Structured Analysis

Tom DeMarco
Principal of the Atlantic Systems Guild
New York

M.S., Columbia, Diplome,
University of Paris

Bell Labs: ESS-1 project

Manager of real-time projects,
distributed online banking systems

J.D. Warnier Prize, Stevens Prize

Fellow of IEEE

Major contributions:
Structured Analysis, Peopleware

Current interests: project management,
change facilitation, litigation
of software-intensive contracts

Tom DeMarco

Structured Analysis: Beginnings of a New Discipline

How it happened

When I arrived at Bell Telephone Laboratories in the fall of 1963, I was immediately assigned to the ESS-1 project. This was a hardware / software endeavor to develop the world's first commercial stored program telephone switch (now installed in telephone offices all over the world). At the time, the project was made up of some 600 persons, divided about half-and-half between hardware and software. There was also a small simulation group (a dozen people?) working to create an early prediction of system performance and robustness.

I was at first assigned to the hardware group. My assignment was to develop certain circuits that enabled Emergency Action, the reconfiguration of processors when one was judged to have failed. This was an intriguing assignment since each of the two processors would diagnose the other and then try to decide together and agree on which one was incapable of further processing – but somehow still capable to judge its own sanity.

To all of our surprise, the software for the project turned out to require a lot more effort than the hardware. By early 1964 an increasing number of hardware engineers were being switched into software to help that effort along. I was among them. By the end of that year I considered myself an experienced software engineer. I was twenty four years old.

The simulation team under the management of Erna Hoover had by this time completed the bulk of its work. The findings were released in the form of a report and some internal documentation. The report (analysis of system performance and expected downtimes) was the major deliverable, but it was one aspect of the internal documentation that ended up getting more attention. Among the documents describing the simulation was a giant diagram that Ms. Hoover called a Petri Net. It was the first time I had ever seen such a diagram. It portrayed the system being simulated as a network of sub-component nodes with information flows connecting the nodes. In a rather elegant trick, some of the more complicated nodes were themselves portrayed as Petri Nets, sub-sub-component nodes with interconnecting information flows.

By now I was in full software construction mode, writing and testing programs to fulfill subsystem specifications handed down by the system architect. While these specifications were probably as good as any in the industry at the time, I found them almost incomprehensible. Each one was a lengthy narrative text describing what the subsystem had to do. I was not the only one having trouble with the specs. Among my fellow software craftsmen (out of deference to Dave Parnas, I shall henceforth not use the term ‘software engineer’ to describe myself or any of my 1960s colleagues), there was a general sense that the specs were probably correct but almost impossible to work from. The one document that we found ourselves using most was Erna’s PetriNet. It showed how all the pieces of the puzzle were related and how they were obliged to interact. The lower level networks gave us a useful pigeon-holing scheme for information from the subsystem specs. When all the elemental requirements from the spec had been slotted by node, it was relatively easy to begin implementation. One of my colleagues, Jut Kodner, observed that the diagram was a better spec than the spec.

When I left the Labs, I went to work for what today would be called a system integrator. I was assigned on contract to build first one and then another time-shared operating system for the then new IBM 360. On both of these projects I created my own network diagrams. In place of a normal specification, I wrote one “mini-specification” per node. I used the same trick when I went to work for La CEGOS Informatique, a French consulting firm that had a contract to build a computerized conveyor system for the new merchandise mart at La Villette in Paris. (La Villette was the successor to the ancient market at Les Halles.) And I used the same trick again on a new telephone switch implemented for GTE. Note that all of these projects (two telephone switches, two time shared executives, and a conveyor control system) were real time control systems, required to meet stringent time constraints on the order of a few milliseconds. All of my work up to this point was in the domain that I now call engineering systems. I had never participated in a single commercial data processing project.

By 1971 I was went to work for the first time in my life outside the engineering sector. I was involved for the next four years building banking systems in Sweden, Holland, France and finally New York. Here again I used my networking methods, though with a bit less success. What was different in financial applications was the presence of a database in the middle of my network. At first it could be treated as a simple file or repository of information. But over time these databases were to become more and more complex and my method gave me no particularly elegant way to deal with them. The truth is that the networks were an elegant and useful description of control systems, where data flow provides the best representation of overall system function, but a less useful tool for database systems where the structure of the repository itself is a better representation.

Though I have now come to believe that dataflow methods are ill-suited to business applications – at least compared to data modeling methods – the networks were as big a hit with my banking customers as they were with my engineering customers. Remember that in the early seventies, the breakthroughs of data modeling including E-R diagrams and relational database had yet to happen or were happening only in academia. For my customers, the network specifications that I was showing them were the only alternative to dreary and endless narrative specifications.

How it became a commercial success

It was in 1974 that I first came across the work of Doug Ross and John Brackett of SofTech. Their tool, called SADT, was a much advanced and in some ways much more elegant variation on my network specifications. It also was the first time that I had seen the adjective 'structured' applied to leveled diagrams. Since all things structured were hot in 1974, this was good news to me. Imagine, something that I'd been doing for years now turned out to be 'structured'!

In 1975 I sought out my old friend Ed Yourdon and proposed to him to develop my network specification concept (much improved by my exposure to SADT) into a two-day training seminar for his new seminar company. Ed's little company had already dabbled in something called Structured Analysis, and though my concept and theirs were only marginally similar, he allowed me to use the term as if we'd really been talking about the same thing all along. Within a year, the original sense of what would constitute Structured Analysis was completely replaced by my concept of writing specifications in the form of leveled dataflow diagrams with complementary data dictionary and mini-specifications.

The course and its successor courses on Structured Analysis were a huge success. By the end of 1975, Ed had assigned a dozen instructors to teaching my courses, and with all the royalty income thus produced, I stole away for two months to write a book [*Structured Analysis and System Specification*, Prentice Hall, 1975] and a video training sequence which both subsequently became excellent royalty properties. A friend later observed – rather nastily, I thought – that if I had spent all my months as productively as those two months working on the video and the book, my present income would be several million dollars per month.

Over the next twenty years, the method prospered. It was implemented in virtually every CASE system ever produced. When CASE went away, the method persisted. Today dataflow representation is a component of virtually every current process, though few organizations are still using my 1975 prescription in its original form.

How my own perspective has changed since 1975

While the market embraced the method called Structured Analysis, I myself have come to have some doubts about the whole approach. Remember that my own early experience applying the method had been in control systems, not commercial systems. When I first started advocating the method it was only to those who were building control systems. It was a great surprise to me that the commercial sector liked the method. In retrospect I believe that the whole dataflow approach is vastly more useful for control systems than typical commercial applications, and the appeal to the commercial users was mostly due to a complete lack of well thought out alternatives. Commercial IT ended up using a control system method because there were as yet no attractive commercial IT methods available.

When good alternatives did become available, many commercial organizations stuck loyally to my approach. I found myself regularly visiting clients who announced themselves to be 100% 'DeMarcofied," though I myself would never have used my method on the projects they were doing. I came to the conclusion that these companies were using a method that was poorly suited to their real needs and the reason had little to do with anything purely technological. They were stuck on the method because it gave a comforting sense of completeness, it appeared to them to be The Answer to all of their problems. When it didn't solve their problems they blamed themselves and tried harder.

I now believe that my 1975 book was overly persuasive and that many in our industry were simply seduced by it. This is partly the result of my unconstrained enthusiasm for a method that had worked superbly for me (in a limited domain), and partly the result of the dismal state of the art of IT books at the time. Many people have told me that mine was the only IT book they ever got through awake, and the only one they ever enjoyed. I think they adopted its prescriptions thinking, "this guy may be dead wrong, but at least I understand what he's saying.

Bill of particulars, then and now

Important parts of the Structured Analysis method were and are useful and practical, as much so today as ever. Other parts are too domain-specific to be generally applicable. And still other parts were simply wrong. In order to call attention to these different categories, I offer the following commented summary tables, the first showing what I thought I knew in 1975 and the second showing what I still believe to be true. As you will see, there are some substantial differences:

What I Thought I Knew in 1975

Principle	Commentary
1. Narrative specs are dumb	These "Victorian Novel" specifications neither specify nor inform
2. Four-stage modeling	A dataflow representation of a system is a model and the analysis life-cycle consists of building a sequence of these models showing four different stages
3. Dataflow is the essential view	The point of view of the data as it passes through the system is the most useful
4. Top-down partitioning	Top-down is good; bottom-up is evil
5. Loose connection criterion	The validity of any partitioning is a function of how thin the interfaces are
6. Defined process of analysis	System analysis always has the same well-defined steps
7. Pseudo-coded minispecs	The lowest level is defined in a formal way
8. Work at the user's desk	Analysts shouldn't hide in their own offices; the real work of analysis is at the user's desk
9. Philosophy of iteration	You can never get it right on the first try; success comes from numerous iterations, each one better than the last
10. The customer is king	The customer knows what the system has to be; the analyst's job is to listen and learn

There were other things that made up the discipline, but these ten were its essence. I felt strongly about all of these but at the time it was the top-down characteristic of the approach that most charmed me. After all, without the notion of top-down, the method could hardly be characterized as 'structured,' and that was an appellation that I coveted. (Remember that the structured disciplines were at their peak in 1975.)

I am writing this in the fall of 2001, and obviously much has changed. Not the least of what has changed is my own perception of the business of systems analysis and specification. In the next table I reproduce the ten principles showing in dark letters those that I think still apply in whole or in part. The gray shaded "ghosts" are just to remind you of the sense of those principles that didn't in my opinion survive the test of time:

What I Still Believe

Principle	Revised Commentary (as of 2001)
1. Narrative specs are dumb	Narrative specs are not the problem; a suitably partitioned spec with narrative text used at the bottom level makes a fine statement of work
2. Four-stage modeling	The four stages I proposed in 1975 were far too time consuming
3. Dataflow is the essential view	Dataflow is one of the essential views, not the only one
4. Top-down partitioning	Partitioning is essential in dealing with anything complex, but top-down partitioning is often far too difficult to achieve and not at all the great advantage it was touted to be
5. Loose connection criterion	This is an important truth: when you're attacking complexity by partitioning, the thinner the interface, the better the partitioning – if the interfaces are still thick, go back and partition again, searching for the natural seams of the domain
6. Defined process of analysis	Defined process is a holy grail that has never yet been found and probably never will be
7. Pseudo-coded minispecs	It's useful to partition the whole and then specify the pieces, but pseudo-code was an awful mistake (puts analysts into coding mode when they should be busy analyzing)
8. Work at the user's desk	Analysts have a tendency to hide at their own desks, but much of the action is in the business area and they need to venture out to find it
9. Philosophy of iteration	We never get it right the first time; the best we can do is improve from one iteration to the next; if we can continue to do this at each iteration, we can get arbitrarily close to a perfect product
10. The customer is king	See below . . .

If I'm right that the specification-by-network approach does not require and never did really benefit from being top-down, then the entire method never did justify the name 'structured.' That is what I believe today. We all profited by calling it structured, but it wasn't. To make matters worse, the attempt to achieve top-down representation sent projects off on a meaningless wild goose chase. These days I often encounter project teams working with enormous diagrams of connected software pieces. These diagrams take up a whole wall of a war room or are laid out on the floor with developers on their hands and knees crawling over them. Of course they are a pain to update, often hand-annotated, not reproducible, don't fit into anybody's documentation standard. And yet they are useful, that's why people use them. This use seems much more consistent with the early value I perceived in dataflow networks.

My final point (my loss of faith that “the customer is king”) is not just a change in my own thinking, but a sign of the maturing of IT in specific and of the business climate in general. In 1975, the typical commercial system we built was a first time automation of what had before been done manually. The customer, of course, was the only one who knew what this was all about and his / her sense of what the automated version would have to do was prime.

Today we are building third and fourth generation automated systems, and IT personnel are often as well or better informed about how the existing system works as their business partners. More important, the meaningful successes of IT today are no longer to be achieved by simple automation or re-automation of existing process. We have moved on to a new era: Our challenge today is to combine improved information technology and market opportunity in order to create product that is radically different from what could have been achieved in a less connected world. This tells us that the new king is neither client nor technologist, but their partnership: the tightly merged amalgam of business and technological expertise. Companies that achieve and maintain such a partnership are the ones who will prosper.

Tom DeMarco

Structured Analysis and System Specification

Yourdon, New York, 1978
pp. 1-7 and 37-44