# Table of Contents

Introduction ........................................................................................................................... 3  
Defining an Enterprise Architecture ......................................................................................... 3  
History of Enterprise Architectures .......................................................................................... 5  
Public Sector Case Studies ..................................................................................................... 6  
Private Sector Case Studies .................................................................................................... 7  
Complexity ............................................................................................................................. 8  
   Modeling Complexity .................................................................................................. 9  
   Partitioning .............................................................................................................. 10  
   Iteration ................................................................................................................ 12  
Partitioned-Iteration ............................................................................................................. 14  
Existing Enterprise Architecture Frameworks ......................................................................... 16  
Advantages of the Partitioned-Iterative Approach ................................................................. 18  
Rules for Success ................................................................................................................. 21  
   Rule 1. Start with Low Hanging Fruit ................................................................. 21  
   Rule 2. Leverage Economy of Small-scale ......................................................... 24  
   Rule 3. Centralize Interoperability, Decentralize Implementations ...................... 24  
Summary ............................................................................................................................. 25  
Glossary ............................................................................................................................... 27  
References ........................................................................................................................... 30
**Introduction**

When enterprise architectures work well, they are a tremendous asset in finding effective ways to better use technology. When they don't work well, they can be a huge counter-productive drain on precious organizational resources. All too often, it is the latter case which is realized.

You can’t afford to ignore the potential benefits of a well done enterprise architecture. These benefits include decreased costs, increased revenues, improved processes, 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 diminished executive credibility.

But don’t despair. You can have the benefits and avoid the problems. This paper will show you how. I’ll discuss some important clues as to how to approach an enterprise architecture from two unrelated disciplines: probability theory and war strategy.

From probability theory, we learn some important lessons about the nature of complexity. The failure to manage complexity is, I believe, the key reason so many attempts at creating enterprise architecture go awry. Few existing methodologies for enterprise architectures address the management of complexity in any meaningful way. And the architectures of today’s organizations are highly complex.

So how do you manage complexity? The answer comes from the unlikely field of war strategy. You will see how a lesson learned by fighter pilots who were trying to survive dog-fights fifty years ago contains an important clue for building complex enterprise architectures today.

Finally, I’ll put it all together into a series of guidelines for developing a good enterprise architecture that will be a critical organizational asset rather than an embarrassing and expensive liability.

**Defining an Enterprise Architecture**

A good starting place for a discussion on enterprise architectures is to agree on a definition of the term *enterprise architecture*.

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

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

While succinct, this definition captures neither the high cost typically associated with developing an enterprise architecture nor the business justification for undergoing the process.

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. [Wiki]
The Wikipedia definition better captures the complexity of so many of today's enterprise architectural methodologies. An even more exhaustive definition comes from the U.S. Government's Office of the CIO:

A strategic information asset base, which defines the business, the information necessary to operate the business, the technologies necessary to support the business operations, and the transitional processes necessary for implementing new technologies in response to the changing business needs. [Gov]

Are you starting to get a sense of why enterprise architectures can be so expensive? This is a lot of information to document.

The expense of creating a typical enterprise architecture is dominated by three factors:

- The exhaustive timeframes needed to perform a comprehensive analysis.
- The large number of highly placed internal staff that become absorbed by the process.
- The army of expensive consultants that are necessary to navigate through the complex methodologies.

But it doesn’t have to be this way. Here is my definition of enterprise architecture, one that contains several hints about how the process can be improved:

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.

Many will argue that my definition is not exhaustive enough. I disagree. In my definition, the goal of an enterprise is not to completely document every business process, every software system, and every database record that exists throughout the organization. Instead, my definition assumes a much more pragmatic goal: finding opportunities to use technology to add business value.

If adding business value is not the ultimate goal of an enterprise architecture, then 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.

Part of the difference between my approach and that of others 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. This level of detail is unnecessary and inappropriate in an enterprise architecture.

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?
- 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?
Notice that in this approach there is no such thing as a finished enterprise architecture. Instead, the enterprise architecture is seen as a living set of documents that guides the use of technology. It is much more analogous to a city plan than to a building blueprint.

The analogy of a city plan to an enterprise architecture was first introduced by Armour in 1999 [Arm] 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:

- What type of buildings will be allowed in which zones (e.g. business or residential)?
- How do buildings connect into the city infrastructure (e.g. plumbing and electrical)?
- What impact will buildings have on others of its ilk (e.g. air quality and traffic flow)?
- Are the buildings built to a standard that will not endanger its inhabitants (e.g. 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.

**History of Enterprise Architectures**

The birth of the field of enterprise architectures is generally credited to an article published in the IBM Systems Journal in 1987 titled “A framework for information systems architecture” by J. A. Zachman [Zac]. Later, Zachman re-named his “information systems” framework to be an “enterprise architecture” framework. Today this framework is known simply as the Zachman Framework.

In 1994, the United States Department of Defense first introduced the Technical Architecture Framework for Information Management (TAFIM) [TAF]. TAFIM was proclaimed to be the new enterprise architectural standard for all defense work. TAFIM went through several iterations before it was finally discontinued in 2000.

While TAFIM is no longer alive, some pieces of it were adopted in The Open Group Architectural Framework (TOGAF). TOGAF is an “open source” framework controlled by The Open Group and is now in version 8.1 [TOG]. TOGAF is probably the most popular enterprise architecture framework in the private sector today followed closely by Zachman.

In 1996, the discipline of enterprise architectures received a major boost by the U.S. Congress. In that year Congress passed the Clinger/Cohen act, also known as the Information Technology Management Reform Act (ITMRA). This act gave the Office of Management and Budget (OMB) widespread authority to dictate standards for “analyzing, tracking, and evaluating the risks and results of all major capital investments made by an executive agency for information systems.” [Cli]

Clinger/Cohen required each executive agency to establish a Chief Information Officer, who, among other duties, was responsible for “developing, maintaining, and facilitating the implementation of an integrated framework for evolving or maintaining existing information technology and acquiring new information technology to achieve the agency’s strategic goals and information resources management goals”. [Cli]
Oddly, Clinger/Cohen never mentions the concept of an enterprise architecture. Nevertheless, the OMB interpreted this act as a mandate for a universal enterprise architectural framework for the entire U.S. Government. This Framework became known as the Federal Enterprise Architectural Framework (FEAF) [FEA]. Today, every executive agency from the Department of Justice to the Department of Homeland Security to the Department of Agriculture has been required by the OMB to develop an enterprise architecture, and to show how that enterprise architecture is aligned with FEAF.

The net result, therefore, of Clinger/Cohen has been that all IT-related work done by or for the U.S. Government is now, in theory at least, being done under the auspices of a single, common enterprise architecture.

**Public Sector Case Studies**

Clinger/Cohen not only mandated the use of a federal enterprise architecture (or, at least, has been so interpreted), it also mandated that the efficacy of the program be regularly monitored and reported. This has given us an important opportunity to see a large enterprise architecture in action.

In effect, we have a major case study of the use of an all-encompassing enterprise architecture (FEAF) by probably the single most complex organization in the world, the U.S. Government. And, since FEAF has been strongly influenced by both TOGAF and Zachman, we can extrapolate the lessons learned from FEAF to those two enterprise architecture frameworks as well.

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 critical the government agency, the more likely it is to have major IT failures.

In November of 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. [GAO1]

The Department of Defense has come under repeated criticism. For example, in June of 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. [GAO2]

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. [GAO3]

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 more than $100 million on a system that was proven ineffective by Hurricane Katrina. Other Federal groups that have been the subject of GAO criticism include the Census Bureau, Federal Aviation Authority, National Air and Space Administration, HUD, Health and Human Services, Medicare, and Medicaid.

In perhaps the ultimate irony, one of the departments criticized for failure to comply with Clinger/ Cohen was the Office of Management and Budget, the very organization that is supposedly monitoring the other agencies for compliance with Clinger/Cohen!

If the Federal Government is the most comprehensive case study of the value of enterprise architectures, then the field is in a pretty sorry state.

**Private Sector Case Studies**

Although private industry failures are not so prone to make headlines, the private sector, too, is perfectly capable of bungling IT on a large scale. Private sector failures that seem largely attributed to failures in enterprise architectural methodologies include:

- McDonald’s failed effort to build an integrated business management system that would tie together their entire restaurant business. Cost: $170 million. [Mac]
Ford’s failed attempt to build an integrated purchasing system. Cost: $400 million. [For]

KMart’s failed attempt to build a state-of-the-art supply chain management system. Cost: $130 million. [Kma]

Nicholas G. Carr, in a recent article in the New York Times, concluded:

A look at the private sector reveals that software debacles are routine. And the more ambitious the project, the higher the odds of disappointment. [Car]

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. [Cha]

From both the public and the private sector, we have example after example of extensive and expensive enterprise architectures failing to align business needs and technology solutions. Given the available evidence, one must conclude that either enterprise architectures are not worth pursuing or that the methodologies we are using to create enterprise architectures are highly flawed.

Intuitively, it seems that an enterprise architecture should be a good thing. Let’s re-examine the definition I gave earlier.

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.

It’s hard to argue with the idea of finding better ways to meet our organizational goals through the use of technology. So that leaves the methodologies as our prime suspects.

**Complexity**

In order to understand the failures of the various enterprise architecture methodologies, we need to understand what characteristics are shared by all of the failed attempts at creating enterprise architectures. What characteristics are shared by systems as diverse as the Internal Revenue Service’s Financial Management System, the Department of Defense’s Business Management Systems, the Department of Homeland Security’s Disaster Management System, the FBI’s Virtual Case File System, McDonald’s Business Management System, Ford’s Procurement System, and KMart’s Supply Chain Management System?

There is only one characteristic shared by every one of these systems. That characteristic is *complexity*. All of these systems are highly complex. I believe that a major weakness of FEAF, TOGAF, and Zachman is in their failure to manage system complexity.
Unfortunately, complexity is not a passing whim. There are three predictions that we can confidently make about the future of IT:

1. Complexity is only going to get worse.
2. If we don’t find approaches to managing complexity we are doomed to fail.
3. The existing approaches don’t work.

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. [Mur]

Modeling Complexity

In order to manage complexity, we must first understand it. And being able to model complexity is a big step forward in understanding it.

The best model that I know of for complexity is dice. This model has the advantages of being intuitive, visual, predictive, and mathematically grounded. I will state the fundamental principle of this model as Sessions’s Law of Complexity:

The complexity of a system is a function of the number of states in which that system can find itself.

Let’s use dice to get a handle on the significance of this law. I’ll compare the complexity of two arbitrary systems, System A and System B, as shown in Figure 1. System A is composed of a single two-headed die (i.e. a “coin”). System B is composed of a single six-sided die (i.e. a standard die).

System A

System B

Figure 1. System A and System B

System A has two potential states: heads and tails. System B has six potential states: 1, 2, 3, 4, 5, and 6. Based on Sessions’s Law of Complexity, System B is 6/2, or 3 times more complex than System A. Even if we don’t understand the mathematics, this seems intuitively plausible.

In general, the number of states of a System of D die, each with S sides, is $S^D$. With this formula, we can calculate the complexity of other systems.
Now let’s compare System B and System C. System B consists of one six-sided die, as before, and System C consists of three six-sided die, as shown in Figure 2.

![Figure 2. System B and System C](image)

The number of states of B is still $6^1$, or 6. The number of states of C is $6^3$, or 216. According to Sessions’s Law of Complexity, System C is 216/6, or 36 times more complex than System B.

If you were to predict a specific outcome of throwing the dice in System C, you would have one chance in 216 in getting it right. If you were to predict a specific outcome of throwing the die in System B, you would have one chance in 6 in getting it right. If you were to make repeated guesses for both systems, you would be right, on average, 36 times more often with System B than you would be with System C. Because System B is less complex than System C, it is easier to predict.

**Partitioning**

With this basic model of complexity, we can gain some insight into how complex systems can be better organized.

Let’s compare another two systems, System C and System D, both composed of three six-sided dice. In System C, the dice are all together, as before. In System D, the dice are divided into three partitions. These two systems are shown in Figure 3.

![Figure 3. System C and System D](image)

Let’s assume that we can deal with the three partitions independently, in effect, three subsystems each like System B.
We already know that the complexity of System C will be 216. The overall complexity of System D is given as the complexity of the first partition plus the complexity of the second partition plus the complexity of the third partition. The complexity of each partition is still given by the rule of $S^D$, which is $6^1$, or 6. The complexity of the whole System D is therefore $6 + 6 + 6$, or 18.

If you aren’t convinced of this, imagine inspecting System C and System D for correctness. With System C you would need to examine 216 different states, checking each for correctness. With system D you would need to examine 6 different states to ensure that the first partition is correct, another 6 states to ensure that the second partition is correct, and another 6 states to ensure that the third partition is correct.

In general, the complexity of a system of $P$ partitions, each with a complexity of $C$ is $C \times P$. If each partition will always hold one die, then this formula becomes $S^1 \times D$, which reduces to $S \times D$.

Now let’s extrapolate these lessons to partitioned and non-partitioned systems in general. To simplify this, we will assume that the partitions always hold one die (as did System D). Then the complexity of the non-partitioned system (e.g. System C) is always $S^D$ and the complexity of the partitioned system (e.g. System D) is always $S \times D$.

So the question then boils down to how the two formulas ($S \times D$) and ($S^D$) compare to each other. Table 1 shows different values for $S$, $D$, the partitioned formula ($S \times D$), and the non-partitioned formula ($S^D$).

<table>
<thead>
<tr>
<th>$S$</th>
<th>$D$</th>
<th>$S \times D$ (partitioned)</th>
<th>$S^D$ (non-partitioned)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>18</td>
<td>126</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>24</td>
<td>1,296</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>30</td>
<td>7,776</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>36</td>
<td>46,656</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>40</td>
<td>279,936</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>46</td>
<td>1,679,616</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>52</td>
<td>10,077,696</td>
</tr>
</tbody>
</table>

Table 1. Partitioned Complexity

Using Table 1, you can see how much more complex a non-partitioned system of 9 dice is compared to a partitioned system containing the same number of dice. It is the ratio of 10,077,696 to 52. In layman’s terms, a lot!

You probably get the point of Table 1. The greater the overall complexity of a system, the greater the impact partitioning has on reducing that complexity.

So this is clue number one about how to deal with complexity. Partitioning.
Iteration

Once we have used partitioning to reduce complexity, we must make another decision. How do we analyze the complexity we have left? There are two choices: iteratively (left to right) or recursively (top to bottom). You can see the difference in these two approaches by looking again at the complexity model of partitioned dice, as shown in Figure 4.

![Partitioned Dice](image)

Figure 4. Partitioned Dice

In an iterative approach to complexity, we analyze each partition individually. For example, we first fully analyze Partition 1. Then we analyze Partition 2. And we continue until each partition is analyzed.

In a recursive approach to complexity, we analyze a horizontal layer of each partition. For example, we first analyze the case in which all of the dice are ones. Then we analyze the case in which the first die is a one and the remaining dice are twos. We continue this analysis until we have completed the last possible case for all partitions.

Which approach is better? Iterative or recursive?

For the answer to this, we must return to the year 1950, and the inquisitiveness of a Air Force Colonel named John Boyd [Boy]. Boyd was not worried about how to build complex IT systems. He probably never heard of an enterprise architecture, or even an IT system. He was worried about how to win dogfights.

A dogfight is a one-on-one battle between two fighter pilots, each trying to shoot the other down before he, himself, is shot down. You can see why an Air Force Colonel might be interested in winning them.

Flying a fighter jet in a combat operation is very complicated. The pilot needs to take into account information from many different sources, all while an enemy pilot is trying to shoot him down. You can get an idea of the complexity that one of Boyd’s pilots had to cope with by looking at a jet’s cockpit, shown in Figure 5.

Boyd was interested not just in any dogfight, but specifically in dogfights between MiG-15s and F-86s. As an ex-pilot and accomplished aircraft designer, Boyd knew both planes very well. He knew the MiG-15 was a better aircraft than the F-86. The MiG-15 could climb faster than the F-86. The MiG-15 could turn faster than the F-86. The MiG-15 had better distance visibility.
There was just one problem with all of this. Even though the MiG-15 was considered a superior aircraft by Boyd and most other aircraft designers, the F-86 was preferred by pilots. The reason it was preferred was simple: in one-on-one dogfights with MiG-15s, the F-86 won nine times out of ten.

This problem fascinated Boyd. Why would an inferior aircraft consistently win over a superior aircraft?

In order to solve this anomaly, Boyd needed to understand how pilots actually operate in dogfights. Boyd had an advantage here. He was not only a pilot, but one of the best dogfighters in history. He therefore had some first-hand knowledge of the subject.

Let’s consider a pilot involved in a dogfight, we’ll call him Pete. Boyd proposed that Pete’s dogfight consists of four distinct stages. In the first stage, Pete observes the state of the world around him, including the enemy plane. In the second stage, Pete orients himself with respect to that state. In the third stage, Pete plans on an appropriate action. In the fourth stage, Pete acts.

So Pete first observes, then orients, then plans, then acts. Boyd called this sequence OOPA (observe, orient, plan, act).

But, and here is a critical fact, Pete doesn’t just do this once. He does this over and over again. In fact, Pete is constantly looping through this sequence of OOPA. And, of course, so is his opponent. So who will win? Pete? Or the anti-Pete? If Pete is flying the F-86, we know he will probably win. But why?

It would seem that the anti-Pete flying the MiG-15 would be better at OOPA-ing than Pete. Since anti-Pete can see further, he should be able to observe better. Since he can turn and climb faster, he should be able to react faster. Yet the anti-Pete loses and Pete wins.

Boyd decided that the primary determinant to winning dogfights was not observing, orienting, planning, or acting better. The primary determinant to winning dogfights was observing, orienting, planning, and acting faster. In other words, how quickly one could iterate. Speed of iteration, Boyd suggested, beats quality of iteration.

The next question Boyd asked is this: why would the F-86 iterate faster? The reason, he concluded, was something that nobody had thought was particularly important. It was the fact that the F-86 had a hydraulic flight stick whereas the MiG-15 had a manual flight stick.

---

1 Readers familiar with Boyd’s work may recognize that Boyd called his loop OODA for Observe, Orient, Deploy, Act. However I have change the deploy to plan for two reasons. First, technology readers will be confused by the acronym for Object-Oriented Design and Analysis, also OODA. Second, as I have read Boyd’s works, I have concluded that “plan”, as used in the IT context, is closer to Boyd’s original meaning than is “deploy”.
Without hydraulics, it took slightly more physical energy to move the MiG-15 flight stick than it did the F-86 flight stick. Even though the MiG-15 would turn faster (or climb higher) once the stick was moved, the amount of energy it took to move the stick was slightly greater for the MiG-15 pilot.

With each iteration, the MiG-15 pilot grew a little more fatigued than the F-86 pilot. And as he gets more fatigued, it took just a bit little longer to complete his OOPA loop. The MiG-15 pilot didn’t lose because he got outfought. He lost because he got out-OOPAed.

I’ll state Boyd’s discovery as Boyd’s Law of Iteration: In analyzing complexity, fast iteration almost always produces better results than in-depth analysis.

**Partitioned-Iteration**

So now we have two lessons as to how to better manage the complexity of enterprise architectures. From the study of dice, we know that *partitioning* complexity is one of the keys to *reducing* complexity. From Boyd’s study of jet fighters, we know that the best way to analyze partitions of complexity is to iterate through them, and to do that iteration with a focus on speed rather than completeness.

Now let’s see how we can apply these two lessons to complex enterprise architectures. Consider a large business management system, similar to the one McDonald’s lost $170 million trying (and failing) to create. I’ll call this system LCBMS, for Large, Complex, Business Management System. Imagine LCBMS needs to include the following functionality:

- Global HR
- Payroll
- General Ledger
- Accounts Payable
- Accounts Receivable
- Billing
- Assets
- Accounting
- Treasury
- Employee Portal

How do we partition the LCBMS? This bulleted list gives us an immediate clue. Don’t create a mega-design for an LCBMS. Create an architecture for a Global HR system, one for a Payroll system, etc. In other words, don’t design a single, large, complex system. Design a number of small, simple systems. Design one, build it, and then start on the next one. The *partitioning* decreases the overall complexity. The *iteration* increases the probability of success.

We can assume that the earlier examples of architectural failures (e.g. the FBI’s failed Virtual Case File System or McDonald’s failed BMS) were all based on the standard enterprise architectural methodologies such as FEAP, TOGAF and Zachman. These methodologies are all non-iterative. It’s
surprising, therefore, that they all failed. It also isn’t surprising that those failures were very expensive.

All enterprise architecture methodologies agree that the following phases must occur:

- business architecture design
- technical architecture design
- implementation
- testing
- deployment

Traditional methodologies take these phases one at a time, fully completing one before the next is started. This is shown in Figure 6.

As shown in Figure 6, traditional methodologies start with an in-depth business architecture of the target system, say our hypothetical LCBMS. Then we create a technical architecture. Then we implement. Then we start testing and debugging. Then we deploy. Notice that we have no business value associated with the project until the final deployment stage is completed.

This analysis also shows us what these organizations should have done. They should have used a partitioned-iterative approach.

![Figure 6. The Mega-Design Approach](image)
The partitioned-iterative approach looks quite different. Figure 7 shows the partitioned-iterative approach in action. Notice that we follow the same phases as before, but now we take them partition by partition.

Figure 7. Partitioned-Iteration

The result of the partitioned iteration approach is a constant streaming forth of functionality. We don’t start one partition until the previous one is completed. In large, complex organizations it is not unusual to have simultaneous projects in parallel but the number should be kept to a minimum, especially in the early iterations. Multiple concurrent projects will increase the need for coordination.

Existing Enterprise Architecture Frameworks

The standard existing enterprise architecture frameworks (including TOGAF, Zachman, and FEAF) share a similar history. They were all heavily influenced by the world of object-oriented design and analysis (OODA).

The fact that these frameworks are in the OODA generation is significant because it dates them pre-SOA (Service Oriented Architectures). Today, most large systems are based on the notion of interoperability of autonomous applications through Web service standards (such as SOAP, WS-Security, and the like). This notion is very much tied to the service-oriented world that did not exist back when the earlier architectural frameworks were created.

Object technology is intended for application implementation, not for architecting enterprises. Their biggest drawback is their inability to manage complexity.
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.

[Roy]

Issues that need to be addressed for large, complex enterprise architectures have to do with the interactions between applications, not the implementations of the applications themselves. The notion that a large system could be made up of interacting autonomous applications, the technical equivalent of partitions, is a notion that did not reach maturity until long after the era of Objects. This, in fact, was the defining feature of the era of SOAs.

The use of SOAs helps manage complexity at the technical level. Just as the business architecture must be partitioned to reduce complexity to manageable levels, so too must the technical architecture. SOAs are by far the best approach we have today to accomplish this technical partitioning.

OODA frameworks often claim to support a notion of iteration, but it is much different than the iteration in partitioned-iteration. The OODA notion of iteration is really not so much about iteration as it is about recursion, as shown in Figure 8. It tackles system complexity by taking it on in bite-sized portions. It doesn’t reduce system complexity. It merely minimizes the amount of complexity that must be faced at any one time.

![Figure 8. OODA Approach to Enterprise Architectures](image-url)
Partitioned-iteration differs from the OODA approach in two important ways. First, and most importantly, it reduces complexity through partitioning rather than attempting to manage it through recursion. But secondarily, partitioned-iteration emphasizes speed of iteration at the expense of exhaustive planning. The theory is that we learn more by doing than we do by planning. Figure 9 shows the partitioned-iteration approach.

Advantages of the Partitioned-Iterative Approach

The partitioned-iterative approach starts delivering value much earlier than the recursive OODA approach. The reason is that we are delivering our system in segments. As long as we design our vertical segments so that each contains measurable business value, iteration translates to faster time to value (TTV).

In the business world, TTV is a critical value measurement, even more important than return on investment (ROI). ROI is a long-term goal. Any project, regardless of how poorly organized, how over
budget, and how miserably aligned with the business requirements can claim a good ROI as long as the project costs are amortized over a long enough period of time. The question is, can the business survive long enough to realize the return!

TTV is much more immediate. Different people have different definitions for TTV, but I define TTV as the amount of time between the approval of a project’s budget and the time the executives feel the project is having a positive impact on the organization’s business. In other words, it is the length of time between dollars going out and perceived value coming in. ROI is largely irrelevant to TTV.

For example, say management approves a budget of $20 million to reorganize customer support. You could take the traditional recursive OODA approach, in which case there will be no perceived value until the entire $20 million project has been completed (if you are lucky). Or you can take a partitioned-iterative approach, breaking the project into value-containing vertical partitions.

Say, the first segment is an on-line, easy-to-access library of useful customer-accessible information. This vertical slice can be delivering perceived value long before completion of the rest of the project, even before it has a positive ROI. If you can show, for example, that within the first month the library has already decreased the length of customer calls to the help desk by 20%, you are showing value even though you may be years from showing a positive ROI for the overall project.

Which project do you think is more likely to be canceled? A project that is returning high-visibility value within a few months (partitioned-iteration) or a project that has returned zero perceived value even after two years of expenditures and effort (recursive OODA)?

Fast time-to-value is of major importance in keeping projects high on management’s priority list. Out of sight, out of mind is a maxim that certainly applies to IT projects.

The second advantage of the partition-iteration approach is that it increases the overall chances of a successful project. This is because you can apply the lessons learned in earlier iterations to efforts that will occur in future iterations.

Going back to the customer call center, let’s say that your immediate partition is completing a new customer user interface and that one of your future iterations is creating a call center user interface. Suppose you had assumed that you could complete the new customer user interface in six person-months and the new call center interface in eight person-months. But when you finish the new customer user interface you discover that it took nine person-months instead of six. Obviously you have a problem in your estimation of the work involved in creating user interfaces.

You now have an opportunity to learn from your mistakes. Let’s say you identified the problem as an underestimate of the time needed to train your developers. When tackling the call center interface, you can make sure that you assign experienced developers. If the development tools were more complicated than you thought, then you can consider switching tools. If the process simply takes longer than you thought it would, you can adjust the remaining project schedule.

The important point is that with partitioned-iteration you discover problems early, before they have run rampant through the entire project. And, if they cannot be fixed, at least they can be recognized before the resulting delays become a surprise to management. Management may not like delays, but it hates surprises even more.
Managing expectations effectively is a key component of almost any project. By using a partitioned-iterative approach, you can more easily react to potential problems and can more effectively communicate realistic expectations to the executive team.

A third advantage of partitioned-iteration is that it ties up much less of the time of your highest level, most critical (and most expensive) staff. This is because the design of each partition requires consensus only among the staff that are directly involved in that partition. In the recursive-OODA approach, most of the staff are involved in most of the design decisions, and seemingly simple design decisions are often painstaking to conclude.

A fourth advantage of the partitioned-iteration approach is that it results in designs that map naturally into an SOA. SOAs are made up of small autonomous services rather than the monolithic monsters typical of the OODA approach. These small, autonomous, interoperating services are much more amenable to reuse, collaboration, interoperability, agility, and automated workflow than their OODA counterparts.

And finally, the partitioned-iteration approach offers a much better solution to feature-creep than does the OODA approach. There are three factors that favor the partitioned-iteration approach in reducing feature-creep.

1. The number of partitions is much greater in partitioned-iteration than in the OODA approach. This reduces the psychological need for individuals to feel that a given partition is their one and only chance to incorporate some desired feature.

2. Fewer people are involved in the design of any one of these partitions than in the OODA approach. This means that in any given iteration, fewer people are being asked their opinions on new features.

3. Partitioned-iteration emphasizes time to value (TTV) instead of return on investment (ROI) that is the darling child of OODA. Most people associate positive ROI with feature-richness. After all, more features means that more people can use a given application. The more people using the application, the greater the “return” that application will deliver. This over-focus on ROI is an irresistible magnet for feature-creep.

TTV, the focus of the partitioned-iteration approach, emphasize quick delivery. Everybody understands the need for time-boxing to achieve quick delivery. Time-boxing means limiting features to those that are critical and that can be delivered within a given timeframe. This lends itself to an organizational spirit that is willing to take a harder look at requested features and cut those that are not truly necessary.

As you can see, there are a number of major benefits that an organization can realize from switching from an OODA enterprise architectural methodology to a partitioned-iteration methodology. These include:

- Faster delivery of quantifiable business value
- Improved likelihood of successful project completion
- More opportunities to learn from previous successes and failures
- More accurate project plans
• Better alignment with technical service-oriented architectures
• Natural resistance to feature-creep

These are some important advantages that should pique the interest of any organization that is looking to find real value from its architectural efforts.

**Rules for Success**

Now that I have given you some background in the theory and practice of a partition-iteration approach to enterprise architectures, I will give you a set of rules that I think will help you be more successful with this approach.

**Rule 1. Start with Low Hanging Fruit**

Once you have taken a first pass and identified your project/partitions, you need to decide which to tackle first. Many practitioners recommend that you first go after the higher risk projects. The logic is that if there are going to be problems, you will discover them earlier. I disagree with this approach.

The first thing you want to do in developing an enterprise architecture is to demonstrate success. This will help you build momentum for your larger goal. This is especially true in organizations that have been burned in the past by enterprise architectures with OODA methodologies.

Thus your first, most important goal in the early iterations is establishing credibility. The best way to establish credibility is to deliver some highly visible successes that can be understood by everybody in the organization.

---

**Get on Board the FastTrack**

Let Roger Sessions bring our Fast Track™ Methodology to your organization: a five day, intensive process designed to provide essential roadmaps for high value organizational alignment of business and technology. The deliverables from this engagement include the following:

• Workshop in the basics of iteration, business partitioning, the Fast Track process, ABC (autonomous business capability) Models, and Value Graphs.
• An ABC Model of your organization with identified potential technology projects.
• A Value Graph Tool with metrics that are custom-tailored to your organization.
• Value Graphs of each potential technology project with best approaches (SOA, SaaS, Re-engineering, etc.) identified.
• An iterative process with tracking mechanisms for critical predictive metrics.

Questions? Contact us at information@objectwatch.com.
In prioritizing partitions, Table 2 gives an overview of the most important factors, how best to analyze each, and how each should influence the overall prioritization.

A good way to prioritize partitions is to use a Priority Target, as shown in Figure 10. In a Priority Target, each axis represents one of the priority considerations that I listed in Table 2. Each axis is given with positive measurements closer to the center and negative measurements closer to the edge.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Analysis</th>
<th>Influence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceived Value</td>
<td>How much value would an executive in your organization place on this project’s completion?</td>
<td>Start with partitions that have a high perceived value in your organization</td>
</tr>
<tr>
<td>Visibility</td>
<td>How visible would the success of this project be across the organization?</td>
<td>Start with projects that have high visibility across the organization. This will improve your chances of getting buy-in from others on future projects</td>
</tr>
<tr>
<td>Cost</td>
<td>How large a portion of the entire budget is allocated for this part of the larger project?</td>
<td>Start with projects that have a low cost. You want to establish a track record of Time to Value</td>
</tr>
<tr>
<td>Risk</td>
<td>What is the risk of this project?</td>
<td>Start with projects with low risk. As I said earlier, your goal is to establish credibility. You can afford to take chances in later iterations.</td>
</tr>
<tr>
<td>Measurability</td>
<td>How easy is it to measure your success, and thereby make a convincing case of how successful you were?</td>
<td>Start with projects that are easy to compute the value the project provided.</td>
</tr>
<tr>
<td>Time to Completion</td>
<td>How long is it likely to take to complete this project?</td>
<td>Start with projects that can be completed as quickly as possible. Your goal is to establish a track record of delivering value quickly.</td>
</tr>
</tbody>
</table>

Table 2. Partition Prioritizing Factors
Once you have created the Priority Target, take your “shots” at each one of the axis. If, for example, this partition has a medium risk, place a circle on the risk axis in the yellow region of the target. When you have finished, you can visually see which candidate projects (partitions) should be tackled first.

For example, let’s say you have identified two projects. The first is developing a customer accessible knowledge base. The second is creating a single-sign-on procedure for better security. You create the priority targets for the two partitions and the result is as shown in Figure 11. It becomes immediately obvious which of the two projects better qualifies as “low hanging fruit”.

![Figure 10. Priority Target](image)

![Figure 11. Comparing Two Priority Targets](image)
Rule 2. Leverage Economy of Small-scale

A common fallacy in enterprise architectures is that you can achieve economies by doing things in large scale. The theory goes that if enough people are doing a large enough project, there are inevitably going to be many redundancies. The elimination of those redundancies results in reduced development and maintenance costs. The larger the project, the more redundancies. The more redundancies, the more opportunities for redundancy elimination. The more redundancy elimination, the lower the overall cost of the project.

Unfortunately, this theory ignores an even more basic law of project management: the law of diminishing resource returns. The more people involved in a project, the less efficient the project as a whole becomes. The law of diminishing resource returns is a corollary of the famous Brooks’ Law. Fred Brooks first observed over 30 years ago that adding more resources to a late software project only makes the project later [Bro]. For this and other observations he was awarded the Turning Prize in 1999.

A relatively small group attacking a well-defined partition of the enterprise can do a reasonable job in a reasonable timeframe. A large group attacking a non-partitioned enterprise may indeed find redundancies. But the cost of finding those redundancies, arguing about how best to eliminate them, and trying to agree on a design for a unified approach will in almost all cases far surpass the cost of the redundancies themselves.

It is true than economies come in scale. But true economies come in small, not large scale.

Rule 3. Centralize Interoperability, Decentralize Implementations

Many organizations build enterprise architectures that are overly centralized. Frequently, they create an office of enterprise architecture and give this office authority over a wide range of decisions. This excessive centralization can result in a stifling bureaucracy and project gridlock.

A centralized enterprise architectural organization is necessary. But the central organization needs to focus on the right issues and not get bogged down on issues that are irrelevant.

The general rule of thumb is that decisions that impact interoperability should be centralized. Decisions that impact implementations should be decentralized. Not knowing which decision is which is a common error in enterprise architectural departments.

Let’s look at some typical errors that come up in enterprise architectures.

- Platform: Many organizations attempt to define a standard software platform, often debating endlessly between, say, Microsoft’s .NET, IBM’s WebSphere, or BEA’s WebLogic. This effort is misplaced. Platform is an implementation decision and has no bearing on how the applications on those platforms will work together. As long as the platform meets the organization’s interoperability requirements, the application team should be given latitude to choose the best platform for their application’s needs.

- Data: Many organizations attempt to define a single data layer that will be shared by all applications in the organizations. This effort is often expensive and rarely successful. How data is stored should be treated as an implementation detail of an application.

- Business Intelligence: Most organizations treat data and business intelligence interchangeability. Whereas data (such as how a customer is stored in a database) is an
implementation detail, intelligence (such as what business we have conducted with a given customer) is an organizational asset. It is appropriate to decide how such intelligence will be shared. It is not appropriate to decide (at the enterprise level) how applications will keep track of the data that feeds this intelligence.

- **Code Sharing:** Many organizations believe that reuse is achieved through code sharing. It is somewhat amazing that this belief persists despite decades of failure to achieve this result. The best way to reduce the amount of code that a given project needs is through delegation of functionality (à la Web services), not through code sharing.

- **Web Service APIs:** Many organizations believe, correctly, that the use of Web services is critical to achieving interoperability. Many organizations think this means that the way applications use the Web service APIs should be standardized. In reality, the Web service APIs are far below the level of concern of applications. Applications typically make use of a buffering layer that is vendor specific, for example, the Windows Communications Framework layer provided by Microsoft’s .NET platform. The purpose of this layer is to insulate applications from needing to understand the intricacies of the Web service APIs. This buffering layer is specific to the platform, and is therefore part of the implementation details of the application.

The bottom line is that enterprise architecture is not the same thing as application architecture. Application architecture is about the design of applications. These designs should be the responsibility of the group owning the applications. This is one of the ways we achieve economies of small-scale (see rule 2: *Leverage Economy of Small-scale*). Enterprise architects should worry about how those applications work together, and thereby provide better value to the organization.

**Summary**

An enterprise architecture can be an important resource in helping an organization find better ways to use technology to support its critical business processes. Unfortunately, many organizations spend tremendous sums attempting to create enterprise architectures and get limited, or even negative, value from the effort. Multi-hundred-million dollar failures are common across both the public and the private sectors.

There are three primary reasons for such frequent expensive failures. The first is an over-reliance on recursive object oriented design and analysis (OODA) architectural methodologies. The second is the false notion that creating an enterprise architecture means developing a detailed blueprint of the entire organization. The third is a failure to break complex structures into smaller, more manageable projects.

OODA methodologies, like all recursive approaches, have a limited ability to manage complexity. Complexity is the main challenge faced in today’s highly volatile organizations.

This paper proposes a different approach to building enterprise architectures. The approach is based on vertically partitioning an organization’s processes, prioritizing the need to improve those processes, and then iterating through those processes with a focus on time-to-value (TTV). This approach is described as *partitioned-iteration*. Unlike recursive methodologies, partitioned-iteration attacks complexity head on.
This paper describes three rules for success in a partitioned-iteration strategy. They are:

1. Start with the low hanging fruit, focusing on quick time-to-value.
2. Leverage economy of small-scale, focusing on agile processes.
3. Centralize interoperability and decentralize implementation, focusing on reduced complexity through partitioning and faster iteration through agility.

Approaches to enterprise architectures based on partitioned-iteration have a number of advantages over those based on recursion. These include:

- Better use of technology to solve pressing business problems.
- Dramatic reductions in the cost of building complex systems.
- Faster delivery of high business value technical projects.
- Better control of overall system costs.
- Improved accuracy in prediction of delivery dates.
- Greatly improved probability of overall system success.

Partitioned-iteration is the most cost-effective approach we have to describing the goals of an organization, how those goals are realized by business processes, and how those business processes can be better served through technology.

In other words, partitioned iteration delivers high business value technical solutions as quickly as possible at the lowest possible cost. High value. Minimum time. Lowest Cost. That is the bottom line.

Roger Sessions is the CTO of ObjectWatch Inc. and author of six books and dozens of articles and white papers. He serves on the Board of Directors of the International Association of Software Architects (www.iasahome.org). Roger is a frequent keynote speaker at events throughout the globe and conducts seminars and training aimed at the architect community. Roger writes a monthly newsletter specifically for enterprise architects and IT managers called the ARCHITECT TECHNOLOGY ADVISORY (ATA). ObjectWatch specializes in helping organizations of all sizes realize the maximum possible value from enterprise architectures through consulting and training. More information about Roger Sessions and ObjectWatch is available at www.objectwatch.com.
Glossary

.NET — An umbrella term for Microsoft’s family of products that includes their Web services infrastructure.

agility — A measure or an entity’s (e.g. an enterprise’s) ability to respond rapidly to changing environmental conditions.

Boyd’s Law of Iteration — In analyzing complexity, fast iteration almost always produces better results than in-depth analysis. Named after Col. John Boyd.

Brook’s Law — Adding additional resources to a late project only makes the project later. Attributed to Frederick Brooks.


collaboration — The process by which two applications in different enterprises are able to programmatically work together, in contrast to interoperability.

enterprise architecture — A description of the goals of an organization, how those goals are realized by business processes, and how those business processes can be better served through technology.

enterprise architecture framework — A methodology for creating an enterprise architecture.

FEAF — See Federal Enterprise Architectural Framework.

Federal Enterprise Architectural Framework (FEAF) — An enterprise architectural framework used by the United States federal government to describe how the various governmental agencies and their IT systems are related to each other.

FEMA — The Federal Emergency Management Agency, the arm of the U.S. Government that is responsible for responding to disasters.

GAO — See General Accountability Office.

General Accountability Office — A branch of the United States Government that is responsible for monitoring the effectiveness of different organizations within the the U.S. Government.

Information Technology Management Reform Act — An act passed by the United States Congress in 1996 that requires all governmental organizations to use effective strategies and frameworks for developing and maintaining IT resources.

interoperability — The process by which two or more applications in the same enterprise can work together programmatically. In contrast to collaboration.

IT — Information Technology.

iteration — Any process that approaches the whole as a series of smaller, bite-sized pieces.

iterative architecture — An approach to architecting systems that designs, builds, and deploys large complex systems in stages.

iterative process — An approach to agile software development in which software development goes through rapid iterations of design, implementation, and re-evaluation, resulting (it is claimed) in
systems that are more closely aligned with changing business conditions. In contrast to waterfall process.

**ITMRA** — See *Information Technology Management Reform Act*.

**Law of Complexity** — The complexity of a system is related to the number of states in which that system can find itself. Often used to compare the complexity of two different approaches to building a system.

**low hanging fruit** — A term describing a subset of choices that deliver the maximum perceived value for the minimum calculated cost.

**Object-Oriented Design and Analysis** — An architectural framework that is based on object-oriented recursive decomposition.

**OODA** — see *Object-Oriented Design and Analysis*.

**OOPA** — (Observe, Orient, Plan, Act) A loop process first proposed by Col. John Boyd in relationship to jet fighters, although he described his loop as Observe, Orient, Deploy, Act. In Boyd’s description, this loop of observe, orient, plan, and act is done iteratively, and an entity that can iterative faster wins over an entity that iterates more completely.

**partitioned iteration** — An approach to building an enterprise architecture in which the enterprise is first partitioned into a series of autonomous vertical segments, then iteratively analyzed, and finally, where appropriate, improved through the use of better business and technical processes.

**priority target** — A visual representation of a proposed deliverable as a target, with lines radiating out from the center of the target representing prioritizing factors and “shots” along each axis showing relative importance of that factor for this deliverable. Shots are placed on each axis with closeness to the center of the target indicating strength of positive influence.

**Rational Unified Process (RUP)** — A process for developing software that is the basis for the IBM/Rational tool suite.

**recursive** — In analysis, a process by which a system is analyzed as being composed of multiple sub-systems, each of which are analyzed as being composed of multiple sub-systems, and so on until a set of final terminal systems are identified which can no longer logically be broken down.

**redundancy** — Overlap in functionality between multiple systems.

**Return on Investment** — A measure (in percent) of the business value of a project based on the increase in profit (either because of increased income or decreased expenses) divided by the cost of the project. For example, a project with a cost of $100,000 that returned $200,000 in increased profit has an ROI of 200%

**ROI** — see *Return on Investment*.

**SOA** — see *service-oriented architecture*.

**Sarbanes-Oxley Act** — An act passed by the U.S. Congress in 2002 to guard against fraudulent accounting activities by corporations. The act requires strict auditing of financial data and personal responsibility on the part of financial officers for failure to comply.
service — See Web service

service-oriented architecture (SOA) — An architecture for interoperability between autonomous systems built on Web service technologies.

SOX — See Sarbanes-Oxley Act.

TAFIM (Technical Architecture Framework for Information Management) — An architectural framework developed by the Department of Defense and officially discontinued in 2000.

time boxing — The rigorous use of typically short delivery cycles to reduce the temptation to constantly add new features to a system.

time to value — The delta between the approval of a project’s budget and when executives feel the project is having a positive impact on the organization’s business.

TOGAF (The Open Group Architectural Framework) 8.1 — An architectural framework that is controlled by The Open Group.

TTV — See Time-to-Value.

Web service — An autonomous software system that can respond to requests from other, independent software systems through a nonproprietary message format and transport protocol. Typically, the message format is SOAP.

workflow — The flow of work through an organization as a high level task is completed, such as processing an insurance claim.

Zachman Framework for enterprise architectures — An architectural framework in which an enterprise is modeled as thirty cells, each of which represents an intersection between a perspective and an abstraction.
References

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


Boy - For an good introduction to John Boyd’s work, see *Genghis John* in Proceedings of the U. S. Naval Institute July 1997, pp. 42-47 By Franklin C. Spinney available at www.d-n-i.net/fcs/comments/c199.htm


Cha - *Why Software Fails* In IEEE Spectrum Sept 2005 By: Robert N. Charette

Cli - Information Technology Management Reform Act of the One Hundred Fourth Congress of the United States at the Second Session.


Gov - This quote is available many places, for example, www.cio.gov/documents/FINAL_White_Paper_on_EA_v62.doc


GAO1 - GAO Report to the Secretary of the Treasury November 2004 FINANCIAL AUDIT IRS’s Fiscal Years 2004 and 2003 Financial Statements

GAO2: 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)

GAO3 - 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.

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


Roy - *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

