The IT Complexity Crisis: Danger and Opportunity

A White Paper by Roger Sessions

November 8, 2009

Table of Contents

In a crisis, be aware of the danger – but recognize the opportunity. - John. F. Kennedy

Introduction

Those of you who have been following my writing know that I have a soft spot for New Zealand. Among its many contributions to civilization is the best coffee in the world served as a concoction called a "flat white." But now I have another reason to love New Zealand. I see New Zealand as offering the world a glimmer of hope of escaping the coming meltdown of IT.

The coming meltdown of IT; the out of control proliferation of IT failure is a future reality from which no country – or enterprise - is immune. The same IT failures that are eroding profitability in the United States are impacting the economy in Australia. IT failures are rampant in the private sector, the public sector, and the not-for-profit sector. No place is safe. No industry is protected. No sector is immune. This is the danger, and it is real.

Surely this is a bit overdramatic?

I'll let you judge the drama of this statement. But the fact is that, worldwide, we are already losing over USD 500 billion *per month* on IT failure, and the problem is getting worse. If you find that figure unbelievable, I ask you to suspend judgment for a few pages. The numbers are estimates, of course. The precise numbers are not the point. The sheer magnitude of the numbers is what is important. As IT professionals we have a responsibility to understand how we can prevent the continuing spiral of failures that is burying us.

Okay, that's the bad news. I prefer, however, to view this, not as a catastrophe, but as an *opportunity*. After all, if we can reverse this trend, we will have an additional USD 500 billion per month to spend on things we care about: things like corporate profitability, improving the delivery of government services, increasing the effectiveness of non-profit organizations, and saving the environment. There is a lot of goodness we can buy for our world with 500 billion dollars per month.

I firmly believe this problem *is* solvable. Not only will the solution bring compelling financial rewards, it will bring many non-tangible rewards as well, such as:

- reducing the complexity of procurement
- helping small and mid-sized businesses compete on more projects
- increasing the agility of organizations
- making the workplace a more collaborative environment
- delivering IT systems on-time and within business expectations.

But we must understand a problem before we can solve it. I will present compelling evidence that the proliferation of IT failures is caused by increasing IT complexity. And this is good news, because complexity is a solvable problem. The solution is *simplicity*.

Simplicity is not so easy to achieve. It requires commitment, training, and process. Simplicity needs to start someplace. In the public sector, New Zealand is a good candidate. But this problem is not limited to the public sector. *Any enterprise is eligible*. Maybe it will start someplace else, perhaps with you, right here, right now.

Note on Calculations

This article is based on a large number of calculations and extrapolations. My readers generally have mixed feeling about math. Some don't care about math and just want to know the bottom line. Some don't believe the bottom line unless they see the math that led there. I have therefore chosen a compromise. When I refer to calculations, I will start by giving the bottom line. I will follow this with an insert on how I calculated the bottom line. If you care about the calculations, read the insert. If you don't, don't.

The Coming IT Meltdown

By all accounts, the financial crisis is a poster child for a meltdown. I haven't seen worldwide estimates for the cost of the financial crisis, but the crisis has cost the United States about USD 1.5 trillion. I use that as my benchmark for a meltdown. Does the current state of IT qualify by this definition?

Calculating the Cost of the Financial Crisis

In both 2008 and 2009, bailout packages were approved for the financial crisis in the United States [01]. In 2008 the bailout package cost USD 700 billion. In 2009 the bailout package cost USD 787 billion. Assuming that no further bailout packages are needed, the total cost of the financial meltdown to the United States taxpayer will be USD (700 + 787) billion = USD 1.49 trillion.

Today, IT failures are taking a heavy toll. Worldwide, the annual cost of IT failure is around USD 6.18 trillion (that's over USD 500 billion per month). If you are interested in how I calculate the cost of IT failures, see the insert *Calculating the Cost of IT Failure*. The annual cost to the US economy is around USD 1.22 trillion per year. New Zealand (remember New Zealand?) is spending USD 3.9 billion per year, quite a chunk of change for a country with a total annual Gross Domestic Product of USD 44 billion.

The United States is losing almost as much money per year to IT failure as it did to the financial meltdown. However the financial meltdown was presumably a one-time affair. The cost of IT failure is paid year after year, with no end in sight.

These numbers are bad enough, but the news gets worse. According to the 2009 U.S. Budget [02], the failure rate is increasing at the rate of around 15% per year. If this trend continues, within another five years or so a total IT meltdown may be unavoidable.

Okay. So there is plenty of bad news here. But there is also some good news. As I said, we have some incredible opportunities. If, say, the United States could solve the problem of IT failure, it could increase its GDP by over one trillion USD per year. A typical private company could increase its profitability by at least 25% with little risk and minimal investment. A public sector entity could improve its services by 25% without increasing taxes a penny. A non-profit organization could do 25% more good for the world. And 25% is only the beginning of future savings. That number could be much greater.

So while the risk of ignoring this problem is significant, the payoff for addressing this problem is huge.

Calculating the Cost of IT Failure

According to the World Technology and Services Alliance, countries spend, on average, 6.4% of the Gross Domestic Product (GDP) on Information Communications Technology, with 43% of this spent on hardware, software, and services. The other 57% is spent on communications technology. This means that, on average, 6.4 X .43 = 2.75 % of GDP is spent on hardware, software, and services. I will lump hardware, software, and services together under the banner of IT.

According to the 2009 U.S. Budget [02], 66% of all Federal IT dollars are invested in projects that are “at risk”. I assume this number is representative of the rest of the world.

A large number of these will eventually fail. I assume the failure rate of an “at risk” project is between 50% and 80%. For this analysis, I’ll use the average: 65%.

These numbers might be conservative. The Standish Group, in a widely quoted study on IT failure [03], estimates failure rates that appear to be higher. They estimate that projects that are late, over budget, failing to meet expectations or cancelled outright at 68% of all IT projects whereas I am calculating 43% (.65 X .66) of the total IT budget.

One requires caution when comparing our numbers, however, because The Standish Group is looking at the percentage of *projects* that will be challenged or cancelled, whereas I am looking at the percentage of the *IT budget* that will fail to deliver value. Looking at the percentage of IT projects (as does Standish) is less meaningful than looking at percentage of budgets (as I do). The reason for this is that many IT projects are small in scope and these projects are less likely to have problems. They contribute a lot to the "percentage of projects" that succeed, but relatively little to the "percentage of IT budget" invested in successful projects.

Had The Standish Group looked at IT budgets, it is likely their failure rate, already higher than mine, would have been higher yet.

The cost of a failed project is only the tip of the iceberg. Every project failure incurs both direct costs (the cost of the IT investment itself) and indirect costs (the lost opportunity costs). Determining an average for these indirect costs is difficult, but an example might help.

Between 1994 and 2005, the Internal Revenue Service spent $185 million on a new electronic fraud detection system. The project was abandoned in 2006 [04].

According to a report in 2008 [05] by the Treasury Inspector General for Tax Administration, the Federal Government lost approximately $894 million in fraudulent refunds during 2006 because the system was not operational.

So in this case, the direct cost of the failure was $185 million and the indirect cost (the lost opportunity costs) was $894 million. Of course, that $894 million was lost just in 2006. Presumably the same amount was lost in 2007 bringing the total up to $1.788 billion. And this doesn't include money lost beyond 2007, interest payments on the $1.788 billion, or the cost of people hired to try to do the work the system would have done had it been delivered. So the indirect costs of this $185 million failure far exceeded $1.788 billion, a ratio of well over 9.6 to 1.

When thinking about indirect costs, you need to include the costs of replacing the failed system, the disruption costs to your business, the lost revenue because of the failed system, the lost opportunity costs on what that lost revenue could have driven, the costs to your customers, lost market share, and on and on. And you need to consider these costs over the number of years the system would have been functional had it not failed (or at least the number of years until you can put another system in place.)

You can see that indirect costs add up quickly. I will assume that the ratio of indirect to direct costs is between 5:1 and 10:1. For this analysis, I'll take the average: 7.5:1.

To find the predicted cost of annual IT failure, we then multiply these numbers together: .0275 (fraction of GDP on IT) X .66 (fraction of IT at risk) X .65 (failure rate of at risk projects) X 7.5 (indirect costs) = .089. To predict the cost of IT failure on any country, multiply its GDP by .089.

The following table performs this calculation of various regions of the world.

Region	**GDP***	**Cost of Failure***
World	69,800.00	6,180.48
USA	13,840.00	1,225.47
New Zealand	44.00	3.90
UK	2,260.00	200.11
Texas	1,250.00	110.68

*USD Billions

Table 1. Predicted Annual Cost of IT Failure

Of course, these calculations are estimates. I recommend you don't get overly focused on the exact amounts. I could be off by ten or twenty percent in either direction. The real point is not the exact numbers, but *the magnitude of the numbers and the fact that the numbers are getting worse.*

Cause of Failure

Is there a primary cause of these IT failures? If so, what is it? An important clue is the rising number of failures. According to the U.S. budget, the number of "at-risk" IT projects has been steadily climbing over the last three years, from 30% in 2007 to 43% in 2008 to 66% in 2009. Whatever it is that is causing the problem must therefore also be climbing at a comparable rate.

Some have suggested the culprit is poor communications between business and IT. While poor communications is without a doubt a contributing factor, there is no evidence that business and IT have gotten any worse at communicating between 2007 and 2009.

Some have suggested the culprit is increased functionality. There are numerous arguments against functionality being the problem. And there is no evidence that functionality has increased at the same rate as IT failures have increased.

The almost certain culprit is complexity. Actually, complexity is indirectly related to functionality, in that a 25% increase in functionality increases complexity by 100% (Glass's Law, see [06]). However complexity is even more impacted by system organization. So complexity goes up as functionality increases, goes down as functionality is partitioned, but then goes up again as connections are made between systems. The overall complexity of a system is a delicate balance between all three of these factors.

Complexity seems to track nicely to system failure. The more complex a system is, the harder and more costly it is to work on that system. And while complexity can correlate with functionality, there are many examples of highly functional systems that are organized much simpler than other systems with much less functionality. Empirically, we have all experienced that the difficulty of maintaining a system is much more related to how functions are organized than to the number of functions.

It is actually possible to measure the complexity of an IT system. We only have anecdotal evidence so far, but I strongly believe that when we plot complexity against failure rates, we will see an almost perfectly linear relationship.

The exact formula for measuring the complexity of a system is given by what I call Sessions's Summation of Bird's Formulation of Glass's Law (a mouthful, no?) For those who would like to understand how this equation works, see the *Measuring IT Complexity* insert.

Once we understand how complex some of our systems are, we understand why they have such high failure rates. We are not good at designing highly complex systems. That is the bad news. But we are very good at architecting simple systems. So all we need is a process for making the systems simple in the first place.

Measuring IT Complexity

Measuring IT complexity is easier than you might think. It all starts with Glass's Law [06], that for every 25% increase in the complexity of the problem space, there is a 100% increase in the complexity of the solution space. In IT systems, there are two contributors to the complexity of the problem space. The first is the number of business functions in the system. The second is the number of connections that system has to other systems.

We need to start with a standard complexity unit (SCU). I'll define one SCU as the amount of complexity that a system has which contains only one business function and no connections to other systems. Based on Glass's Law, this is the least complex system possible.

Okay, we start with a system with one business function and no connections. By definition, this has 1 SCU. Now let's start adding more business functions into the system. As we add more functions into the system, the number of SCUs goes up a predictable amount. This amount is calculated using Bird's Formula. Bird is Chris Bird, who showed me how to rewrite Glass's Law as a mathematical formula.

Let's say a system S has bf number of business functions, and no connections to other systems. Then the number of SCUs in that system is given by

S = 10 raised to the power of ((log(2)/log(1.25) X log (bf))

The log(2) and the log(1.25) are both constants, and can thus be combined, giving

S = 10 raised to the power of (3.1 X log(bf))

or, more simply,

$$S = 10^{3.1 \log(bf)}$$

Similarly, we can calculate the complexity of a system with one business function and cn connections to other systems. Note that I am simplifying the analysis by assuming that a new connection adds about the same amount of complexity as a new business function, which follows my experience. If you don't agree, it is easy enough to modify the equation accordingly. But given my assumption, the complexity because of connections is as follows:

$$S = 10^{3.1 \log(cn)}$$

Most systems have both multiple business functions and multiple connections, so we need to add the two together:

$$S = 10^{3.1 \log(bf)} + 10^{3.1 \log(cn)}$$

Most systems are not made of a single system, but a number of smaller systems, each of which has a complexity described by the above equations. An SOA, for example, would have multiple services, each of which has a complexity rating.

If we assume that our system is an SOA, then the complexity of the SOA as a whole is the summation of the complexity of each of the individual services. This is expressed as what I describe as Sessions's Summation of Bird's Formulation of Glass's Law. It looks like this:

Sessions's Summation for SOA Complexity

(Sessions's Summation of Bird's Formulation of Glass's Law.)

An SOA with m services has a total complexity (in SCUs) of

$$\sum_{i=1}^{m} 10^{3.10 \log(bf_i)} + 10^{3.10 \log(cn_i)}$$

Where the bf_i is the number of business functions in the ith service and cn_i is the number of connections in the ith service.

Now while Sessions's Summation looks rather ugly, it is in fact easily calculated using a straightforward spreadsheet.

Sessions's Summation gives us an easy way to compare two different architectures with respect to their complexity. Just plug both into Sessions's Summation and read the resulting number. That number is the complexity in SCUs of the architecture. If one architecture has a total SCU of 1,000 and another a total SCU of 500, then the first is twice as complex as the second. It is also twice as likely to fail.

Notice that while functionality is a factor in complexity, it is not the major factor. Much larger factors are how many services there are, how many functions there are in each of the services and how those services are connected to each other.

Designing Simpler IT Systems

Let's take a moment to review. So far, I have made the following points:

- IT Failure is a major problem (or opportunity, depending on your perspective).
- IT Failure is most likely a function of complexity.
- We know how to measure complexity.

You might think that this is the end of the story. All we need to do is measure the complexity of an architecture before we build it, and if it is complex, do something else.

But being able to measure the complexity of an architecture gets us only part of the way. *Our goal should be to design the least complex architecture possible that solves the business problem*. This is our only hope of controlling complexity. Being able to measure the complexity of an architecture does not tell us whether there is a simpler architecture right around the corner.

Suppose, for example, that we designed an architecture that measured 10,000 Standard Complexity Units (SCUs). Is that the best we can do? What if there is another architectural solution that has only 5,000 SCUs? That second solution would have half the complexity, cost half as much to build, and be half as likely to fail. Does it exist? And if it does, how do we find it?

We need a process that guides us to the simplest possible architecture from the beginning. This means finding the best possible balance between the number of functions in systems and the number of connections between systems.

The process that does this is called Simple Iterative Partitions (SIP). SIP is a pre-design process that partitions business functions into sub-systems such that the overall system has the least possible complexity needed to solve the business problem.

I won't describe the SIP process in detail here, since I have done that in numerous other places, most notably, my recent book [07] and White Papers [08]. I'll just note that it is a five-phase iterative process as shown in Figure 1 and that most of what we think of as architecture and implementation fits in the final phase.

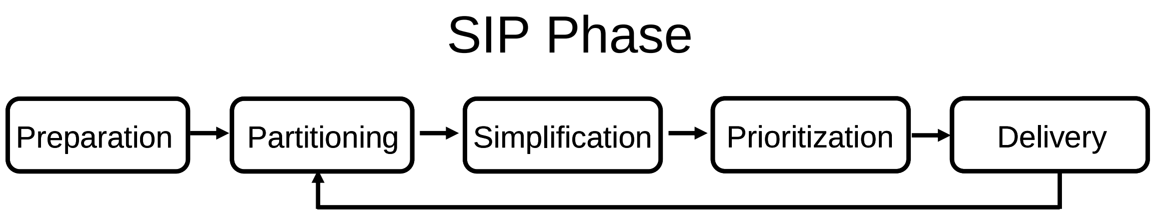

Figure 1. The Five Phases of SIP

Briefly, SIP works as follows. First, a system is decomposed into a collection of low-level business functions. These business functions are then analyzed and placed in subsets based on synergistic relationships. These subsets are then treated as sub-systems and/or sub-projects. In a service-oriented architecture, these become the definitions for services.

The SIP-driven simplification can be dramatic. This is easiest to see in an example.

Consider an inter-library loan (ILL) system. An ILL system allows libraries to borrow books from each other. If I go into my local library and they don't have my book but another library does, my library can borrow the book on my behalf from the other library.

I often run this exercise in my SIP workshops. Every group that has been assigned this exercise comes up with some variant on what I will call the standard architecture.

The standard architecture has three services: Borrowing-Library, Inter-Library-Loan-Service, and Lending-Library. These services, their methods, and the messages between the services is shown in Figure 2.

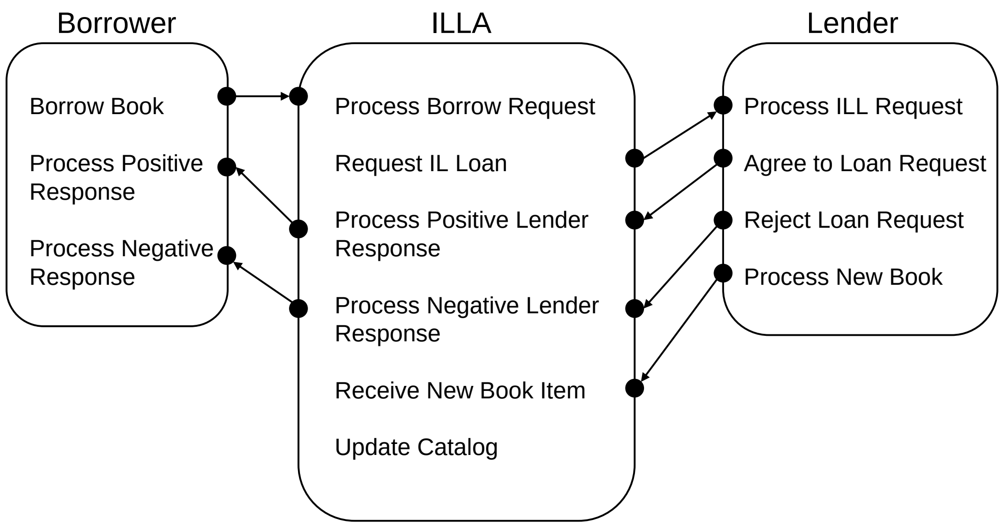

Figure 2. Standard Architecture

If we plug the standard architecture into the formula for SOA complexity, we get a reading of 891.

Calculating the Complexity of the Standard Architecture

The Borrower service has 3 business functions which contribute 30 SCUs and 3 connections which contribute 30 SCUs.

The ILLA service has 6 business functions which contribute 261 SCUs and 7 connections which contribute 422 SCUs.

The Lender service has 4 business functions which contribute 74 SCUs and 4 connections which contribute 74 SCUs.

The total number of SCUs for the architecture is 30 + 30 + 261 + 422 + 74 + 74 = 891.

Now let's take a look at the SIP generated architecture. For those of you interested in getting an idea of the SIP process, read the insert. If you just want to see the end result, skip the insert.

The SIP Process for the Interlibrary Loan System

The SIP process starts with a decomposition of a system into atomic business functions. An atomic business function is the lowest level of functionality in a system that is still recognizable to the business.

The first level decomposition of the Interlibrary Loan System would look like:

Interlibrary-Loan

- Borrower
- Lender

The decomposition continues:
Interlibrary-Loan

- Borrower
    - Request-Book
    - Process-Positive-Response
    - Process-Negative-Response
- Lender
    - Process-Borrow-Request
    - Agree-To-Loan-Request
    - Reject-Loan-Request
    - Process-New-Book
    - Update-Catalog

If we continue the decomposition we start getting into implementation details. We don't want to do that (it is way too early for implementation). So we stop the decomposition.

We now have a collection of eight atomic business functions as follows:

- Request-Book
- Process-Positive-Response
- Process-Negative-Response
- Process-Borrow-Request
- Agree-To-Loan-Request
- Reject-Loan-Request
- Process-New-Book
- Update-Catalog

Next we partition this collection into subsets based on the relationship called *synergistic*. Two atomic business functions A and B are considered to be synergistic if, from the business's perspective, A is not useful without B and vice versa.

So, for example, Request-Book is synergistic with Process-Positive-Response. It doesn't make sense to request a book unless one can process an agreement to loan the book.

Along the same lines, Request-Book and Agree-to-Loan-Request are not synergistic. From the business perspective, borrowing books could be useful even if one cannot loan books. And Update-Catalog is not synergistic with anything else.

Once we have completed the synergistic analysis, we have the following subsets:

Subset 1
- Request-Book
- Process-Positive-Response
- Process-Negative-Response

Subset 2
- Process-Borrow-Request
- Agree-To-Loan-Request
- Reject-Loan-Request
- Process-New-Book

Subset 3
- Update-Catalog

In SIP, every one of these subsets is autonomous with respect to the other. That means, among other things, that data is not shared between subsets and that any activity coordination must occur through work requests.

Based on the coordination through work requests, we notice a few missing pieces. If Subset 2 does Process-New-Book and Subset 3 does Update-Catalog, then Subset 2 must let Subset 3 know about the new book. Similarly, if Subset 3 does Update-Catalog and Subset 1 does Request-Book, Subset 3 must send the catalog over to Subset 1. Once we fill in these missing pieces, we get these updated subsets:

Subset 1
- Request-Book
- Process-Positive-Response

- Process-Negative-Response
- Receive-Catalog

Subset 2

- Process-Borrow-Request
- Agree-To-Loan-Request
- Reject-Loan-Request
- Process-New-Book

Subset 3

- Update-Catalog
- Receive-New-Book-Item

We might as well name the subsets. The obvious names of Subsets 1, 2, and 3 are Borrower, Lender, and ILLA (InterLibrary Loan Agency)

Two errors are frequently made during the SIP process. The first is ending up with atomic business functions that aren't really business functions but are more like implementation details. The second is misunderstanding the concept of synergistic placement.

Either of these errors can have serious ramifications as to the final complexity of the architecture. In SIP workshops, we spend considerable time on these two concepts.

There are three advantages to strictly following this process.

The first advantage is that the resulting architecture is the least complex architecture possible that solves the business problem. Remember that complexity is decreased by reducing functionality and increased by adding connections. The decomposition results in smaller and smaller packets of functionality. The synergistic placement results in the minimal number of connections between functions. The combination of decomposition and synergistic placement gives us the ideal balance.

The second advantage is that if the process is followed correctly, *there is only one theoretically possible outcome*. Two different SIP-trained architects should end up with similar results. This greatly simplifies confirming results.

The reason a given SIP analysis has only one theoretical outcome is that the synergistic function is in a mathematical family of functions called equivalence relations. The mathematics of partitions guarantees that, for a specific collection of items and a specific equivalence relation, there is one and only one partition (outcome).

The third advantage is that the resulting architecture can be validated. We can check the synergies in the subsets and verify that they have been assigned correctly.

So in theory, this process guarantees that, for a given business problem, we will find the unique partition of functionality that results in the least possible complexity and we can verify that we have done this correctly. In practice, this guarantee is highly dependent on how well the SIP facilitator understands the process and how willing the SIP team members are to work together in a collaborative manner.

Process is a means, not an ends. The goal is delivering higher quality IT systems faster and at lower cost. To do this, we need to ensure that our systems are architected as simply as possible while still meeting the business problem. This is the value of SIP, not the process, but what the process delivers.

The SIP generated architecture is shown in Figure 3.

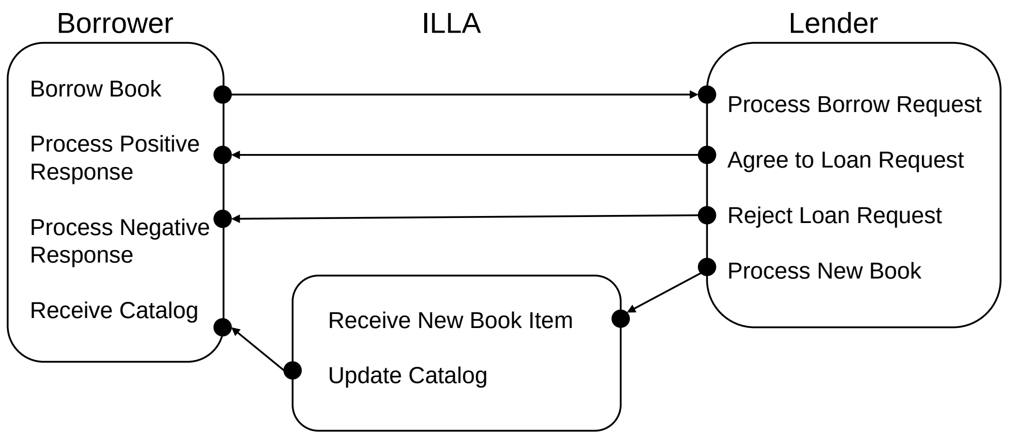

Figure 3. SIP Generated Architecture

If we plug the SIP generated architecture into the formula for complexity, we get 314 SCUs.

Calculating the Complexity of the SIP Architecture

The Borrower service has 4 business functions which contribute 74 SCUs and 4 connections which contribute 74 SCUs.

The ILLA service has 2 business functions which contribute 9 SCUs and 2 connections which contribute 9 SCUs.

The Lender service has 4 business functions which contribute 74 SCUs and 4 connections which contribute 74 SCUs.

The total number of SCUs for the architecture is 74 + 74 + 9 + 9 + 74 + 74 = 314.

Now we can ask the question, which architecture is better: the architecture designed by the vast majority of architects or the architecture created when those same architects follow the SIP process? Just from looking at the two architectures, it seems obvious that the SIP architecture is simpler. The SCUs quantify the difference. The standard architecture has 891 SCUs. The SIP architecture has 314 SCUs.

The complexity readings tell us that the SIP architecture has 35% of the complexity of the standard solution. Assuming that SCUs correlate linearly with cost (which I believe they do) the SIP architecture will cost 35% of what the standard architecture would cost.

We can take at least a high level look at this claim.

In the standard architecture, the ILLA service is involved with every loan request. If the ILLA service is down, the whole system is down. This means that we need to run ILLA on highly reliable hardware. The ILLA system also is going to be challenged as the user population increases. This means that we need to run ILLA on some type of cluster architecture. All of this means expensive hardware, complex software implementations, and specialized programmers.

In the SIP architecture, the ILLA service is only involved in receiving new books into the system and creating the catalog. This happens infrequently. Loan requests are directly processed between the borrowing and lending libraries. If the ILLA service is down, the system as a whole continues working fine. This ILLA service can be implemented on an inexpensive PC by an entry level programmer in a few weeks.

Is the SIP architecture going to be exactly 35% of the cost of the standard architecture? It is hard to say. But it is clearly going to be much cheaper to implement and run while at the same time being more reliable, scalable, and most of the other desirable qualities that we associate with good architecture.

Some might argue that they could have come up with the SIP solution without SIP. After all, they might say, isn't this a standard peer-to-peer pattern?

It is, indeed, possible that some architects might have come up with this solution. All I can say for sure is that in my three years of conducting workshops on complexity with highly experienced architects, none have yet.

And probability is not on their side. We have at least ten business functions that need to be split up into services. There are 21,147 ways this could be done. Of these, 21,146 are wrong in the sense that they are more complex than necessary. The chance that an architect will come up with the one optimal solution, even with the benefit of years of experience, is quite small. And, as I said, it hasn't happened yet.

Simplifying Procurement

One additional benefit in using SIP is that you end up with self-contained subsystems. In SIP terminology, these are called autonomous business capabilities (ABCs). Each of these ABCs is a system that has the maximum autonomy possible with respect to the business functions in other ABCs.

This means that each of these can be treated as an independent effort. They can be architected, implemented, and delivered independently of each other. Because each contains a collection of closely related business functions, each delivers useful business value in its own right. And because the ABCs interoperate in well-defined ways, each drops into the larger system relatively easily.

For those considering SOAs, the relationship between ABCs, an architectural concept, and services, as an implementation concept, should be obvious.

This opens up a new possibility for procurement in an organization that outsources part or all of its work. Typically a new system is bid as a large, monolithic, highly complex, very expensive system. Very few vendors are capable of delivering such systems. The vast majority are not even capable of responding to the bid request.

SIP opens up another approach: rather than bidding out one system, bidding out a number of smaller, well defined, simpler, relatively inexpensive systems. Many smaller and midsized vendors *can* compete for these bids.

This gives you greater control of your solution. You minimize your dependency on one large vendor. And your partitioned solution allows you to more effectively choose the right vendor for the right functionality thus playing to the strengths of the individual vendor.

Take the InterLibrary Loan System as an example. Of course this example is far smaller than the large systems I am discussing, but it serves to illustrate the concept of partitioned procurement.

In the standard procurement process we would create a laundry list of the functions needed. For the InterLibrary Loan System, this number is only ten, but a real life system will have hundreds, perhaps thousands of functions. Because of the complexity of the real-world systems, it could cost at least USD 100,000 just to respond to a large system bid. This limits the field to a small handful of vendors. They will be bidding on a highly complex system, which means they will be bidding high. And whoever wins has control of your system.

In the SIP based procurement process, the SIP analysis would be completed before the bidding was started (or as a preliminary project). In the case of the InterLibrary Loan System, for example, we would have identified three systems, not one. Each of those systems would be relatively simple. And each could be bid out independently of the others.

Say it costs $100,000 to deliver a SCU. The total cost of the system is expected to be around $891,000 (891 SCUs X $100,000/SCU). Rather than bid out a single system with 891 SCUs at an expected cost of $891,000, we bid out three smaller systems. The first (Borrower) has an expected cost of $148,000. The second (Inter-Library Loan) has an expected cost of $17,000. The third (Lender) has an expected cost of $148,000. The total cost of the SIP architecture is $314,000 for a total savings of $578,000.

But perhaps just as important as the huge potential cost savings is the opportunity to include small and medium-sized organizations in the bidding process. In existing procurement processes, these organizations frequently get just the crumbs that fall from the table of the large consulting organizations. In a SIP process, they can sit at the table as equal partners.

Quick Review

Let's take a moment for review. I have described the magnitude of the IT Failure problem (I believe I used the term "meltdown"). I have identified the cause of this problem: unmanaged complexity. I have shown how complexity can be measured. I have shown how these measurements allow architectures to be compared for relative complexity. I have described a process that leads organizations to finding the simplest possible architecture for a given business problem. I have shown how this process will reduce complexity, and thus failure rates and cost. And I have shown how a procurement process based on simplification is good for both the client and the service providers.

At this point, haven't the world's problems been solved? Unfortunately, no. we have a few impediments we must address.

Impediments to Simplicity

So what are the impediments to attacking complexity?

At this point, I am on shakier ground. I feel comfortable in the realm of logic, numbers, and process. I feel less comfortable in the realm of psychology, group dynamics, and organizational politics. Yet it is in this latter realm that most of the impediments lay.

When people choose not to follow my recommendations, it is usually because of one of three reasons. I'll cover each in turn.

Reason 1. Fear of New Process

Some believe that this process is new and therefore risky. In fact, this process is not new and the risk factor is very low.

This process is more than two years old and has been used successfully by a number of organizations. The cost of the SIP analysis is negligible compared to the cost of the whole project, less than 5%. The SIP analysis is almost entirely completed before the usual process begins. In the unlikely event that one is not satisfied with the analysis, the process will at least bring a comprehensive understanding of the problem being addressed, its needs, obstacles and potential solution paths. The much more likely outcome is that you will reduce the overall project cost dramatically while simultaneously improving the chances of overall project success.

For the InterLibrary Loan example, our cost is less than $44,000 for the SIP analysis (5% of original cost of $891,000) and the potential benefit is $578,000 (the difference in cost between the original architecture and the SIP architecture) plus numerous non-financial benefits. Sounds like good odds to me. And in performing the SIP analysis, you will better understand the business process and needed solution prior to implementation.

In any case, we know that what we are doing now is not working. What little perceived risk there might be in trying something new is negligible compared to the risk of doing things the same way we always have which, unfortunately, continues to cost us billions of dollars in project failures

Reason 2. Fear of managing multiple vendors

Some believe that the cost of managing multiple vendors will more than make up for the cost savings. This assumes that the cost of managing vendors is linear with respect to the number of vendors. This is patently false.

The cost of managing a given vendor is linear with respect to the complexity of the project that vendor is delivering. It is much easier (and cheaper) to manage three vendors delivering three simple projects than it is to manage one vendor delivering one large, complex project. There are numerous examples of failed complex projects that involved only a single vendor. In fact, this is the norm; the large majority of big projects have been bid out as single-solution projects, and many of these ultimately failed.

The use of multiple vendors also gives you, the project owner, more control. You award the projects based on the vendor's strengths in that one area and you avoid being held hostage to one large vendor that may be strong in one area yet weak in other areas.

Reason 3. Political Realities

Some believe that the political realities within their enterprise oppose efforts at simplification. In some cases, they are right.

SIP requires a close cooperation across business groups. It assumes that business functionality will be partitioned based on complexity impact, not based on which division has the most political sway. It also requires a collaborative relationship between the IT group and the business groups. In many organizations, this relationship is weak at best.

Because of the cross-organization cooperation needed, success is unlikely unless the process is endorsed by senior management.

The belief by your executives that simplicity is not attainable is a self-fulfilling prophesy. That's the bad news. The good news is that the belief by your executives that simplicity is attainable is also a self-fulfilling prophesy.

Introducing Simplicity Into Your Organization

In my experience, the best way to introduce SIP and a new culture of simplicity to an organization is to start out small. Well, not too small. Say your next million dollar project. Convince your executives that this approach has tremendous promise and suggest that they try SIP on a relatively self contained project. Try to find one that is comparable to other projects that have been done so that you can document the positive impact in each of these areas:

- Reduced project cost
- Increased satisfaction with the end result
- Increased maintainability, agility, and scalability
- Improved morale
- Improved relationships between business and IT
- Better accuracy in predicting project deliveries

Once you have identified a candidate project, lay your groundwork. Learn about the project. Understand the timeline, the issues, and the business reasons for supporting the project.

Next, elicit the support of the project owner. Explain the problems with complexity, Describe how complexity overwhelmed past projects and how SIP might help avoid their fate. *Discuss the benefits of SIP in terms of business benefit.*

Ask the project owner to help you create a business case for SIP. Create a list of candidate business functions. Show a standard architectural approach to partitioning these functions. Run the complexity analysis. Run a quick and dirty SIP partitioning exercise and give some estimates as to how much complexity might be reduced.

Give your business case in terms of cost/benefit. The full SIP analysis will cost X and generate Y direct savings in reduced complexity and Z indirect savings as well as improved chances of project success.

Step out of your current role and think like an executive. Evaluate all the issues that are connected to this project. Create a compelling argument about how SIP will help.
Sell your boss on the solution and begin selling up the chain (with his/her help).

Once you are successful selling SIP, make sure you can deliver. Be aware of the pitfalls. Bring in whatever expertise you need. Remember, if you fail on the first project, you are unlikely to ever get another chance.

Once you have documented SIP's benefits on a smaller project, you can suggest that SIP be tried on a larger scale. Once SIP has proven itself on two significant projects, it will take on a life of its own. *Nobody* will want to go back to the old way of creating IT systems.

Call to Action

Okay, so now you know it all. If we are ready to save the world through simplification, where do we start?

If I was to seek the ideal country to embrace simplicity, here are some of the characteristics I would seek:

- The country needs to be big enough to support some seriously large IT projects. Projects like the merging of cities, with their systems and business processes. Projects like the re-architecture and delivery of more efficient and patient-centered healthcare systems. Projects that look at innovation in the support of large export-facing organizations. Projects that are looking at supporting institutions that are focused on ensuring profits are kept in the country and reinvested in the country.
- The country needs to have a healthy preference for local expertise and knowledge over advice coming in from outside the country, which most of the times does not scale to the local reality without some serious rethinking.
- The country needs to have a strong enough economy to support a healthy ecosystem of small and medium sized consulting organizations.
- The people in the country need to take pride in showing the rest of the world that they can lead the way.

- The country needs to have excellent education opportunities in the various fields related to IT.
- The educational institutions need to be willing to incorporate training into the importance of complexity control into their curriculums.
- And it goes without saying, the country must have truly outstanding coffee.

All of which lead me back to New Zealand. New Zealand scores on each of these points. And of all the countries and states I have visited, I have found some of my most receptive audiences in the beautiful island down under. And their coffee is to die for.

But of course, the campaign against complexity has barely begun. It's an open field and it's anybody's game. Perhaps the next success stories will not come from New Zealand. Perhaps they will come from your country and your organization. Are you ready to embrace simplicity? How do YOUR coffee shops stack up?

References

[01] http://topics.nytimes.com/top/reference/timestopics/subjects/c/credit_crisis

[02] Budget of the United States Government, Fiscal Year 2009, Analytical Perspective, published by the Office of the President of the United States, page 169.

[03] The Standish Report is not available for free, but a good overview of the report is available at http://www.pmhut.com/the-chaos-report-2009-on-it-project-failure

[04] GAO Testimony before the Subcommittee on Federal Financial Management, Government Information, Federal Services, and International Security, Committee on Homeland Security and Governmental Affairs, U.S. Senate, July 31, 2008.

[05] The Electronic Fraud Detection System Redesign Failure Resulted in Fraudulent Returns and Refunds Not Being Identified, Office of the Treasury Inspector General for Tax Administration, August 9, 2006, Reference Number: 2006-20-108.

[06] Glass's Law comes from Robert Glass's Fact and Fallacies About Software Engineering. He did not discover the law. He actually described it from a paper by Scott Woodfield, but Glass did more than anybody to publicize the law.

[07] Simple Architectures for Complex Enterprises by Roger Sessions

[08] The SIP White Papers are available at http://www.objectwatch.com/white_papers.htm#SIP

About Roger Sessions

Roger Sessions, a tireless advocate for efficiency through simplicity, is the CTO of ObjectWatch, a company he founded thirteen years ago. He has written seven books including his most recent, *Simple Architectures for Complex Enterprises*, and dozens of articles. He assists both public and private sector organizations in reducing IT complexity by blending existing architectural methodologies and SIP. In addition, Sessions provides architectural reviews and analysis. Sessions holds multiple patents in software and architectural methodology. He is a Fellow of the International Association of Software Architects (IASA), Editor-in-Chief of the IASA Perspectives Journal, and a Microsoft recognized MVP in Enterprise Architecture. A frequent keynote speaker, Sessions has presented in countries around the world on the topics of IT Complexity and Enterprise Architecture. Sessions has a Masters Degree in Computer Science from the University of Pennsylvania. He lives in Chappell Hill, Texas.

Roger loves feedback (and a good, stirring debate or two!) Join Roger in his crusade against complexity. His blog is SimpleArchitectures.blogspot.com and his Twitter ID is @RSessions.

For more information

For more information on how SIP can help you manage the complexity of your IT projects, write information@objectwatch.com.

Legal Notices

SIP Training

Both public and company-specific training available. Write information@objectwatch.com for details.