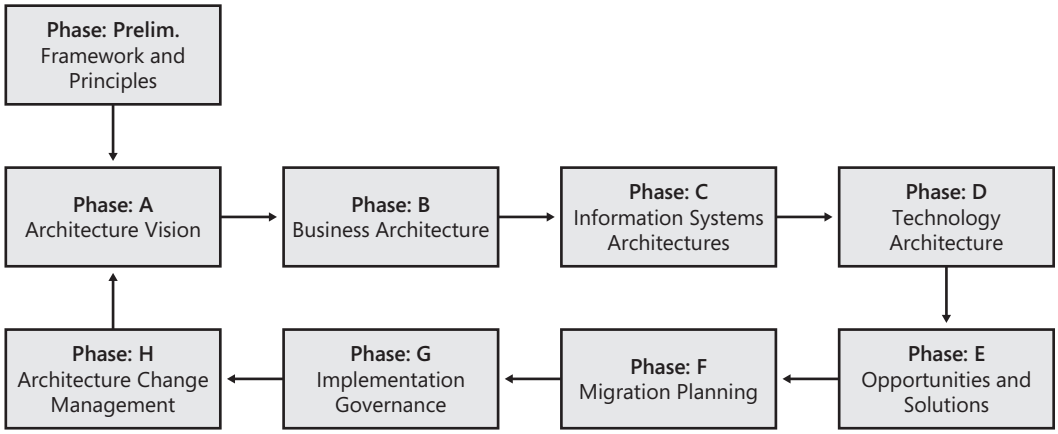


FIGURE 1-6 The TOGAF Architecture Development Method (ADM).

As shown in Figure 1-6, the TOGAF ADM consists of eight phases that are cycled through after an initial "priming of the pump."

The Preliminary Phase typically has three goals:

- To make sure everybody in the organization is comfortable with the process
- To modify the TOGAF process as necessary to fit within the organization's culture
- To set up the governance system that will oversee future architectural work at the organization

In some organizations, achieving buy-in on the need for an enterprise architecture can be very difficult. This is especially true when the effort is driven from the IT organization, and even more so when there is a history of discord between the business and the technical sides of the organization (an all too common situation).

After we have completed the Preliminary Phase, we start Phase A. Phase A begins, at least in theory, with a *Request for Architecture Work* from some group within the organization. This document includes the business reasons for the request, budget and personnel information, and any constraints that need to be considered.

As soon as the Request for Architecture Work (or some equivalent) has been received, we ensure that the project has the necessary support, define the scope of the project, identify constraints, document the business requirements, and establish high-level definitions for both the baseline (starting) architecture and target (desired) architecture.

These baseline and target definitions will include high-level definitions of all four of the enterprise architecture subarchitectures shown back in Figure 1-4—namely, business, technology, data, and application architectures.

The culmination of Phase A will be a Statement of Architecture Work, which must be approved by the various stakeholders before the next phase of the ADM begins. The output of this phase is to create an architectural vision for the first pass through the ADM cycle. An enterprise architect will guide the organization in choosing the project and validating the project against the architectural principles established in the Preliminary Phase. The enterprise architect will also ensure that the appropriate stakeholders have been identified and their issues have been addressed.

The Architectural Vision created in Phase A will be the main input into Phase B. The goal in Phase B is to create a detailed baseline and target business architecture and perform a full analysis of the gaps between them.

Phase B is quite involved—involving business modeling, highly detailed business analysis, and technical requirements documentation. A successful Phase B requires input from many stakeholders. The major outputs will be a detailed description of the baseline and target business objectives, and gap descriptions (that is, descriptions of how to get from the baseline to the target) of the business architecture.

Phase C does for the information systems architecture what Phase B does for the business architecture. In this phase, the enterprise architect works primarily with the technical team. TOGAF defines nine specific steps, each with multiple sub-steps:

1. Develop baseline data architecture description.
2. Review and validate principles, reference models, viewpoints, and tools.
3. Create architecture models, including logical data models, data management process models, and relationship models that map business functions to CRUD (Create, Read, Update, Delete) data operations.
4. Select data-architecture building blocks.
5. Conduct formal checkpoint reviews of the architecture model and building blocks with stakeholders.
6. Review qualitative criteria (for example, performance, reliability, security, integrity).
7. Complete data architecture.
8. Conduct checkpoint/impact analysis.
9. Perform gap analysis.

The most important deliverable from this phase will be the Target Data and Applications Architecture.

Phase D completes the technical architecture—the infrastructure necessary to support the proposed new architecture.

Phase E evaluates the various implementation possibilities, identifies the major implementation projects that might be undertaken, and evaluates the business opportunity associated with each. The TOGAF standard recommends that our first pass at Phase E "focus on projects that will deliver short-term payoffs and so create an impetus for proceeding with longer-term projects."

This is good advice in any architectural methodology. Therefore, we should be looking for projects that can be completed as cheaply as possible while delivering the highest perceived value. A good starting place to look for such projects is the organizational pain-points that initially convinced the organization to adopt an enterprise architectural-based strategy in the first place.

Phase F is closely related to Phase E. In this phase, we work with the organization's governance body to sort the projects identified in Phase E into priority order, which is determined not only by the cost and benefits (identified in Phase E) but also the risk factors.

In Phase G, we take the prioritized list of projects and create architectural specifications for the implementation projects. These specifications will include acceptance criteria and lists of risks and issues.

The final phase is H. In this phase, we modify the architectural change management process with any new artifacts created in this last iteration and with new information that becomes available.

We are now ready to start the cycle again. One of the goals from the first cycle should be information transfer so that outside consulting services are required less and less as more and more iterations of the cycle are completed.

For the most part, the results of the TOGAF process will be determined as much by the individuals in charge of the enterprise architecture as they will by the TOGAF specification itself. TOGAF is meant to be highly adaptable, and details for the various architectural artifacts is sparse. As one book on TOGAF says:

> *TOGAF is not wholly specific with respect to generated documents; in fact, it provides very little in the way of prescriptive document templates—merely guidelines for inputs and outputs.*[20]

The TOGAF specification is also flexible with respect to the phases. As the specification itself says:

> *One of the tasks before applying the ADM is to review its components for applicability, and then tailor them as appropriate to the circumstances of the individual enterprise. This activity might well produce an "enterprise-specific" ADM.*[21]

---

[20] Guide to Enterprise IT Architecture by Col Perks and Tony Beveridge, Springer, published 2003, ISBN 0-387-95132-6.

[21] TOGAF Version 8.1.1.

TOGAF allows phases to be done incompletely, skipped, combined, reordered, or reshaped to fit the needs of the situation. So it should be no surprise if two different TOGAF consultants end up using two very different processes, even when working with the same organization.

TOGAF is even more flexible about the actual generated architecture. In fact, TOGAF is, to a surprising degree, "architecture agnostic." The final architecture might be good, bad, or indifferent. TOGAF merely describes *how* to generate an enterprise architecture, not necessarily how to generate a *good* enterprise architecture. For this, you are dependent on the experience of your staff, TOGAF consultant, or both. People adopting TOGAF hoping to acquire a magic bullet will be sorely disappointed.

And you might also notice a common trend. As with Zachman, TOGAF has no process that specifically focuses on the control of complexity. Like Zachman, it does not model complexity, attempt to understand what causes complexity, or show how the use of the methodology reduces complexity.

Federal Enterprise Architecture

The Federal Enterprise Architecture (FEA) is the latest attempt by the federal government to unite its myriad agencies and functions under a single common and ubiquitous enterprise architecture (EA). FEA is still in its infancy, as most of the major pieces have been available only since 2006. However, it has a long tradition behind it, and, if nothing else, has many failures from which it has hopefully learned some valuable lessons.

FEA is the most complete of all the methodologies discussed in this chapter. It has both a comprehensive taxonomy, like Zachman, and an architectural process, like TOGAF. FEA can be viewed as either a methodology for creating an enterprise architecture or the result of applying that process to a particular enterprise—namely, the U.S. government. I will be looking at FEA from the methodology perspective. My particular interest here is in how can we apply the FEA methodology to private enterprises.

Most writers describe FEA as simply consisting of five reference models, one each for business, service, components, technical, and data. It is true that FEA has these five references models, but there is much more to FEA than just the reference models. A full treatment of FEA needs to include all of the following:

- A perspective on how enterprise architectures should be viewed (the segment model, that I will describe shortly)
- A set of reference models for describing different perspectives of the enterprise architecture (the five models just mentioned)
- A process for creating an enterprise architecture
- A transitional process for migrating from a pre-EA to a post-EA paradigm

- A taxonomy for cataloging assets that fall in the purview of the enterprise architecture
- An approach to measuring the success of using the enterprise architecture to drive business value

You can see that the FEA is about much more than models. It includes everything necessary to build an enterprise architecture for probably the most complex organization on earth: the U.S. government. As the FEA-Program Management Office (FEAPMO) says, FEA, taken *in toto*, provides

> *… a common language and framework to describe and analyze IT investments, enhance collaboration and ultimately transform the Federal government into a citizen-centered, results-oriented, and market-based organization as set forth in the President's Management Agenda.*[22]

Although it might be a stretch to imagine that anything short of divine intervention could "transform the federal government into a citizen-centered, results-oriented, and market-based organization," there is, at least, hope that some of the FEA methodology could help enterprises deal with the much more mundane problem of aligning business and IT. So, let's take a look at what FEA has to offer.

The FEA perspective on EA is that an enterprise is built of *segments*, an idea first introduced by FEAF[23]. A segment is a major line-of-business functionality, such as human resources. There are two types of segments: *core mission area segments* and *business services segments*.

A *core mission area segment* is one that is central to the mission or purpose of a particular political boundary within the enterprise. For example, in the Health and Human Services (HHS) agency of the federal government, *health* is a core mission area segment.

A *business services segment* is one that is foundational to most, if not all, political organizations. For example, *financial management* is a business services segment that is required by all federal agencies.

Another type of enterprise architecture asset is an *enterprise service*. An enterprise service is a well-defined function that spans political boundaries. An example of an enterprise service is *security management*. Security management is a service that works in a unified manner across the whole swath of the enterprise.

The difference between *enterprise services* and *segments*, especially *business service segments*, is confusing. Both are shared across the entire enterprise. The difference is that business service segments have a scope that encompass only a single political organization. Enterprise services have a scope that encompass the entire enterprise.

---

[22] FEA Consolidated Reference Model Document Version 2.1, December 2006, published by the Federal Enterprise Architecture Program Management Office, Office of Management of Budget.

[23] A Practical Guide to Federal Enterprise Architecture by the CIO Council, Version 1.0, February 2001.

In the federal government, for example, both HHS and the Environmental Protection Agency (EPA) use the business service segment *human resources*. However, the people who are managed by human resources are in a different group for HHS than they are for the EPA.

Both HHS and the EPA also use the enterprise service *security management*. But the security credentials that are managed by the security management service are not specific to either of those agencies. Security credentials are managed effectively only when they are managed at the scope of the enterprise.

Resist the temptation to equate either *segments* or *services* with services as in *service-oriented architectures*. There are two reasons such a comparison would be flawed. First, enterprise services, business-service segments, and core mission-area segments are all much broader in focus than services found in service-oriented architectures. Second, segments are an organizational unit for an *enterprise architecture*, whereas services are an organizational unit for *technical implementations*. As organizational units for an enterprise architecture, their depth includes not just the technical, but also the business and the data architectures.

One final note about segments: although segments function at the political (that is, agency) level, they are defined at the enterprise (that is, government) level. Enterprise services, of course, both function and are defined at the enterprise level.

The fact that segments are defined globally facilitates their reuse across political boundaries. One can map out the usage of segments across the political boundaries of the enterprise and then use that map to seek opportunities for architectural reuse. Figure 1-7, for example, shows a segment map of the federal government from the FEA Practice Guide[24]. As you can see, there are many segments (the vertical columns) that are used in multiple agencies and any or all of these are good candidates for sharing.

The five FEA reference models are all about establishing common languages. The goal here is to facilitate communication, cooperation, and collaboration across political boundaries. According to the FEAPMO:

> *The FEA consists of a set of interrelated "reference models" designed to facilitate cross-agency analysis and the identification of duplicative investments, gaps and opportunities for collaboration within and across agencies. Collectively, the reference models comprise a framework for describing important elements of the FEA in a common and consistent way.*[25]

[24] FEA Practice Guidance, December 2006, published by the Federal Enterprise Architecture Program Management Office, Office of Management of Budget.

[25] FEA Consolidated Reference Model Document Version 2.1, December 2006, published by the Federal Enterprise Architecture Program Management Office, Office of Management of Budget.

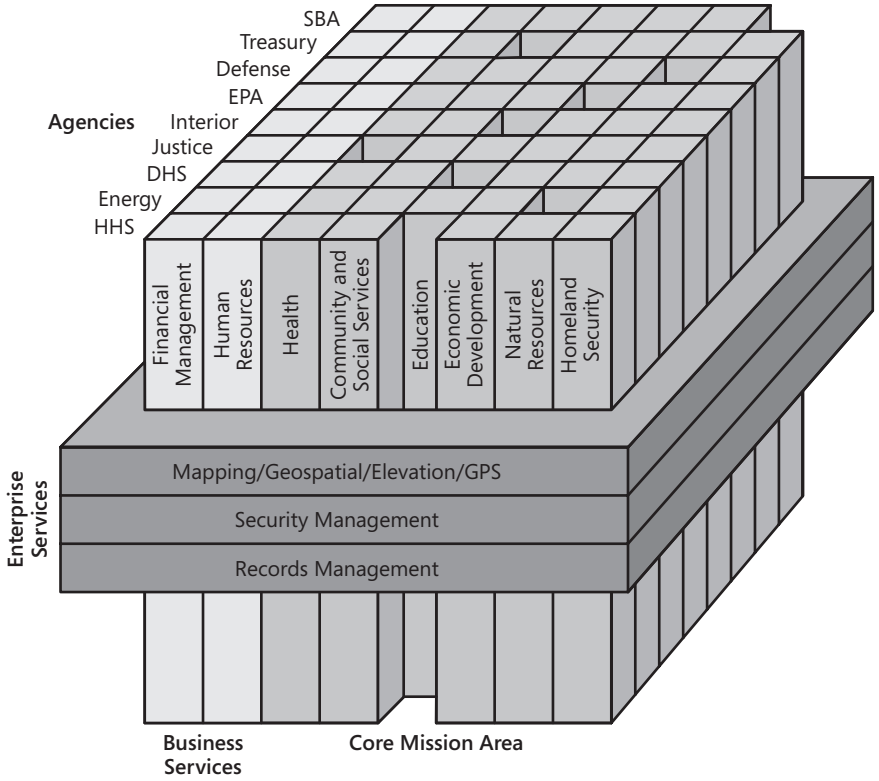


FIGURE 1-7 Segment map of the federal government.

Why do we need a common language? Consider this exchange:

James: Do you have a torch I can borrow?

Roger: No, I'm afraid not.

James: Do you know where I can get one?

Roger: The hardware store in town should have one.

So James goes out to the hardware store and buys himself a torch. He returns.

Roger: Did you get your torch?

James: Yes, here it is.

Roger: That's not a torch! That's a flashlight. Why didn't you say so? I have one you could have borrowed.

James: Well why didn't you say so?

The problem, of course, is that James comes from England where what I call a *flashlight*, they call a *torch*. And when I hear *torch*, I think of a *blowtorch*. Although we both speak English, we don't necessarily speak the same English. The result is that James goes out and unnecessarily spends money on something that I could have lent him.

This is exactly the problem that the FEA Reference Models are trying to solve on a much larger scale. Suppose the Internal Revenue Service (IRS) decides it needs a *demographics* system to track taxpayer data. They ask around to see if anybody has one they can modify for their purposes. Nobody does.

Little do they know that right next door the Government Printing Office (GPO) has a perfectly good demographics system that is almost exactly what the IRS needs. They just happen to call it a *customer analytics* system.

So, the IRS goes out and builds its system from scratch rather than simply modifying the one already built (and paid for) by the GPO. And, in doing so, the IRS will waste considerably more money than James spent on his unnecessary flashlight.

This, in a nutshell, is the goal of the five FEA reference models: to give standard terms and definitions for the domains of enterprise architecture and thereby facilitate collaboration and sharing across the federal government. The five reference models are as follows:

- The *Business Reference Model (BRM)* gives a business view of the various functions of the federal government. For example, the BRM defines a standard business capability called *water resource management* that is a sub-function of *natural resources* that is considered a *line-of-business* of the broader business area *services for citizens*.[26]
- The *Components Reference Model (CRM)* gives a more IT-oriented view of systems that can support business functionality. For example, the CRM defines a *customer analytics* system, which I mentioned earlier in the hypothetical interchange between the Internal Revenue Service and the Government Printing Office.[27]
- The *Technical Reference Model (TRM)* defines the various technologies and standards that can be used in building IT systems. For example, the TRM defines HTTP as a *protocol* that is a subset of a *service transport* that is a subset of *service access and delivery*.[28]
- The *Data Reference Model (DRM)* defines standard ways of describing data. For example, the DRM defines an *entity* as something that *contains attributes* and *participates in relationships*.[29]

[26] ibid.

[27] ibid.

[28] ibid.

[29] The Data Reference Model, Version 2.0, November 2005, published by the Federal Enterprise Architecture Program Management Office, Office of Management of Budget.

- The *Performance Reference Model (PRM)* defines standard ways of describing the value delivered by enterprise architectures. For example, the PRM describes *quality* as a technology measurement area that is defined as "the extent to which technology satisfies functionality or capability requirements."[30]

The FEA process is primarily focused on creating a segment architecture for a subset of the overall enterprise (in FEA's case, the enterprise is the federal government and the subset is a governmental agency) and is described in the FEA Practice Guidance[31]. I discussed the FEA vision on enterprise segments earlier. The overall segment-architecture development process is (at a very high level) as follows:

- **Step 1: Architectural Analysis** Define a simple and concise vision for the segment, and relate it back to the organizational plan.
- **Step 2: Architectural Definition** Define the desired architectural state of the segment, document the performance goals, consider design alternatives, and develop an enterprise architecture for the segment, including business, data, services, and technology architectures.
- **Step 3: Investment and Funding Strategy** Consider how the project will be funded.
- **Step 4: Program Management Plan and Execution of Projects** Create a plan for managing and executing the project, including milestones and performance measures that will asses project success.

The FEA framework for measuring organizational success in using enterprise architecture is defined in the Federal Enterprise Architecture Program EA Assessment Framework 2.1[32]. Federal agencies are rated as to their overall maturity levels in three main categories:

- **Architectural completion** Maturity level of the architecture itself
- **Architectural use** How effectively the agency uses its architecture to drive decision-making
- **Architectural results** The benefits being realized by the use of the architecture

The Office of Management and Budget (OMB) assigns each agency a success rating, based on its scores in each category and a cumulative score, as follows:

- **Green** The agency rates quite well in the *completion* area. (It has a quite mature enterprise architecture.) It also rates well in both the *use* area (that is, it is effectively using that enterprise architecture to drive ongoing strategy) and the *results* area (that is, the usage of that architecture is driving business value).

---

[30] FEA Consolidated Reference Model Document Version 2.1, December 2006, published by the Federal Enterprise Architecture Program Management Office, Office of Management of Budget.

[31] FEA Practice Guidance, December 2006, published by the Federal Enterprise Architecture Program Management Office, Office of Management of Budget.

[32] Federal Enterprise Architecture Program EA Assessment Framework 2.1, Dec 2006.

- **Yellow** The agency rates quite well in the *completion* area. It also rates well in either the *use* area or the *results* area.
- **Red** The agency does not have a completed architecture, is not effectively using that architecture, or both.

The framework is interesting beyond the confines of the public sector. The category ratings can be fruitfully adapted by many enterprises to assess the maturity level of their own architectural efforts. Figure 1-8, for example, shows my own interpretation of the OMB maturity rankings for *architectural completion* as I adapt them for the private sector. Similar adaptations can be created for *architectural usage* and *architectural results*.

This completes the discussion of FEA. As you can see, FEA includes quite a bit of methodology.

The one thing that FEA does not include is a methodology that specifically addresses how one manages complexity. In this one regard, FEA is just like Zachman and TOGAF. And it is in the failure to control complexity that one can find the root cause of so many enterprise architecture failures of the U.S. government.

Category: Architectural Completion

Description: This category measures the architectural maturity of an enterprise's architecture in terms of performance, business, data, service, and technology. This includes an assessment of the architectural artifacts and both the baseline (existing) and target (goal) architectures.

Level	Name	Description
1	Initial	The enterprise is using informal and ad-hoc EA processes. Some architectural artifacts for a given architectural level may exist, but the levels are not linked or the linkage is incomplete.
2	Baseline	The enterprise has developed a baseline (as-is) architecture. The architecture has enterprisewide scope, and the linkages between levels are well established and clearly articulated.
3	Target	The enterprise has developed both a baseline architecture (as described above) and a target (goal) architecture. The target architecture is aligned to enterprisewide goals and organizational responsibilities. The target architecture addresses the priorities and performance objectives identified in the enterprise business plan.
4	Integrated	The enterprise has developed at least vertically partitioned architecture that has been approved by the business owner in writing. The relevant organization(s) within the enterprise are actively migrating toward the relevant architecture.
5	Optimized	The enterprise has developed multiple vertically partitioned architectures that support core mission business functions, all approved by the appropriate business owners.

FIGURE 1-8 OMB Ranking of Architectural Completion, adapted for the private sector by Roger Sessions.

Summary

I have given you an overview of some of the problems that drive organizations to consider enterprise architectures and described the commonly used enterprise architectural methodologies—the three most popular being Zachman, TOGAF, and FEA.

The original goal of enterprise architecture was to address the growing rift between technological capability and business need in an environment in which both were becoming increasingly complex. And while all of the existing enterprise architecture methodologies claim to help address complexity, none of the existing methodologies do so in any meaningful way.

It's easy to understand why none of these methodologies attempt to address complexity. The problem of complexity is, well, complex. But that is, indeed, the problem we need to understand if we are to leverage enterprise architectures in any meaningful way. So the problem of complexity is where we go next.

Chapter 6

A Case Study in Complexity

Let's take a look at a real-life case study of a complex system. There are three important lessons to be learned here. The first is how complexity creeps into a project, even one with the benefit of extensive planning. The second is how this unchecked complexity leads to project failure, even one with seemingly unlimited resources. The third is how Simple Iterative Partitions (SIP) might have saved this project, even when it was well into failure mode.

The case study I'll discuss is one of the largest and most complex systems yet tackled by any government organization. It is the National Program for Information Technology (NPfIT), a program run by the British Government's National Health Service (NHS). Sometimes NPfIT is referred to simply as the National Program, or, as they say in Britain, the National Programme. Remember these acronyms—NPfIT and NHS—you will be seeing a lot of them in this chapter.

Overview of NPfIT

NPfIT was launched in June 2002. The basic goal of NPfIT was, and continues to be, to automate and centralize the massive recordkeeping that is the backbone of its national health care system run by the NHS. Health care in Britain is mostly nationalized, unlike the United States where health care is mostly ad hoc. This centralized system creates a unique opportunity to standardize the recordkeeping of a very large number of patients and health care providers. NPfIT is promising the following capabilities when completed:

- Automation of all patient care information.
- Access to any patient record by any authorized health care professional in the UK
- Ability for primary health care staff to book appointments for patients with any other health care worker in any health care facility in the UK
- Automation of prescription services

The NHS describes NPfIT systems as follows:

> *A key aim of the National Program [NPfIT] is to give healthcare professionals access to patient information safely, securely and easily, whenever and wherever it is needed. The National Program is an essential element in delivering The NHS Plan. It is creating a multi-billion pound infrastructure which will improve patient*

care by enabling clinicians and other NHS staff to increase their efficiency and effectiveness.[1]

In a nutshell, the NPfIT promises an integrated system connecting every patient, physician, laboratory, pharmacy, and health care facility in the UK. NPfIT functionality can be loosely divided into three main categories: regional clinical information systems (CIS), infrastructure systems, and shared applications. The NPfIT architecture is shown in Figure 6-1.

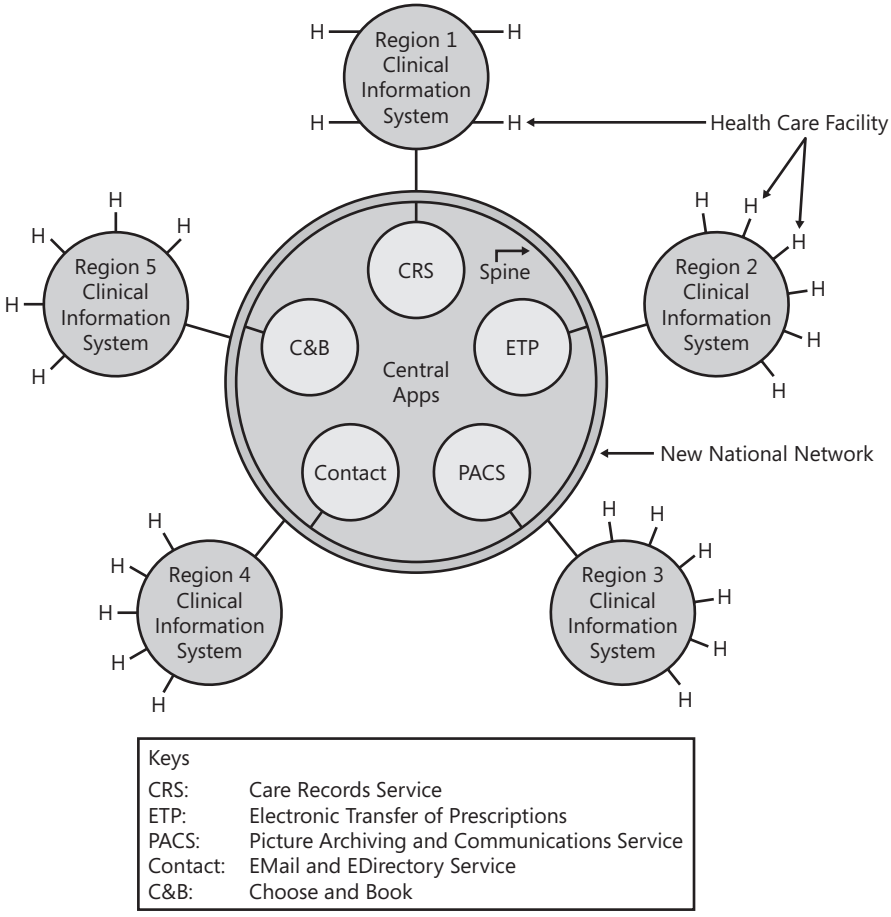


FIGURE 6-1 NPfIT architecture.

Regional clinical information systems connect health care providers (for example, hospitals, clinics, and physician offices) within a geographic area and provide their main point of contact for NPfIT. These are shown as the "hairy spheres" hanging off the central sphere shown in Figure 6-1.

[1] "The National Program for IT Implementation Guide," December 2006.

By my estimates, the regional clinical information systems account for approximately 79 percent of the total initial budget for NPfIT, approximately $9.8 billion. Keep in mind that NPfIT expenses are given in British pounds, and I have converted these numbers to U.S. dollars. These costing numbers are based on conflicting source data and a fluctuating exchange rate, so take these estimates as educated guesses.

Infrastructure systems will provide connectivity, security, and directory services to the NPfIT. These infrastructure systems include the New National Network (N3), which provides the network facilities, and the spine, which includes shared software facilities such as directory services. Care Records Service (CRS), the shared patient records, is sometimes shown as part of the spine and sometimes as separate shared applications.

After the regional clinical information systems, the infrastructure is the second largest part of the NPfIT budget, accounting for approximately 18 percent of the initial NPfIT budget, or $2.3 billion.

Shared coordinated activity across the entire NPfIT system appears to make up a relatively small part of the overall NPfIT budget, less than 5 percent, or about $300 million. The most important of these shared applications include the following:

- **Choose and Book** A system that allows an appointment to be booked for any patient at any facility in the system
- **Electronic Transfer of Prescriptions** A system that allows prescriptions to be entered for any patient in the system and filled by any pharmacy in the system
- **Picture Archiving and Communications Service** A system that allows the central storage and retrieval of picture data, especially x-rays

The amount of data that must be coordinated is immense. According to the NHS[2], in a typical week the NHS processes

- Six million patient visits to general practitioners
- Over 64,500 emergency calls by NHS ambulances
- 360,000 patient x-rays
- 13.7 million NHS prescriptions

The NHS estimates that NPfIt will need to coordinate about 3 million critical processes and 30 million transactions per day.

The NPfIT geography is split into five *clusters*, or regional groups of patients and health care providers. The NPfIT budget is almost $2 billion per cluster. The clusters are arranged as follows:

- North East (which includes Tees Valley, Northumberland, South Yorkshire, West Yorkshire)

[2] "The National Program for IT Implementation Guide," December 2006.

- North West and West Midland (which includes Greater Manchester, Cheshire)
- Eastern (which includes Essex, Trent)
- London
- Southern (which includes Avon, Dorset, Thames Valley)

The initial budget was allocated in 2004 among many different vendors. The highly lucrative regional cluster contracts each had a primary vendor, a CIS vendor, and other miscellaneous secondary vendors. The primary and CIS vendors awarded to each region is shown in Figure 6-2.

Regional Cluster					
	North East	North West and West Midland	Eastern	London	Southern
Primary Vendor	Accenture	CSC	Accenture	BT	Fujitsu
CIS Vendor	iSoft	iSoft	iSoft	IDX	IDX

FIGURE 6-2 Primary and CIS vendors by regional cluster.

So NPfIT is a multibillion-dollar project split between at least a dozen vendors spread over a geographic territory of close to 100,000 square miles; it offers services to 60 million people and is expected to process over 300 transactions per second. I would call this project highly complex.

Now that you have a basic overview of NPfIT, let's see how well this project did using traditional architectural approaches. Perhaps you will recognize ghosts of your own projects in this description.

Current Status of NPfIT

NPfIT has been in crisis almost from the first day. By mid-2004 (barely a year into the contract), both Fujitsu and BT, two of the five primary regional vendors, were having trouble with their IDX (regional CIS vendor) relationships, and this trouble never let up. According to a confidential draft audit by the National Audit Office (NAO, a British government audit office),

> *By mid-2004 NHS Connecting for Health was concerned about the effectiveness of supplier management of both BT and Fujitsu, and the performance of IDX… However, by April 2005, even though NHS Connecting for Health [the British government bureau responsible for NPfIT] had been applying increasing pressure, working with the prime contractors, to encourage IDX to match its planned deliveries, insufficient progress had been demonstrated and Fujitsu lost confidence in IDX's ability to deliver the Common Solution project.*

The CSC/iSOFT partnership was faring little better. According to that same confidential NAO audit,

> *CSC, the Local Service Provider for the North West Cluster, agreed to a remediation plan with NHS Connecting for Health for the delivery of Phase 1 Release 1 as it was having problems meeting the original target dates... Further delays led to a second remediation plan which pushed the deployment dates for two elements of Phase 1 Release 1 further back into 2006, some 19 to 22 months later than originally planned.*

But of all the partnerships, the one that probably fared the worst was the Accenture/iSOFT partnership. By September 2006, Accenture had decided that the pain associated with this project was not worth it, and abandoned the project altogether. According to a baseline case study[3] in so doing it walked away from almost $4 billion in revenue writing off $500 million it had already spent, and agreeing to pay $100 million "to settle its legal obligations."

iSOFT was involved in three of the four partnerships, and the strain on that company might bankrupt it or, at the very least, force its sale. According to its financial results released in December 2006, the company took an almost $800-million loss for the fiscal year ending in April 2006—a huge loss for a company that had total revenues of the year of only $340 million.[4]

As you can see, every major company involved in the regional clusters has taken a severe financial hit from NPfIT. It seemed that everybody underestimated the complexity of this project. The costs of this underestimation will likely be measured in the tens of billions of dollars.

In the area of user confidence, NPfIT is in serious trouble. There are three critical constituencies that have been alienated by NHS's approach to NPfIT: health care workers, patients, and IT professionals.

A good indication of how the health care professionals feel about NPfIT is found in a recent editorial of the British Journal of General Practice (May 2005). It says,

> *The impact on patients and professionals has yet to be seriously addressed. A very different approach is needed to nurture culture change... The £30 billion question is not just whether NPfIT will get the technology right but whether it can also win the hearts and minds of the people on whom the NHS depends every day.*

Patients are also unhappy about NPfIT, even at this early stage of the project. Most of the patient concerns are directed at the ability of NPfIT to protect records. This distrust is illustrated by a Web site, *http://www.TheBigOptOut.org*, which states,

---

[3] "U.K. Dept. of Health: Prescription for Disaster," November 13, 2006, by Laton McCartney.

[4] iSOFT Group Interim results for the six months ended October 31, 2006.

> *This system is designed to be a huge national database of patient medical records and personal information (sometimes referred to as the NHS 'spine') with no opt-out mechanism for patients at all. It is being rolled out during 2007, and is objectionable for many of the same reasons as the government's proposed ID database... You will no longer be able to attend any Sexual Health or GUM (Genito-Urinary Medicine) Clinic anonymously as all these details will also be held on this national database, alongside your medical records. For the first time everyone's most up-to-date and confidential details are to be held on one massive database.*

But of all the constituent groups that have expressed unhappiness with NPfIT, the most vocal by far has been the IT community. In January 2005, The British Computer Society (BCS) sent a position paper to the NAO describing a number of concerns with the NPfIT approach, including the following:

- Failure to communicate with health care users
- Monolithic approach
- Stifling of innovation among the health informatics market
- Lack of record confidentiality
- Quality of the shared data

In April 2006, 23 highly respected academicians sent an open letter to the Health Select Committee. In this letter, they made some harsh statements:

> *Concrete, objective information about NPfIT's progress is not available to external observers. Reliable sources within NPfIT have raised concerns about the technology itself. The National Audit Office report about NPfIT is delayed until this summer, at earliest; the report is not expected to address major technical issues. As computer scientists, engineers and informaticians, we question the wisdom of continuing NPfIT without an independent assessment of its basic technical viability.*

In October 2006, this same group sent another open letter to the same committee:

> *Since then [April] a steady stream of reports have increased our alarm about NPfIT. We support Connecting for Health in their commitment to ensure that the NHS has cost-effective, modern IT systems, and we strongly believe that an independent and constructive technical review in the form that we proposed is an essential step in helping the project to succeed... we believe that there is a compelling case for your committee to conduct an immediate Inquiry: to establish the scale of the risks facing NPfIT; to initiate the technical review; and to identify appropriate shorter-term measures to protect the program's objectives.*

The BCS offered to help the NHS with a review of the NPfIT architecture. What did NHS think of this generous offer? Not much. Lord Warner, head of the NHS responded forcefully:

> *"I do not support the call by 23 academics to the House of Commons Health Select Committee to commission a review of the NPfIT's technical architecture. I want the Program's management and suppliers to concentrate on implementation, and not be diverted by attending to another review."*[5]

Soon after, Lord Warner apparently had had enough. Like Accenture, he was bailing out. In December 2006, he announced his retirement from the NHS. He was followed in July 2007 by Richard Granger, Director General of IT for NHS, the man who was widely blamed for most, if not all, of NPfIT's problems.

At this point, nobody knows what the eventual cost for NPfIT will be. Estimates range from $48 billion to $100 billion. It seems likely that the project will go down in history as the world's most expensive IT failure.

The SIP Approach

Clearly, NPfIT is a very expensive project in very deep trouble. But could SIP have helped? Let's look at how the SIP process would have likely played out with NPfIT.

Let's start in Phase 1. The first deliverable of Phase 1 is an audit of organizational readiness. Such an audit would have revealed deep distrust between the NHS IT organization and the business units (health care providers). This would have been an immediate sign of concern.

Also in Phase 1 we would have delivered extensive training in the nature of complexity. We would have spent considerable time discussing how important it was that complexity, especially on such a massive undertaking as NPfIT, be managed as the absolute highest priority.

In Phase 2, we would have been working on the partitioning. In the case of NPfIT, considerable effort had already been done on partitioning; Figure 6-1 could be viewed as an ABC diagram of NPfIT. The question is, does that diagram represent good partitioning? Is it even a partitioning (in the mathematical sense) at all?

Figure 6-1 does not give us enough information to answer this question. We need to understand not only how the organization is being decomposed into sets of functionality, but what the type relationships are between those sets.

So let's tackle this. Figure 6-3 shows an ABC diagram of the clinical information part of NPfIT (the part that owns 80 percent of the NPfIT budget), focusing on types, implementations, and deployments. Compare this figure to Figure 6-1.

---

5 BJHC.CO.UK, November, 2006.

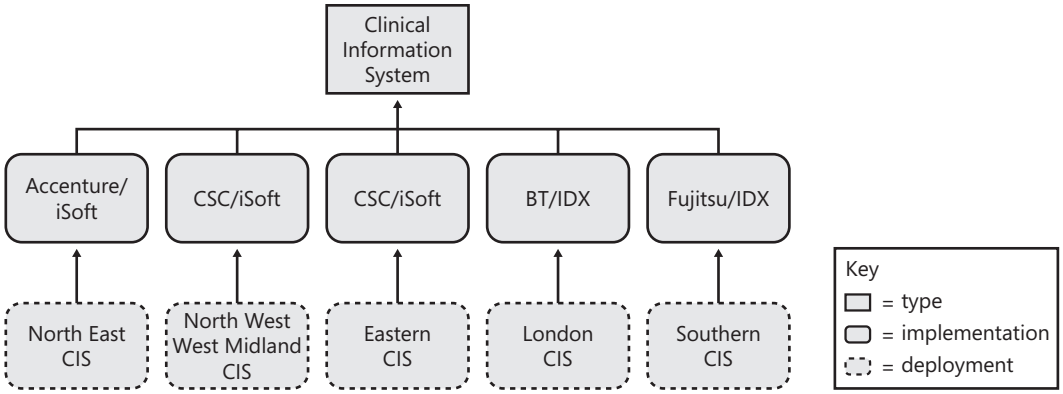


FIGURE 6-3 ABC diagram of NPfIT regional CIS.

In Figure 6-3, the central problem of NPfIT jumps out like a sore thumb (a $10 billion sore thumb). They implemented the various regional clinical information systems as siblings (in SIP talk) rather than as clones. In other words, they created five different implementations of the same system. The same very complex system.

Interestingly, NHS did this on purpose. Now why, you might ask, would anybody take a highly complex system that they would be lucky to implement properly once and tempt the fates with five completely different implementations created by five completely different vendors?

The reason NHS gave for the multiple implementations was that it didn't want to be dependent on any one vendor. This example illustrates a common reason that so many projects become so complex so quickly: poor communication between the business and IT units.

Somebody in the business group decides on some business requirement—say, X. In this case, X can be stated as, "There must be no dependency on any one vendor for the regional CIS portion of NPfIT." X gets passed around. It sounds reasonable. Who wants to be dependent on one vendor? X is accepted as a business requirement. It drives a series of technical requirements. In this case, the technical requirement is that there must five independent implementations of the regional CIS.

Everything seems reasonable. A reasonable business requirement driving the necessary technical requirements. So what would have been done differently using SIP?

A SIP process would have encouraged this business requirement to have been measured against the complexity it would introduce. Complexity, in the SIP world, trumps almost everything. The diagram in Figure 6-3 would have been a warning sign that we have a huge amount of unnecessary complexity. Because both the business and technical folks would have already been through the SIP training, they would understand the frightening implications of complexity. On a project of this scope, the project motto should be, "Our Top Three Concerns: Complexity, Complexity, Complexity."

Given a common conditioned response to complexity, it would have been easy to discuss the importance of this particular business requirement relative to its cost. We would have asked some pointed questions. Is it really necessary to be vendor independent? Is the multibillion dollar cost worth vendor independence? Is meeting this requirement going to put the project at more risk than if we dropped this requirement? Is it even possible to be vendor independent? Are multiple implementations the only way to achieve vendor independence? Would parallel implementations, with one chosen in a final shootout, be a better approach to achieving vendor independence?

I don't know which solution would have been chosen in a SIP approach. But I know one solution that would *not* have been chosen: five independent implementations of the same type. This is an extreme case of an unpartitioned architecture. And an unpartitioned architecture, in a SIP analysis, is unacceptable. It is not unacceptable because one person or another doesn't like the diagrams it produces. It is unacceptable because it fails to satisfy the mathematical models that predict whether or not the architecture can be successful.

So by the end of Phase 2, we would have dropped four of the five proposed implementations for regional clinical information systems. Expected complexity reduction: 80 percent.

But we aren't done yet. Next we enter Phase 3, the phase in which we simplify our partition. I'll continue my focus on the regional CIS portion of NPfIT.

Of course, we have already done quite a bit to simplify the regional CIS portion. We have eliminated 80 percent of the work, but we are still left with a highly complex system. What is the best way to simplify a highly complex system? If you have been following the SIP discussion, the answer should be obvious: partitioning. The most effective way to tame the regional CIS monster is to partition it into four or five subsets, each with synergistic functionality, and each with functionality that is autonomous with respect to the functionality in the other subsets.

One possible partition of subsets might include, for example, patient registration, appointment booking, prescriptions, patient records, and lab and radiology tests.

To explain this in SIP terminology, we have taken an autonomous business capability (ABC) that includes the regional CIS and decomposed it into five lower level ABCs. Figure 6-4 shows the regional CIS ABC before and after this process.

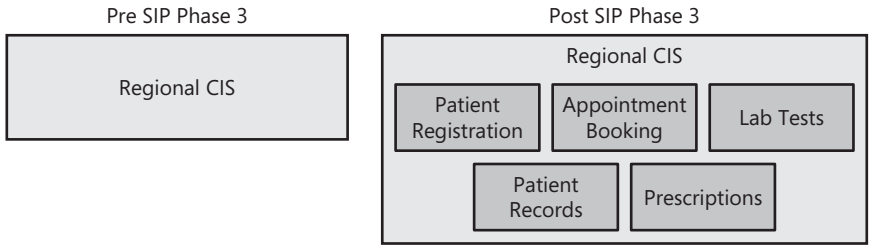


FIGURE 6-4 Decomposition of regional CIS.

At this point, we check our post-SIP analysis against the five Laws of Partitions. The First Law says that all the original functionality of the regional CIS must end up in one and only one of the subsets. The Second Law says that the subsets must make sense from an organizational perspective. The Third Law says that there should be a reasonable number of subsets in the partition. The Fourth Law says that subsets must be roughly equal in size and stature. The Fifth Law says that subset interactions must be minimal and well regulated. The first four laws can be checked relatively easily. The fifth law needs to be revisited after we have more details about the technical architecture.

The partitioning of the regional CIS ABC will likely result in a huge further reduction in complexity. How much? The mathematical models predict possible reductions of more than 99.99 percent. These are based on theoretical numbers, not real-world numbers, but as I discussed in Chapter 3, "Mathematics of Complexity," 90-percent reductions in the real world are likely. And remember, we have already removed 80 percent of the complexity, so now we are removing 90 percent of the 20 percent that is left. This means that realistically we are now down to perhaps 2 percent of the complexity with which we started.

And there is yet more we can do to reduce complexity. We can look at reducing both the functionality footprint (the amount of functionality in the final system) and the implementation footprint (the impact on the IT staff).

Reducing the functionality footprint means re-examining all the business and technical requirements and confirming that, first of all, every business requirement is absolutely necessary, and second of all, that every technical requirement can be traced back to a business requirement. Remember that we have already found one business requirement (vendor dependence) that is either unnecessary or highly suspect.

Reducing the implementation footprint means looking for opportunities to consolidate or outsource subsets. The type information we have generated on the ABCs will be a great help in our efforts to reduce the implementation footprint.

The next phase is Phase 4, in which we prioritize the subsets making up the partition. Again, I will focus on the regional CIS portion of NPfIT.

In Phase 3, we identified five subsets of the regional CIS that together form a partition:

- Patient Registration
- Appointment Booking
- Prescriptions
- Patient Records
- Lab Tests

In the actual NHS plan, this functionality was delivered en masse. In the SIP approach, we want to deliver this functionality iteratively. In Phase 4, we decide on the order of iteration.

Iteration order should be based on risk, cost, and benefit. The basic rule of thumb is to go for the low-hanging fruit first. In the SIP world, low-hanging fruit is defined as ABCs that are highly visible, low cost, and low risk. These requirements are sometimes at odds with each other (although, in my experience, less often than people think). The best way to sort this out is with the Value Graph Analysis that I described in Chapter 5, "SIP Process." If we were using Value Graph Analysis in this project, we would have standardized the analysis back in Phase 1 of the project.

What usually makes an ABC "high visibility" is its association with organizational pain points. Let's say, for example, that NHS was notorious for the length of time it took to book appointments. This factor would tend to move *Appointment Booking* ahead in the priority list. *Lab Tests*, on the other hand, might be something that is already handled reasonably well. Lab Tests might still be worth doing, say, because it can reduce the cost of processing lab tests, but without high visibility, it doesn't rate as a high priority.

Let's say that at the end of Phase 4 we have decided on the following order of iterations:

1. Appointment Booking
2. Patient Registration
3. Prescriptions
4. Patient Records
5. Lab Tests

Next is Phase 5, the iterative phase. As I have said, Phase 5 is the one in which we have the fewest opinions, other than that the candidate ABCs be implemented in an iterative fashion and that the order follow the priority laid out in Phase 4. The implementation of an ABC is effectively a solution architecture and implementation issue, and I'm assuming that an organization already has processes in place to create and implement a solution architecture. You might, for example, use The Open Group Architectural Framework (TOGAF), with its emphasis on process and current and future architectures. You might use some of the Federal Enterprise Architecture (FEA) characterizations of functionality given in its Reference Models. You might use Zachman's taxonomy to ensure that you are considering each important perspective on each important capability of the system. You might use IBM's Unified Process or the Microsoft Solution Framework to guide the implementation process. These are outside of the scope of SIP.

But the iterative approach is not outside the scope of SIP. I believe that the first ABC should be rolled out, tested, approved, deployed, and embraced before the next one is started. Such an approach allows you to learn your lessons as cheaply as possible and apply them as broadly as possible. It also helps you build enthusiasm for the overall project. Nothing succeeds, as they say, like success. Success attracts success. Let's see how such an approach might have benefited NPfIT. We could look at any number of issues plaguing NPfIT. Let's consider one that I haven't discussed yet: risk management.

LORENZO was an existing product developed by iSOFT, and NHS was impressed with LORENZO's user-friendly screens and broad CIS functionality. For this reason, NHS encouraged its use as a core component for all of its regional CIS systems.

Accenture seemed similarly impressed with LORENZO. In June 2004, Accenture/iSOFT released a joint press release saying,

> *A set of information processing tools promoting governance, quality, efficiency, and consent in healthcare, LORENZO facilitates the free flow of information among the entire healthcare community, including general practitioners, hospitals and patients. As Accenture deploys LORENZO across the two regions, the software's unified architecture will form the basis of solutions tailored to meet local requirements and information needs of healthcare professionals.*[6]

But there was a hidden time bomb in LORENZO. This time bomb can be summed up in two words: client/server.

According to a performance audit of LORENZO conducted in April 2006 by Health Industry Insights and commissioned by iSOFT, the LORENZO architecture as it existed in 2004 was "based on a fat client/server model."[7] Accenture was either blissfully unaware of the fact that LORENZO was a client/server system or was ignorant of the issues one faces with client/server architectures.

What is the problem with client/server models? The client/server architecture is based on a two-machine configuration. One machine (the "client") contains the user-interface code and the business logic. The other machine (the "server") contains the code that manages data in the database.

The two machines are "connected" by database connections. A database connection is created when a client machine requests access rights to the database owned by the server. The database looks at the credentials of the requesting machine, and, if it is satisfied, creates a database connection. A database connection is technically a block of data that the client presents to the server machine when making data access requests. When the client machine is ready to shut down for the day, it releases its database connection by letting the server machine know that it will no longer require the services of the database.

There are several reasons that client/server architectures are so popular. For one, they are very fast. They are fast because the client machine requests the database connection (a highly expensive request) only once, in the beginning of the day, when the client machine is first started.

---

6 "iSOFT Delivers LORENZO for Deployment," Accenture, June 16, 2004.

7 "Coming of Age: A Report on a Performance Benchmark Test of iSOFT's LORENZO Clinical Information System," by Marc Holland and Luisa Bordoni, April 2006.

Client/server systems are also easy to implement because the code that presents the data (the "user interface logic") is located in the same process as the code that manipulates the data (the "business logic"). This makes it easy to mingle the presentation logic and the business logic, with the result of lightning-fast data presentation and manipulation.

So back to my original question. What is wrong with a client/server architecture? Actually, there is only one problem with client/server systems. They do not scale. Although they work great for small numbers of users (measured, say, in the dozens), they do not work at all well for large numbers of users (measured, say, in the thousands). And the user requirements of NPfIT were measured in the tens of thousands.

The reason client/server architectures do not scale well is that each client machine requires a dedicated database connection. Databases are limited in the number of database connections they can support. When each client requires a dedicated database connection, the number of client machines is limited by the number of database connections supported by the database. And because client machines are in a one-to-one relationship to users, this limits the number of users who can use the system at any one time.

So a client/server architecture, with its extreme limitation on numbers of clients, is a problem for NPfIT. A big problem.

To address the scalability limitations of client/server architectures, a new style of technical architecture was developed, initially, in the 1970s, and was quite mature by the mid-1980s. This new style of technical architecture is known as three-tier.

In a three-tier architecture, one machine runs the database, as it had in the client/server architecture. But now the user-interface logic and the business logic are separated. The user-interface logic lives on the machine before which the human being sits. But the business logic lives on another machine. This machine is often referred to as the "middle tier" because it conceptually lives in between the user interface machine and the database machine.

It is the middle tier machine that owns the database connections. This arrangement allows a pooling of those very expensive database connections so that when a database connection is not being used by one client, it can be used by another.

So the obvious issue that iSOFT faced with its LORENZO product, back in 2004, was how to take a product based on a fundamentally nonscalable architecture and turn it into a scalable system. There is really only one answer to this problem. The company had to rearchitect LORENZO from a client/server architecture to some variation of a three-tier architecture.

This, according to that previously quoted audit, is exactly what iSOFT did. In fact, the company decided that it would go one better. It would bypass the three-tier architecture and move directly to an even more advanced architecture known as service-oriented architecture (SOA). An SOA is essentially an architecture in which the middle tier has been split further apart, with business functionality distributed over a number of middle-tier-like machines, each using industry-standard service-oriented messages as a communications protocol.

As the audit stated,

> *this new [LORENZO] architecture... utilizes a service oriented architecture (SOA) ... making iSOFT the first major CIS vendor worldwide to base its overall architecture principally on SOA. This architecture will serve as the foundation for the entire line of LORENZO solutions, allowing different subsets or combinations of existing and planned functional capabilities to be delivered on a common technical platform. For both iSOFT and its clients, this strategy will facilitate the ability to cost-effectively configure and scale CIS applications to meet a wide range of organizational models and functional demands...because the client machine is almost entirely focused on working with the human client.*

Although this transformation from client/server to SOA was absolutely necessary from a scalability perspective, it was also something else: highly risky.

Many organizations have "ported" three-tier architectures to SOAs. This process is usually straightforward because the two architectures are so similar. However, LORENZO, remember, was not a three-tier architecture. It was a client/server architecture.

The transformation from client/server to either three-tier or SOA is rarely straightforward. Either process requires massive changes to the underlying programs. All of that nicely intermingled user-interface and business logic needs to be painstakingly located and laboriously separated. More often than not, it is less expensive to re-implement the system from scratch rather than try to make (and debug) the necessary changes. So while LORENZO might have been a wonderful product, it was a product that would have to be rewritten from the ground up to meet the needs of NPfIT. And further, it would need to be rewritten by a group that had no previous experience in either three-tier architectures or SOAs, both of which are highly specialized areas.

There is no way to know if Accenture knew about this high-risk factor back in 2004. It should have. Any reasonably competent architect could have looked at the LORENZO code and recognized the unmistakable fingerprint of a client/server architecture. But there was no indication in its joint press release that this issue was understood or that the risk factor had been addressed.

The indications are that by the time the limitations of LORENZO's architecture were understood, three of the five regional clusters of NPfIT were in serious trouble and Accenture was so deeply over its head that it was ready to jump from the sinking ship.

An iterative approach to delivering the regions would not have made the iSOFT architectural limitations any less real. But it would have made them obvious much earlier in the project. While it might have been too late to save all three regions that had bet on LORENZO, at least two of the regions could have learned from the painful lessons of the first. Billions of dollars would likely have been saved overall.

Iterative delivery is a key strategy in managing high-risk factors. Unfortunately, it is a strategy that was not used by NPfIT.

There is yet another problem facing NPfIT besides risky architectures, and this is low user confidence. Let's see how this played out in NPfIT and how iteration could have helped.

Regardless of how good or bad NPfIT ends up, its ultimate success or failure is in the hands of its users. The support of the hundreds of thousands of health care workers and patients will determine the final judgment of this project. As with most large IT projects, user perception *is* reality. If users think the project is a success, it is a success. If users think the project is a failure, it is a failure, regardless of how much the project owners believe otherwise.

Iterative delivery can be a great help here. If the early deliveries are a failure, their failures are limited in scope and in visibility. If they are a success, the enthusiasm of the initial users becomes contagious. Everybody wants to be the next owner of the new toy!

As I pointed out earlier in this chapter, NPfIT suffers a major credibility gap with health care workers, patients, and the IT community. It seems that nobody other than NHS management believes that this multibillion dollar investment is going to pay off.

Could it have been different? Suppose NHS had chosen the highest visibility ABC from the list of candidates, the *Appointment Booking* ABC. Imagine that NHS had endured years of criticism for difficulties in its current booking procedures and then rolled out this new automated booking system. Suppose it first showed prototypes to the health care professionals. Say they loved the interface but had a few suggestions. NHS then incorporated those suggestions and rolled out the Appointment Booking to one region.

Very quickly booking in that region went from six-month waiting lists to four-day waiting lists. Appointments that used to require hours of standing in line now take a few minutes on a Web browser or on a phone. Other regions would be clamoring to be the next one in line for deployment.

As Appointment Booking was deployed across the UK, the entire health care system would have appeared to have been transformed. Even though only one small part of the overall health care process, appointments, had been affected, that impact would have been felt in a positive way by every constituent group.

As NHS started work on its next ABC, Patient Registration, it would be basking in the success of its previous work. It would be facing a world that supported its efforts, believed its promises, and eagerly awaited its next delivery.

This is the way it could have been had NHS used an iterative delivery model based on SIP. But it didn't. And instead, it faces a world that ridicules its efforts, laughs at its promises, and dreads its next delivery. The world believes that NPfIT will be a failure. In the eyes of the world, failure is all NHS has delivered. Why should the future be any different?

Ironically, even if NPfIT does manage to deliver any successes, it will be hard pressed to prove it. Why? Because at the start of this multibillion dollar project, nobody bothered to document what success would look like in any measurable fashion. Let me show you what I mean.

The NPfIT business plan of 2005[8] gave these success indicators for patients:

- Patients will have a greater opportunity to influence the way they are treated by the NHS.
- Patients will be able to discuss their treatment options and experience a more personalised health service.
- Patients will experience greater convenience and certainty, which will reduce the stress of referral.
- Patients will have a choice of time and place, which will enable them to fit their treatment in with their life, not the other way round.

For health care providers, the business plan promised these benefits:

- General practitioners and their practice staff will have much greater access to their patients' care management plans, ensuring that the correct appointments are made.
- General practitioners and practice staff will see a reduction in the amount of time spent on the paper chase and bureaucracy associated with existing referral processes.
- Consultants and booking staff will see a reduction in the administrative burden of chasing hospital appointments on behalf of patients.
- The volume of Did Not Attends (DNAs) will reduce, because patients will agree on their date, and consultants will have a more secure referral audit trail.

What do all of these deliverables have in common? None have any yardstick that can be used to measure success or failure. None are attached to any dollar amount that can help justify the project. In fact, one could argue that all of these "success factors" could have been met by simply replacing the manual pencil sharpeners by electric ones!

I made the assertion in Chapter 5 that while many organizations claim to use an ROI (return on investment) yardstick to justify new projects, few really do. NPfIT is an excellent example of this. There is not a single ROI measurable included in the so-called success factors.

SIP is dogmatic about the need for measurable success factors tied to dollar amounts. It is a critical part of the prioritization activity of Phase 4 and is made concrete in the Value Graph Analysis. What would SIP-mandated measurable success factors have looked like? Here are some possible examples:

- A reduction by 50 percent in the personnel hours spent managing patient booking. This will save 140 million person hours per year at a savings of approximately $1.56 billion annually.
- A reduction by 50 percent of the DNAs (Did Not Attends), for a savings of $780 million annually.

[8] NHS Business Plan 2005/2006.

- A reduction by 75 percent of the cost of managing patient records, for a savings of $3.50 billion annually.

Do these specific measurables make sense? They are at least consistent with the NHS released data. I have no way to know if they are accurate or not, but these are the kind of measurements that would have served two purposes. First, they would have allowed the NHS to determine if it had, in fact, met its goals once (or if) NPfIT is ever completed. Second, they could have been used to convince a skeptical public that the project was worth undertaking in the first place.

NHS is in the process of learning a very expensive, very painful lesson. Complexity is your enemy. Control it, or it will control you.

Summary

In this chapter, I looked at a system called NPfIT (National Program for Information Technology). This system shares three characteristics with many other IT systems:

1. It is highly complex.
2. It is very expensive.
3. It is headed down a path of failure.

The lessons we can learn by examining NPfIT can help us avoid similar problems in other large projects. We saw how the features of SIP could have helped control the complexity of this massive project and, by extension, can be used to avoid similar problems in other projects.

SIP's obsession with complexity control would have helped bring sanity to a discussion of the business requirements of NPfIT. SIP's partitioning could have pinpointed areas of extreme complexity very early in the project. SIP's simplification could have removed as much as 98 percent of the project's complexity, and possibly more. SIP's iteration could have highlighted risks early in the project, where they could have been corrected easily and helped transform a world of skeptics to a world of supporters.

If SIP could do all of this for a project of the complexity of NPfIT, what can it do for your project?

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Z

Author Biography

Roger Sessions is the CTO of ObjectWatch. He has written seven books, including *Software Fortresses: Modeling Enterprise Architectures,* and more than 100 articles. He is on the Board of Directors of the International Association of Software Architects, is Editor-in-Chief of *Perspectives* of the International Association of Software Architects, and is a Microsoft recognized MVP in enterprise architecture. He has given talks in more than 30 countries and 70 cities, and at 100 conferences on the topic of enterprise architecture. He consults with major corporations and public-sector organizations throughout the world on the need to understand, model, and control complexity at the enterprise architectural level. *ComputerWorld* (New Zealand) describes him as "The Simplicity Guru." He holds multiple patents in software engineering and enterprise architecture.

More than 10,000 developers, architects, and executives follow Roger's writing regularly in his *ObjectWatch Newsletter*, now in its fifteenth year. For a free subscription or for information on contacting Roger, see the *ObjectWatch* web site at *www.objectwatch.com*.

ObjectWatch offers the SIP methodology for controlling enterprise architectural complexity primarily through partner relationships. For information about a SIP partner in your area, or to inquire about becoming a SIP partner, contact *information@objectwatch.com*.