

Trond Frantzen's

Rapid Agile

Business System Analysis

The PowerStart Approach – How to Complete Business System
Requirements Faster than Ever Before!

First Words

So, why do software projects have so many problems?

"Without knowing anything at all about your current project, I'll bet even money that you'll be late. After all, well over half of all projects deliver late or deliver less than was promised by the original deadline. It's far worse when a project is on an admittedly aggressive schedule. Project people seem disconcerted when I proclaim that I'm willing to bet against them. They try so hard to believe that they'll buck the odds. What usually happens is that everyone agrees that the deadline is very tight; everyone works very hard; and then, when people see that they won't make it, they are shocked, disappointed, and deeply dismayed."

Tim Lister in **Waltzing with Bears**

For many projects software acquisition, development and installation simply takes too long, or it doesn't result in what the business needs.

In **Waltzing with Bears: Managing Risk on Software Projects**¹, Tom DeMarco and Tim Lister identify the five primary reasons why software projects have so many problems.

1. **Schedule Flaws:** Either an error in the original schedule or an error in the way the project is run can affect its timing.
2. **Requirements Inflation:** This happens when what is needed changes during development.
3. **Staff Turnover:** When key people leave during a project, it can have a serious impact on continuity and schedule.
4. **Specification Breakdown:** Anything less than complete agreement on specifications can be fatal.
5. **Underperformance:** Substandard work by anyone on the team will affect project quality.

¹ Tom DeMarco & Tim Lister, **Waltzing with Bears: Managing Risk on Software Projects** (ISBN: 0932633609), Dorset House, New York, NY, 2003.

This book isn't about project management; it's about a new way to conduct business system analysis and how to uncover the requirements completely, accurately and quickly – right up front.

Therefore, we're not going to talk about flaws to a project's schedule (DeMarco's and Lister's item # 1). Nor are we going to discuss their item # 3 (Staff Turnover), since that's a different kind of problem than requirements analysis. And finally, we're not going to touch Underperformance (item # 5), since we would rather discuss a method that enables all team members to soar with eagles.

That leaves **Requirements Inflation** (item # 2) and **Specification Breakdown** (# 4).

Requirements Inflation, also known as "scope creep", is everywhere. It happens on virtually every project. It seems that we no sooner have agreement on a project's scope, and then it changes. The business community can't seem to make up its mind. They always want to change things. Every project seems to be a "moving target". To eliminate scope creep or requirements inflation we need a better way to determine the requirements of a project. This book is about that way.

Specification Breakdown (anything less than complete agreement on the requirements can be fatal to a project) is the other area of concern. Most project requirements are very technical (being based on the solution) or are so vague that the business experts simply can't understand the specifications well enough to agree with them. This lack of agreement leads to what appears to be new requirements, resulting in what many see as scope creep. The general nature of most specifications (should they be called "generalizations"?), and the fact that most specifications are based on a predefined solution to the problem (which is not yet clearly defined) leads invariably to disagreement, scope creep and disillusionment with the technical team (who are usually the ones responsible for creating the requirements spec in the first place). And if the business folks created the requirements spec, it has the exact same problems (although not based quite as much on the technical solution) and lack of precision as that produced by anyone else.

To eliminate the infinite catch-22 of revisiting, redefining, revising, and respecifying system requirements – the endless loop between specification breakdown and requirements inflation – there must be a better way. This book is about that better way.

What this rapid requirements analysis approach is all about

This requirements analysis method will eliminate requirements inflation (scope creep) and the specification breakdown. It will help you to produce the Business Requirements Document the very first time, right up front, without having to revisit and reiterate the requirements. And, it will give you a way of getting proof that your requirement specification is complete. In other words, it will help you produce, very quickly, the single-iteration Business Requirements Document.

This requirements analysis method is for everyone. Yes, everyone. Whether you are a professional with twenty years of experience, or a new business system analyst, or even a certified software engineer – and whether your projects are short two-week efforts or multi-million dollar business projects – this agile system requirements analysis and specification method is for you.

This book is about **agile** business system analysis; and how to create a modern, useful system specification *really fast*. It's about how to uncover, specify, and integrate information technology with the business requirements of today and beyond.

It's also about how to find the exact, specific questions to ask your business experts (users), and a way of doing the highest quality work in the fastest way possible, without chaos.

And it's about how you can be the kind of extraordinary communicator and professional needed in the 21st century.

It's about business-based requirements analysis suitable to building the complex integrated systems for the 21st century and beyond.

It's about you and how to specify system requirements successfully. And fast.

This is **not** a book on program design, programming, web page design or database design (although it's a prerequisite for all of these). It's not even about programming languages or other software tools. It's not a silver bullet nor is it a magic wand to wave over troubled legacy systems. But most importantly, it's not more of the **Same Old Stuff**.

This is a book about getting system requirements specified fast and right the very first time.

In no other area of software development has there been as much misinformation as in the area of system analysis. Many useless development practices have been relentlessly hyped as "rapid development practices", "light analysis methods", "agile" methods and even "best practices". These labels, being far from the truth, have made many professionals cynical about virtually all analysis methods. While some methods are useful, many have been hyped so far beyond their actual capabilities that they have contributed to this general cynicism. Many training and consulting firms want to convince you that their new silver bullet will be **the answer** to getting a requirements document done in an impossible timeframe.

"So, is this more of the **Same Old Stuff**," you may ask, "but with different packaging?" Well, find out for yourself. Try the method described in this book – and you tell me.

I know there will be some who seek instant answers and instant tools. But the only tools that count in the area of system requirements analysis is your ability to ask the right questions of your business experts (users), and to properly document their responses. This book will show you how – with precision and accuracy.

Productivity and Analysis

The focus of the analysis method you are about to learn about are the words **fast**, **complete** and **accurate**. These are key words we will revisit many times in many different ways. In the world of system development they translate into "**productivity**" and "**quality**".

So, what exactly do we mean by **analysis**? The web site YourDictionary.com tells us analysis is, "*The separation of an intellectual or material whole into its constituent parts for individual study*" and "*The study of such constituent parts and their interrelationships in making up a whole.*" The Oxford English Dictionary tells us, more simply and directly, it is "... *the separation of something into its constituent elements.*"

Yet any system, by its very nature, is a synthesis of different elements **all brought together**. A database can be a system, and so can a garden hose. The importance of the analysis that we conduct is that it reduces the complexity of any system to bite-size pieces that can then be examined individually. In other words, we can minimize complexity by partitioning the effort.

The approach to system analysis that you are reading about – which I call "**The PowerStart Approach**" – provides a method by which any system can be analyzed rapidly – reduced to the simplest levels of information – and then put together again in a good system solution (design), faster than ever before possible.

But why is analysis so important? After all, there is a whole school of thought (yes, sadly, still thriving) which claims that we simply spend too much time on analysis already, “... so let’s just get on with the real work of implementing the system.” There’s another school of Luddites that are being pulled into the 21st century against their will, shouting and screaming, “... complete requirements analysis is impossible and can’t be done, so let’s just fuhgedaboutit and get on and implement something!”

The value of careful and complete analysis, I believe, is that it enables us to know where we are going and how complex the going will be. In other words, analysis enables us to create a clear and complete roadmap of the effort required to achieve our business goals. Accordingly, the analysis stage provides the foundation for all subsequent work. A lot of costly backtracking can be avoided in the later stages if analysis is done in a planned, logical and scientific manner.

But it has to be fast, or it will be seen as a waste of time.

The Human Factor

System analysis is not exclusively in the domain of computer-based systems. While there are systems everywhere we look today, not all of them are computer-based. We must try to not fall into the trap of thinking that all systems boil down into microchips and database accesses, because we’ll miss one of the critical points of system analysis: The human factor.

Not that long ago, all systems were human-based. Remember the Charles Dickens character of Bob Cratchit, who slaved over his narrow desk and warmed his hands over a candle? While working conditions have improved somewhat, a lot of information handling is still done manually.

The fact remains that people and machines must work in harmony to achieve the very best business results. Computers provide the efficiency, the speed, the lookup capability, the presentation facility, and liberate people from the mindless pursuit of technical facts. People, on the other hand, provide the synthesis, the multi-dimensional interface skills, the judgment factor, and the ability to make decisions with hypothetical data.

The information technology (IT) professional in today’s world of integrated systems must analyze and design the full assimilation of people and computers to meet the business needs of the client. And they must do this in a way so that the users – the business experts – and all others can understand the resulting specification. In other words, the architecture and blueprints must be (a) exactly what the client needs; (b) clearly understandable to the client and others who must use it; and, (c) meet the budget requirements of the client.

The “PowerStart” Approach to rapid business system analysis that’s detailed in this book will enable you to do exactly that – without ever alienating the client or user.

The Different Business Environments

You have no doubt noticed that each business environment appears to be different. In reality they are not. Businesses differ very little.

At first glance, a chain of retail shoe stores may not seem to have anything in common with an insurance company; but under the surface they still share the basic activities of accounts payable, accounts receivable, sales and customer records. Their culture and how they do things may be different, but the underlying business functions are the same.

The methods underlying the “PowerStart” Approach to requirements analysis described in this book are applicable to any business, therefore it doesn’t matter what type of particular business or application is being analyzed. The same methods can be applied to retailing, distribution, manufacturing, financial services, call centers, healthcare or just about anything else.

Finding the Questions to Ask

“Good” analysis is the science of determining the right questions and of asking them right – which then leads (we hope) to the right answers. In other words, it is more important for us to realize that the answers can only be derived from first determining the questions which produce these answers. That’s analysis.

Perhaps the correct title for this section should have been “Getting at the Right Questions.” You will find a that common theme throughout the “PowerStart” approach is that analysis requires asking a lot of questions, and they have to be not just “good” but the right questions in order to get at useful information.

Finding the right questions is the first step in analysis. And it's the toughest step, too. Asking questions in a "good" way that's clearly understandable to the client in a meaningful context requires the most sophisticated information system on the planet. But "good questions" sounds more like an art than a science. How do we define "good"? What does it mean? Does it mean the same to everyone? All the time?

Good questions are precise and have definite limits that are clear to the listener. A good question indicates a clear context, which makes it more likely that it will get consistent answers from different people.

Naturally enough, not everyone has the same understanding of what makes a "good" question. So, to avoid the complaint that system analysis is an "art," the PowerStart approach to requirements analysis, as described in this book, is based on a systematic approach and language of structure. This *systematic approach* enables the analyst to immediately know all the right questions – in context of the target project – that must be asked of the business experts, to quickly get at the answers.

The *language of structure* required by the PowerStart approach to agile system requirements analysis is a distinct method for asking good questions that are clear, concise, unambiguous, and understandable to the listener. The methods underlying the PowerStart approach provide a solid framework by which system analysts can gather detailed information and document system requirements in a clear and understandable way.

Delivering What the Client Wants, On Time

The most elegant software architecture in the world (a system that works right) isn't worth much if it doesn't do what it's supposed to do (the right system).

A cathedral may display a brilliant system of vaulting, yet it would make a terrible garage. Yes, it would work – but it's not a garage. If a garage is needed, then let's design a good garage. Similarly, an automated system must serve the specific needs of its business users.

One of the worst things we can do as system analysts is to force upon the client what we believe they need. While we might want to try to convince them of what we believe to be worthy, it is our responsibility to deliver to them exactly what they want if they remain unconvinced of the goodness of our approach to their business or their organization. To do otherwise is to suggest that we know more about their business than they do. And if we really believe that ... well, it can't be very stimulating to work for someone whose intellect is so unlike ours, in one direction or the other.

At some time or other, when starting a project, most of us have been faced with the "Blank Page Syndrome."

It works something like this: We get a new project, and all conscious thought immediately grinds to a halt. After a few days of staring at holes in space, wondering where to begin and what to do next, panic and frustration begin to set in. Eventually, however, we overcome that helpless, rudderless feeling, and we finally get started. But getting started was a momentous task.

The underlying methods of the PowerStart approach to requirements analysis specifically get things started really fast, while completely eliminating the frustrating "*Where do I begin?*" and "*What do I do next?*" questions. In a dramatic departure from the uncertain methods of the past, the PowerStart system analysis method is very straightforward, making it easy to begin and finish.

1. Analysis begins immediately when the project starts. Immediately. The first activity is to find the **context** in which to ask specific questions of the business experts regarding the target system. Since all systems are essentially stimulus/response systems, then any system will have many different circumstances or conditions to deal with. These are the contexts, or what we call business **events**.
2. The second activity is to find the **subject** of questions to ask users. We also recognize that the same subject can show up in different contexts.
3. The third activity is to **structure** the question about each *subject* in the right *context*. These "good" context-driven questions with a specific single subject, presented in precise, non-threatening language, lead to "good" answers virtually immediately. This, in turn, leads to rapid and visible progress.

Finding subjects to ask questions about, and the context to ask the questions, is a central issue of the PowerStart approach to business system analysis. While the above is not a specific structure as to how we do it (you have to read the whole book for that), I promise not to leave you with wonderfully esoteric abstractions like "*All you have to do is think it through carefully.*" While that may sound wonderful, it's also pretty useless. The PowerStart system analysis method will guide you precisely to each relevant subject in the target system, and then clearly identify the question to ask in a very specific context.

Rather than the usual abstractions, those who study the PowerStart agile approach as described in this book will learn how to go about identifying business events or conditions in the target system; how to find subjects for context-specific questions; how to structure the questions to be asked; and how to organize and document the results – all without pain.

In future – after you have read this book and studied its content – you will take a very organized and scientific approach to “system requirements analysis,” a subject so often avoided because of its supposed inherent abstract nature. You will bring organized patterns of thought, subject identification, and a specific syntax for questions and answers to the process, enabling you to deliver a system specification that is clear, concise, unambiguous, understandable, brief, in business terms – and completed very quickly.

Providing your clients and users with the systems they want, on time and on budget, is neither impossible nor is it beneficial wishful thinking. It should be a common occurrence. With today’s technology, there is no doubt you can implement anything that you can specify. The challenge is learning how to specify what is wanted – quickly, accurately, completely, and in an understandable way.

Business Development Teams

The most successful project teams consist of (a) business partners (i.e., clients) who bring the business vision, and (b) system professionals who provide the architectural and technology perspective. This kind of proactive cooperation and participation in the system development or acquisition effort allows for differences in perspective and a better focus on the business. These teams – consisting of business experts and system professionals – enable a true “development partnership” between the members of the teams.

Bringing business professionals – often called users or clients – together with system professionals in a JAD²-like *requirements discovery session* (RDS) is the cornerstone of the environment you will create when applying the PowerStart system requirements analysis method. This environment creates a dynamic and a “user buy-in” rarely seen before.

On the surface, the PowerStart approach to system analysis appears to be non-technical in its application. It was developed this way because there was no apparent reason to make system requirements analysis a technical exercise – nor do we want to alienate the business community. Under the surface, however, is a very technically precise approach that enables us to find context-specific questions to ask the business experts, document concise answers to those questions, and a method to generate a prescription for database design (if required) based directly on the business needs.

² **JAD** = **J**oint **A**pplication **D**evelopment; usually conducted in teams consisting of system developers and end users (business experts).

The PowerStart First Principle – Simplicity

Several years ago, Gerald M. (Jerry) Weinberg coined what he called **The Lump Law**, which stated: *"In order to understand anything, we shouldn't try to understand everything all at once."* In other words, if we want to understand anything at all, we should learn about it in tiny, understandable chunks, and synthesize it chunk by chunk. It's frustrating and self-defeating to try to understand everything all at once, especially if the subject is complex or new. It only leads to information overload.

This same principle applies to business and system requirements analysis. That is, if we try to understand everything about the target system's requirements all at once (i.e., linking all the stuff in the system), we'll suffer from serious brain sprain. And we certainly won't get the specification right the first time, because it will simply be too complex. This is usually what happens when we try to figure out the solution first and then try to extract the system specification to fit the solution.



Accordingly, I have paraphrased and restructured Weinberg's *Lump Law* into **The PowerStart First Principle – Simplicity**, because it is so fundamental to our approach:

"Partition the effort to minimize complexity."

In the context of agile business system requirements analysis, this can be stated as follows: Minimize complexity by partitioning the effort required to produce the business and system requirement specification into small, non-redundant, manageable and understandable pieces.

The PowerStart approach to system requirements analysis applies this principle rigorously, and we will revisit it many times, in context.

Source Material & Foundations

You may already have noticed that I have included only a brief bibliography of great works by wonderful people. That's quite intentional. A bibliography, by definition, can only be frozen in the time capsule created by hardcopy publishing. Any bibliography is great stuff when something is first published – but undoubtedly more has been researched, experienced and written since that first publication date on the inside cover of any book.

The abbreviated bibliography at the back is the foundation for the work you're now reading. The PowerStart approach to system requirements engineering is not simply the *Same Old Stuff*, nor is it regurgitated from someone else's work. However, the foundations were established by others in years gone by; and it's their pioneering work that allowed me to put together the PowerStart requirements analysis approach. And if you haven't read the foundation works by now, you probably won't, since so many of these fine old books are now out of date or out of print. Most recent works, including object-oriented versions, are spin-offs from the original foundation works.

And that other long list of loosely related methodology works that so many have come to expect as standard reference materials – well, there isn't much point in referring you to works that are well written but still don't address **how to find the questions** to ask in analysis. (After all, what's analysis all about?) So I won't. But if you want a list of really good books written by excellent authors and thinkers that are quite current – go and see the bibliography at the back. Or visit the website for **The Institute for System Analysis** (www.TIFSA.org) where you will find an extensive list of publications by many great authors.

Let's also give credit to the pioneers who deserve it the most.

None of the methods of the PowerStart approach to system requirements analysis would be possible without the brilliant foundations conceived by several industry leaders. That list – and you'll find lots of information on these individuals with a good Internet search – must include the following:

- The event-based concepts of Steve McMenamin and John Palmer;
- The process modeling methods formulated by Tom DeMarco, James & Suzanne Robertson, and Chris Gane & Trish Sarson;
- The data modeling and normalization methods of Peter Chen, Matt Flavin, Edgar F. Codd, Ronald G. Ross, Charles W. Bachman and John Zachman;
- The information engineering philosophy and system thinking of Gerald M. Weinberg; and
- The object-oriented analysis approaches of Grady Booch and Peter Coad.

And most importantly, many enhancements to this approach to system requirements analysis (i.e., the PowerStart approach) have come from the in-field experience of the many clients I have had the honor to work with over the years.

The New Business Revolution

Before we proceed to discuss the PowerStart approach to business and system requirements analysis it will be useful to review the information technology (IT) industry and how we previously approached system analysis and design. Before we learn about something new, it's often helpful to know where we came from, how we used to do things, what was right with what we used to do, why we do what we do, and why we need to change our conventional approaches.

It's my belief that we generally don't try very hard to change our mental model of our comfortable, known world until we are so expert and familiar with our present paradigm that we become uncomfortable with it. Let me explain.

We readily accept and learn something new when we know all about what's wrong with what we are presently doing. If there is nothing wrong with what we're doing, then we're happy with that situation and have no desire to complicate our lives by changing simply for the sake of change. However, as we become more familiar with an approach, and more expert at it, we also become more familiar with its inherent weaknesses – what we can't do with it, what doesn't work, and what's awkward and difficult. Eventually, these imperfections become the major distractions that become the catalysts for the desire to change. Like our experience with some aspects of yesterday's "Structured Analysis".

When we are completely familiar and expert with our present paradigm, that's when we become uncomfortable with it. Then we seek change. We're ready for it. In the world of software, that's when we switch from one product to another, even though we were initially quite happy with the original product. In the world of system requirements analysis, that's when we are ready to switch from the conventional world of structured analysis and data modeling to a new paradigm.

And the need to change paradigms is now more urgent than ever before – because there is a revolution growing, a business revolution. As a result, business system analysis and design is dramatically changing forever. But to understand why we are where we are today, let's take a brief journey through history to review our pedigree.

Heritage

In the beginning, somewhere around 1955 anyway, the availability of commercially usable computers created a new area of specialization. We called it "data processing".

In those early days programmers wrestled with machine language, and even plug boards, to automate processes to help the number-crunchers in private business and

government – mostly the insurance companies, the banking industry and the military. Wonderful things were expected from this fledgling industry of ours, including a whole lot of magical answers to difficult questions.

Unfortunately, we didn't know the questions, much less the answers. That wasn't anyone's fault. It was a new and young profession and we were all struggling. Sadly, we learned in time, the computer was not the panacea we had so eagerly wished for. In fact, for some, it turned out to be a bit of a headache.

It quickly became evident that while computer systems and programs may be static, business is not. The world just wouldn't stand still while we, coding feverishly in the back room, built the computer system.

The tools we had at the time – 2nd and 3rd generation technology and languages – wouldn't allow us to even consider the dynamic aspects of any business problem, so we built monoliths of code that required vast amounts of money and effort. There are some who are still doing it.

Sometimes these rigid system structures, incapable of responding to changes, didn't at all help companies grow and prosper. While smaller, more dynamic organizations were able to respond to needs of the market, the larger monolithic organizations with extensive rigid system structures lagged woefully behind when they had to address rapid and effective change.

Part of the problem was, of course, technology. The machinery we used (both hardware and software) wasn't capable of meeting the business demands placed on it from either the storage or processing points of view. Handling large volumes of data wasn't very practical, and processing was slow by comparison to the need.

Furthermore, programming was a new and hermetic trade that required highly specialized people. Programming languages were awkward and technically difficult.

These new technicians were to become the gurus of data processing.

Evolution

The problems associated with those early efforts at system building gave rise to the notion that since systems were made of code (an erroneous inference, we may add), then the way to fix them would be to improve the programming languages.

So we see the entry of Assembler, Autocoder, COBOL, and finally (sigh) RPG.

Well, this helped but there still seemed to be "users" who complained that the systems didn't do what they really wanted.

"Well," said the experts, *"we'll freeze the specifications. That will do the trick."*

And it sure did! It helped us apply rigor to the process of building systems. And you know what word usually follows rigor ... "Freezing the spec" did just that.

Those same experts who said "freezing the spec" would solve the problem (we were calling them consultants by now) proceeded to propose another solution to the problem.

"Obviously," the experts said, *"the problem must be in the way the code is organized."*

That sounded reasonable, so we saw the entry of **Modular Programming** and **Structured Programming** – and everyone agreed these methods must be the answer!

And they were. To a point.

These wonderful new methods helped (but only on the technical side) to produce "neat code" which could then be corrected easier. Among other wonders of the western world, we had **GOTO-less programming** and other techniques to reduce the maintenance effort. This certainly made it easier to fix the bugs that somehow always found their way into the computer system.

But it didn't help much with the original code. The users – the business people – they still complained. These new methods sure made programming easier, but it didn't help to accurately specify what the user wanted and what was to be programmed.

So the experts of the day spoke again:

"All right," they said, "if the code is good and it's well organized – clearly, then, the problem must be in the design, right?"

Well, yes, that was right. Of course it was.

So we advanced some more. The **Structured Design** concepts of Larry Constantine and Ed Yourdon helped considerably. But even this didn't solve the problems of systems that didn't quite do what the business people thought they wanted them to do. They weren't always entirely clear as to what that was, but they knew it was more than what they were getting.

What they were getting, they said, was so ... technical. And it should cost less. And, anyway, systems people should be able to do it faster.

But, they sometimes admitted, what they were getting was without a doubt far better than it ever was in the past. Constantine and Yourdon had brought about a quantum leap in computer system specification. Technical, but good.

So, by about 1975, the “*Structured Revolution*”, as we called it, was well underway. Nothing could stop us now! This appeared to be the answer to all our “system specification” problems!

In the subsequent years we learned all about **Structured Programming** from Tim Lister, Chris Gane and Ed Yourdon. We also experienced **Chief Programmer Teams** from Harlan Mills, **Walkthroughs** from Ed Yourdon and Jerry Weinberg, more **Structured Design** from Larry Constantine and Ed Yourdon, and finally **Structured Analysis** from Tom DeMarco, with notational differences in the spin-off version by Chris Gane & Trish Sarson (STRADIS³). A whole lot of “good structured stuff” was being practiced in hundreds of organizations across the United States, Canada and Europe.

We saw enthusiastic system teams designing and implementing systems faster and better than ever before. It was like Camelot in some organizations! We were Knights of King Yourdon’s Court, and the world was Structured. We were winning! We had come of age.

So between 1976 and 1986 over 600,000 system professionals were trained in North America and Europe in the Structured Methods. The legacy lingers to this very day.

And, if we look carefully, we’ll see that most of the legacy systems that we worked on to replace were built during this early period of enthusiasm and passion for the new Structured Methods. It therefore follows that much of the maintenance done on those legacy systems, and their documentation, also resulted from those efforts.

Nonetheless, we did make tremendous gains in both productivity and quality. But maintenance – whether it was defect removal or functional change to make the system do what the client wanted – the amount of maintenance we faced did not go down in a measurable, meaningful way. And our documentation was still a mess.

³ **STRADIS** = **Structured Analysis and Design of Information Systems**, a structured analysis methodology that’s similar to the original work by Tom DeMarco (which was, however, published a year later). Chris Gane and Trish Sarson published this work under the title **Structured Systems Analysis: Tools and Techniques** (ISBN: 0930196007), McDonnell Douglas, 1977.

But why? We had good code (we fixed that one), and good design (we were mending that one too), so what could possibly be wrong?

Let's look at comments by one of our respected industry peers, and what he had to say about the issue. John Boddie⁴, a consultant from Landenberg (Pennsylvania), wrote the following in American Programmer magazine:

*"Let's face it. Most programs spring from truly wretched specifications, if indeed there are any specifications at all. Throughout the development process, users, programmers, and analysts are engaged in a ritual dance of successive **approximation** of the required product.*

*"However, if we can **improve our ability to create specifications** ... the need for the aforementioned dance disappears. Now, armed with detailed design documentation and specifications for the system to be delivered, the programmer needs only to translate the requirements to code. ... **It is the analyst, through improved specifications, that allows the programmer to do an acceptable job.**"*

With such harsh comments from experts in our own field, does this mean that we failed with the Structured Revolution? Could it really be that our analysis was faulty? Could all the goodness and light of Structured Analysis and Design not have been the miracle weapon, the silver bullet, we had been expecting? And if structured analysis and design didn't help us much, could we fix it at all? After all, we were doing our work faster and better. Productivity was up, and so was quality. We knew that much. But, was there really any point in doing the wrong thing faster? No matter how well we did it?

While the best of these "Structured Methods", particularly Tom DeMarco's⁵ (and even the spin-off by Gane & Sarson⁶) became the *de facto* requirements specification for virtually all CASE⁷ products on the market, not much was achieved in the "better-way-of-doing-things" department. The industry virtually stood still, resting on its laurels, for an entire decade.

⁴ John Boddie is author of **Crunch Mode: Building Effective Systems on a Tight Schedule** (ISBN: 0131949608), Prentice-Hall, Englewood Cliffs, NJ, 1987.

⁵ Tom DeMarco, **Structured Analysis and System Specification** (ISBN: 0138543801), Prentice Hall, Englewood Cliffs, NJ, 1979.

⁶ Chris Gane & Trish Sarson, **Structured Systems Analysis: Tools and Techniques** (ISBN: 0930196007), McDonnell Douglas, 1977.

⁷ **CASE** = Computer Assisted Software Engineering.

Driven by new technology, however, choices were starting to appear. But, we still had to solve the nagging question of how to describe and organize the data; i.e., how to define what we needed to know about the *business* and what data was needed to support business *conditions* and *circumstances*.

So during the late 1970's and through the early 80's, on a second front, technology led the way again with the introduction of database management systems (DBMSs) and the need for data modeling. Unfortunately, as time and technological progress moved relentlessly on, new issues of data organization and integration became even bigger and tougher to resolve and understand.

So a whole new set of challenges came into being. The systems community leaped at promises of these new data tools, anticipating that "the answer" was in new ways of database design. So the experts spoke again. "*Organize the data,*" they said, "*and your problems will be over.*"

And a great many latter-day experts went to work to explain the data side of "data processing," trying to explain and implement the earlier works of data gurus such as Dr. Edgar F. Codd⁸.

Following the works of Dr. Codd were two academics – Hans Albrecht Schmid (Germany) and J. Richard Swenson (Canada) – who published an early paper, ***On the Semantics of the Relational Data Model***, which they presented to a Special Interest Group of the Association for Computer Management, at the ACM Conference on the Management of Data (SIGMOD Conference) at San Jose, CA, in 1975. While this early entry in the field of theoretical relational data modeling – based primarily on the work of Dr. E. F. Codd – was well received in the world of academia, it was largely ignored by private enterprise.

However, Dr. Peter Chen followed in 1976 with publication of ***The Entity-Relationship Model – Toward a Unified View of Data***. Peter Chen's now famous work, which expanded on Schmid & Swenson's work of a year earlier, was first published in the ACM Transactions on Database Systems in March 1976. His seminal work is considered to be the foundation of the ER Model. (Dr. Chen's paper is one of the most referenced technical papers in systems journals. For many years he was also the M.J. Foster Distinguished Chair Professor of Computer Science at Louisiana State University.)

⁸ Dr. Edgar F. Codd is considered to be the father of the relational data model by just about everyone, including himself. His seminal work is considered to be an article titled ***A Relational Model of Data for Large Shared Data Banks*** published in the Communication of the ACM (June 1970). However, this was actually preceded by release of his IBM Research Report (RJ599, August 19, 1969) on the subject titled ***Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks***. Since his early work, Dr. Codd has consistently expanded his definition of the relational model in many papers. In some academic reports Dr. Codd is referred to as "Edward" or "Ted" Codd.

Eventually there were a whole lot of others who either became recognized or joined the fray: people like Charlie Bachman, Chris Date, and John Zachman, to mention a few. These industry experts, and many others, all had good technical answers to the organization and location of data. In other words, everyone agreed they introduced good, sound stuff.

So, while the Structured Analysis Revolution was making extraordinary gains on the process side of system development, successes on the data-oriented side were also very impressive.

With the rise of data modeling⁹ whole data organizations were set up to design, build and administer databases that would contain all you ever wanted to know about the organization but were afraid to ask. Gargantuan projects were undertaken to define and specify requirements and to control access to corporate data.

Unfortunately, while the theories were very convincing, only a small handful of these "big bang" projects were actually successful.

The introduction of DBMSs placed the focus on "data" (a very good move) and away from "processing" (not so good). Sadly, while we got closer to relational databases, this selectivity of emphasis moved us even further away from the business.

The argument was that if data was organized internally to reflect the interests of the corporate structure, and the latest output technology was added at the user end, then everyone in the organization would have access to all the data they needed to manage the business better. All the information would be "*right there in the database.*" All anyone had to do was to ask.

To design such a structure it was necessary to perform a variety of mysterious analytical tasks to reduce the data to its so-called simplest form. Part of this exercise was to create a model of the corporate data in "*normalized form*" suitable for database design. Database designers wrestled with the technical problem of developing a repository of "*normalized data of interest to the enterprise*" and dutifully ignored other business objectives in favor of focusing on logical and physical database designs.

To accomplish this, large departments sprang up, and these database designers wanted to implement a whole world of data. While the motivation was right, the implementation clearly had a problem. We soon came to call some of these valiant efforts "*close encounters of the third normal form*".

⁹ Data modeling is also known as entity-modeling, entity-relationship modeling, object modeling, enterprise modeling and E-R diagramming; and sometimes even Information Engineering (IE).

As we now know, most of these exercises took a long time and cost lots of money. And the problem of accurately and quickly specifying the solutions clients thought they wanted still hung on, despite the utopian environment offered by IMS, IDMS, R2D2, and S2000.

Even after we added an “R” to IDMS, and accepted the second coming of DB2, the problem of creating an accurate specification, fast and in non-technical business terms, was still not solved.

Having said that, let’s not misunderstand. There has been, and continues to be, very considerable evidence of success on the “data” side of the information technology industry. The only point of contention is the selectivity and exclusion of process orientation by some of yesterday’s and today’s professionals.

In my view, the process-oriented data flow diagramming specialists – in their excitement and enthusiasm over the Structured Revolution – only looked at one side of the equation; while the data-oriented experts only looked at the other. And never the twain would meet. Yet the profession had always been called “data processing.” There had to be a hint there somewhere. In my opinion, there must be two sides to the business of building systems: The data side and the process side.

Data Modeling vs. Information Modeling

In 1981, the late and great Matt Flavin wrote a small, concise book describing a workable approach to defining organizational information needs. His wonderful book, ***Fundamental Concepts of Information Modeling***, was reprinted in 1986 by publisher Prentice-Hall¹⁰. The book is neither well known nor well read. But it was well ahead of its time.

Incredible as it may seem, that book was written as long ago as the late 1970’s, and then first published in 1981. In his book Matt proposed the use of object-based entity-relationship diagrams to illustrate the organization of information based on the way a company conducts its business.

Matt’s version of the entity-relationship diagram (ERD) was supported by descriptions, in business terms, of each object’s data and business policies. His work was largely ignored because most data processors of the day were too interested in the emerging technologies to pay attention to the esoteric art of business-based analysis, including data analysis, however valuable it might be.

¹⁰ Matt Flavin, ***Fundamental Concepts of Information Modeling*** (ISBN: 0133355896), Prentice Hall, Englewood Cliffs, NJ, 1986.

The challenge in Matt's great but complex work was that to understand it, you already had to understand its foundation, and you had to have an open mind to be able to deal with new insights. To a new reader – someone not yet introduced to the concepts of data modeling – this book would probably mean less than nothing. And if you already knew something about data modeling, you would need a very open mind indeed. Matt's thinking was quite different from the accepted, convoluted thinking of the day.

There is nothing elementary about this book, as is suggested by its title. It is not an easy read. While it does contain a clear description of what I have come to believe is the essential foundation for business data modeling, it's still quite an advanced read. It integrates concepts of Business Object Modeling (more commonly known as *conceptual data modeling* and *object modeling*) with database design and implementation concerns. It is detailed, concise, clear (if you can learn to love the pain of studying it), and quite unambiguous. But, it is deeply complex. It is not for a novice.

But it's nice to know who shone the first light on the path to lead the way. Matt's work was the first significant expansion, in my opinion, to the works of E.F. Codd and C.J. Date and others of that era. Matt treated it as a business modeling exercise, using objects, which resulted in a database model.

We certainly have come a long way since Matt wrote his powerful book. His little book (one of the IT profession's best kept secrets) was a personal significant discovery of that time. It eventually provided the primary foundation necessary for development of the requirements analysis method that we now call the PowerStart approach. Matt's foundation work, as well as discoveries in the field of consulting, eventually led to the PowerStart business system analysis method now taught through hundreds of workshops and used on many consulting assignments.

Meanwhile, the data-oriented school of conventional beliefs – led by Charlie Bachman on one side and Dr. Codd on the other – was arguing and bickering as to the true meaning of the relational model and how this "true normal form" could solve all problems past, present, and future.

While all this was going on, equally interesting work was developing on another front.

Essential Business Models

The “Structured Analysis” camp, unfettered by internal bickering (mostly), continued to refine data flow diagramming, or process modeling as it was coming to be known, during the early 1980’s.

Their ongoing work – felt to be old hat by some since it didn’t focus on data modeling, and ridiculed as the “same old stuff” by those who were out of the loop and didn’t understand it – was intended to solve another problem: How to not use the entire budget documenting the current system, only to throw the documentation away when done or, at best, to archive it in the darkest dungeon of documentation. And, worse, the conventional approach to Structured Analysis – whether it was top down or bottom up – didn’t seem to give analysts a better understanding of the business requirement after they had done “structured analysis” than before.

So, how did all this come about? Well, through the “structured revolution” between 1976 and 1984, many of us were weaned on a litany of ...

Current Physical → Current Logical → New Logical → New Physical

This became our shorthand version of a system analysis methodology. Let me refresh your memory as to how this worked:

1. First – we uncovered the “**Current Physical**” system; that is, we documented the existing system in all its finery as best we could. This took a long time and never delivered anything new. (This is sometimes known as the “**AS IS**” system.)
2. Second – we *distilled* the “**Current Logical**” system from the *Current Physical*; that is, we tried to remove all traces of implementation in the existing system, so we wouldn’t have an implementation or technological bias when developing the new system. Most system designers had a lot of trouble with this.
3. Third – we added new “logical” business requirements to the *Current Logical* model to arrive at the “**New Logical**” system model; that is, we added new business requirements without reference to *how* it would be implemented (the system solution), but rather *what* would be added. Keeping a solution bias out of this proved again to be very difficult for most system designers.
4. Fourth – to the *New Logical Model* we eventually added hardware, software, manual procedures and constraints, finally delivering the “**New Physical**” system model – or, a specification for all seasons. This was more fun than the other three. (This is sometimes known as the “**TO BE**” system.)

Current Physical → Current Logical → New Logical → New Physical. **Whew!**

But for those of us who understood it, or thought we did, this was still a whole lot faster and better than the more conventional method of creating a system specification. After all, a picture was worth a thousand words. Accordingly, the hundreds of pictures we produced were worth ...

Two things, however, were wrong with this otherwise excellent process:

Problem # 1: The method almost forced system developers, by default, to build the same system again, with the same ways of doing business. The major differences in the "new" system would be better application of technology and perhaps better, more reliable code. But certainly not much was achieved in the "better business methods" department. And while we all agreed that systems consist of software and manual procedures, no one ever really looked beyond the code.

Problem # 2: Re-creating the existing system's documentation (that was the *Current Physical* model) was certainly an admission that the "current" documentation either wasn't there or couldn't be trusted. While our users were impressed by our ability to recreate some of the documentation, they were less impressed by the fact that – after a great deal of time – we were no closer to specifying their new system.

Interestingly enough, this exercise of creating the *Current Physical*, or documenting the existing system – full of its biases and older ways of doing business – was one of the most time consuming and unproductive undertakings imaginable. Trying to figure out the intricacies and culture of the existing system often became an archaeological dig of mammoth proportions.

But let's face it: The user didn't ask for the current system; they asked for a new one.

When we were done (and most will admit that we were never really done), we were sure a whole lot wiser about the existing system, but we were not a whole lot wiser about the requirements for the new system! It took a long time to do this *Current Physical* model, and the resulting documentation was usually filed away in some dark recess to never be looked at again. But we did it because it was the then-current industry wisdom. And it was better than what we had before.

No wonder users got a bit frustrated when told that a team of highly paid computer professionals were documenting the system some other highly paid computer professionals designed and documented a few years ago. *Why*, they wondered, *do we need a blueprint of the old building*, functional as it is, *to be able to build a new one*? The new building would have different architecture, different and expanded functionality, different dimensions, and different patterns of data and traffic flows.

Why did we need a blueprint of the old building to be able to create a blueprint for the new building?

The first theory (told to us by the experts) was, of course, that we had to know where we had been to be able to figure out where we were going; otherwise we might stumble along the way.

This is interesting, but...

The other theory (we were told by these same experts of the day) was that it was far too difficult to get our users to accurately tell us about their requirements. After all, we were told, they didn't really understand the complexities of the business, and in any event they would only tell us about the old system with a few changes. They were just users after all.

(Thankfully, times change. Today we call them clients and subject matter experts.)

We now know that those popular theories were based on some imagined fantasy. Our users already knew where they were, and where they had been. That's why they wanted new systems. And they knew it wasn't just a matter of regurgitating the old system with a few new wrinkles. They knew that they wanted changes to the current system because they were dissatisfied with the status quo. And they wanted to improve the business methods as well as the application of technology. They wanted to re-engineer the systems and, if possible, the business.

And, yes, they could tell us all about it – if we had a common language with which to communicate effectively, to ask questions, and to document the results. If we only knew how to ask the questions, then we would be able to get at the answers our users had.

I believe that a *Current Physical* model of a system (or an "AS IS" model) was only documented because analysts didn't know how to find the specific questions to ask users. Therefore, it becomes easier to document what already existed and then use that foundation as a starting point.

In the past, it made a lot of sense to document answers from the existing system (based on code and output) when we had no idea what the questions should be, and we didn't have a common language with which to communicate with users. It made a lot of sense – but it didn't make a new system, with new functionality, or new opportunities for business or process re-engineering. It only gave us the old stuff, with old business methods, and it took much too long – simply because we didn't know how to find the questions to ask them.

And that's the way almost everyone has been doing it since the beginning of time, when wheels were square. And we all know it doesn't work, don't we?

But since we did have to go through the exercise of the archaeological dig to create the *Current Physical* model, it's little wonder that it took almost forever doing analysis. And to make matters worse, our users were rarely involved in this front-end dig to uncover the existing system.

Now if this wasn't enough, the difficulty encountered in extricating a *Current Logical* model from the *Physical* one proved to be not for the faint-of-heart. Senior analysts suffered serious trauma attempting to extract the logical processes and their associated data, piece by tiny, unyielding piece. The resulting models were woefully lacking in accuracy and acceptability, and the work done was often far less productive and less understandable than the experts had predicted.

So, what was the problem?

The tools introduced by Tom DeMarco (data flow diagrams) were clearly more than adequate for illustrating data movement, memory, and external interfaces, as well as for specifying the processes. But there was something missing. It wasn't the tools (they seemed to work OK). And there wasn't much point in throwing away a good set of tools with which we seemed to be able to communicate some things quite effectively. Clearly, we needed to find new ways of using the tools.

On the other side of the street (some said there was more light there), the methods advocated by Codd, Chen, Date, Bachman and others (entity-relationship diagrams) were also more than adequate to define and understand the data from a database design perspective. But there was something missing there too. Again, it didn't appear to be the tools. Once again there wasn't much point in throwing away a perfectly good set of tools. Equally clearly, we needed to find new ways of using these data modeling tools – and perhaps to consolidate or integrate them with the other tool, the data flow diagram.

All of this research and experience on both sides of the "data processing" coin was of great value. We were finally realizing that we were a maturing industry, and we had to use some of this "good stuff" from both schools of thought to address business concerns and not just data and technology.

And then it happened – the first few tentative steps towards using all the great good from the preceding 20 years of systems analysis experience and applying it to real business needs. In 1981, Stephen McMenamin and John Palmer publicly introduced the concept of "events." That was the same year Matt Flavin first published ***Fundamental Concepts of Information Modeling***. And in 1982, Steve McMenamin and I met in Toronto and discussed the event-partitioning concept at length.

McMenamin and Palmer later followed with their groundbreaking book, ***Essential Systems Analysis***, first published by Prentice-Hall in 1986 and reissued in 1994¹¹.

In my opinion, this was the first significant early work on event-partitioned analysis. Among other interesting items, their book showed how to avoid the laborious step of creating a model of the *Current Physical* system. This allowed analysts to start at the “logical” model rather than *Current Physical* level and thereby avoid the painful process of “logicalization”. The word “essential” was used to describe the processes and the data that had to exist regardless of the technology applied to the target system. These models were said to be technologically neutral.

McMenamin and Palmer’s work replaced the *Logical Model* defined by Tom DeMarco in name only¹². The new “event model” resulted directly in the previously ill-named *Logical Model*. Event models also provided a basis for modeling the entire business requirement rather than just the computer system.

In theory (but certainly not in practice) the *Current Physical Model* disappeared (thank goodness) and along with it the arduous task of logicalization – sometimes known as “*work in the impossible region*.” Clearly the focus was now on the business and not its machinery, in contrast to the earlier notions that concentrated exclusively on the technology and the software systems.

In my own earlier book on a complete system development methodology¹³ I devoted a section to expanding McMenamin & Palmer’s event-partitioning concepts. However, that book was primarily on a flexible project-based methodology and did not deal in depth with the concept of event-partitioned object-based analysis. As you can imagine, many advances have occurred since then.

A New Structured Revolution

So what came out of all of this history?

Well, quietly but surely, a new revolution has been growing since the first days of the Structured Methods – a revolution from within the business community and from outside systems departments. The legacy of the Structured Revolution is business and user-centered systems development.

¹¹ Stephen M. McMenamin & John F. Palmer, ***Essential Systems Analysis*** (ISBN: 0132879050), Prentice Hall, Englewood Cliffs, NJ, 1994.

¹² In ***Structured Analysis and System Specification*** (ISBN: 0138543801), Prentice Hall, Englewood Cliffs, NJ, 1979.

¹³ ***A Game Plan for Systems Development*** (ISBN: 0133461564), Prentice Hall, Englewood Cliffs, NJ, 1988.

With the earlier introduction of walkthroughs we involved users in looking carefully at their systems. Now, after years of participating in walkthroughs, these users are looking for better business methods to analyze and specify their requirements, and new ways of applying technology to their business systems. Accordingly, many organizations are unwilling to continue supporting technological solutions that are not understood, developed or supported by the user community.

The tolerance level from the business community for “neat” technical solutions is at its lowest ever. They are now demanding – and for the first time with real teeth, since they can now engage in a technological end-run on the systems department – real productivity in analyzing, design and specifying their target systems. And some of them won’t settle for anything less than achieving real productivity immediately.

The New “Agile” Business System Analysis

I’m sure we all agree that to survive and prosper in these early years of the 21st century we need fast, accurate ways to specify business requirements – much faster and better than the original methods of the Structured Revolution.

The data flow diagrams from the 1976 School of Structured Analysis, and entity-relationship diagrams from the opposing 1976 School of Data Modeling, are both still invaluable in their usefulness to represent business needs graphically. And any technology that makes getting the picture easier as well as faster is going to be around for a long time. So let’s not totally discount these good, old tools. But they are *tools*, not methods. Methods (such as the approach defined in this book) use these tools, and will survive the test of time.

These vibrant, graphic tools – modified to suit newer methods – enable us to uncover and document useful system requirements virtually immediately. But to do this effectively, we must also liberate ourselves from an exclusive data modeling or an exclusive process modeling mindset. After freeing ourselves from the code-oriented monolithic mindset of the 1970’s and 80’s, you will find it is much easier to accept the notion that we need only describe the client’s information requirements in business terms, and in complete yet sufficient detail to move on to the next step – the solution design and implementation of a commercially usable system.

Combining the methods in this book with a highly client-interactive “discovery” approach, our experience indicates that the PowerStart approach to system requirements analysis can be effectively implemented in virtually any size of organization, including government, given motivated individuals.

And, yes, the business partners (users) can tell us all about their needs – they can answer our questions, clearly and concisely – when we use a common language with which to find the precise questions to ask and to document the results. When we know how to find the exact questions to ask, then we can get the answers they have.

Conclusion

None of what I have said invalidates or is intended to treat cavalierly any previous work in Structured Analysis, Structured Design, Data Modeling or Object-Oriented Analysis. Those were the foundations, and they were very good. The foundations we have so far discussed have enabled the development of this new discipline of integrated data modeling and process modeling to respond to new business challenges – the challenges of greater understandability, faster results, and truly significant productivity gains.

The target of analysis and design has always been to understand the exact needs of the client and their business, and to create a specification of the target system so we can acquire it or build it and implement it. While we were often successful at creating a generalized requirements document, we have also often failed to document a clear and unambiguous functional specification – quickly. That’s because, until now, we really haven’t had an appropriate, user-oriented, non-technical approach to finding the exact questions that would enable us to understand and specify the client’s target system. With today’s technology, we can implement anything that we can specify. The challenge, then, is to know how to specify – clearly, concisely, completely and accurately.

As with any profession, this is best accomplished by understanding the concepts developed in the past and by adapting these to our modern environment; and to avoid dogmatic application of old rules. The best professionals do not persist in using methods from the past, just because the methods were wonderful – in the past.

Commercially, we have only about five decades of history in the IS business. Compared to other industries, that’s not a lot of time. But during those five decades, we developed a whole lot of “good stuff.” And in the methodology underlying the PowerStart approach to system requirements analysis you will see that we keep some of that “good old stuff”. I’m a firm believer that it isn’t useful to throw everything out every time someone discovers a new mousetrap.

By the time you first apply the PowerStart approach on a project I am confident you will agree that this is a far better approach than any you have ever used before, built on a solid foundation of practical methods. You will see that specifying systems well

doesn't have to take forever. And the process doesn't have to be alien to the business folks either. Not any more.

But to accomplish all of this we must be prepared to use modern methods suitable to today's business environment and to the technologies that can catapult businesses successfully into the competitive 21st century. We have to stop arguing about “top-down” and “bottom-up” and get on with the work. And that means the emphasis must be on the business policy. A clear understanding of policy – and the ability to distinguish it from procedure – is essential to successful, fast, high quality work.

And we must always remember – ***form*** (the shape or implementation a system takes) ***follows function*** (the condition that needs to be supported). Understanding this fundamental concept is essential to developing great systems¹⁴.

To accomplish all of this we must have a passionate belief that the only effective way to reduce the maintenance problem is to get it right the first time. And “getting it right the first time,” above all else, is knowing how to find the precise questions to ask our business partners and system users.

One thing is for certain: If we haven't got time to do it right, we sure won't have time to do it wrong.

Many important persons have suggested for many years now that good system analysis and design is really a matter of art rather than science. I disagree. If that were true, then most professionals in the industry today would be excluded from being “good” analysts and designers. The quality of a system specification is not a question of “goodness”, but rather an issue of whether or not the requirements have been specified accurately. Accuracy and completeness are the only measures of goodness.

It is now time to stop calling system analysis an “art” (which disqualifies just about everybody, since so few of us are artists) and call it by its proper name – a precise discipline that everyone can learn. Let's call it simply *business system analysis*. After all, that's pretty simple language that just about everyone understands.

¹⁴ Louis H. Sullivan in **The Tall Office Building Artistically Considered** (Lippincott's magazine, March 1896) crafted a beautifully written commentary that is surprisingly relevant to modern system engineers as well as architects. Sullivan discusses the “essence” of a problem, *iteration*, *refinement*, and how form (design or implementation) must, by natural law, follow functional definition (the essential business needs) in architecture. He even discusses principles of normalization and canonical synthesis in the architectural world (“As I am here seeking not for an individual or special solution, but for a *true normal type*, the attention must be confined to those conditions that, in the main, are *constant* in all tall office buildings, and even mere incidental and accidental variations eliminated from the consideration, as *harmful to the clearness of the main inquiry*.”) Sullivan is, of course, famous for the often paraphrased term “Form ever follows function.”

Getting Started

With that discussion of our history and pedigree, you have begun a marvelous journey – a dynamic experience with the PowerStart approach to agile business system requirements analysis. Let's move ahead through these pages together and learn how to uncover, determine, specify, and integrate information technology with today's business requirements. Let's explore fascinating and powerful methods of good business-oriented communication, and another way of doing top-quality work in the fastest way possible.

And let's turn the art of analysis into a discipline. I know you will never look back.

Trond Frantzen
Trond.Frantzen@PowerstartGroup.com