Larry Putnam's Career in Software Estimating

Early Interest

I first became interested in computers while I was a young officer in the Army doing graduate studies in physics at the Naval Post Graduate School during the period 1959 to 1961.

In one of my physics courses I needed to do some very detailed, tedious calculations which required precision out to twelve decimal places. The only tools to do that kind of work at that time were big, clumsy, desk-top mechanical calculators. For one graded exercise I had to go into the local town and hire one of these machines over the week-end to be able to do the necessary calculations. This was back in slide-rule days; it was just impossible to get more than about 2 to 3 decimal places of precision.

1961-1964 Army Nuclear Weapon Program Office

Following graduate school I was assigned to the Army's Special Weapons Development Division in the Combat Development Command at Ft. Bliss, Texas. One of my jobs there was supervising the preparation of the Army's nuclear weapons selection tables. This involved a great deal of calculation of tables pertaining to the effects of nuclear weapons. These tables were used to pick the right weapon for a particular tactical operation. The Army would have to re-do these tables every time their inventory of weapons changed. Consequently we were continually turning out new tables which were several hundred pages in length.

We were doing this work on a small computer about the size of a refrigerator, manufactured by the Bendix Corporation. It was a model G15 computer and had about the power a programmable calculator has today. This machine was programmed in Assembly Language. Later I had an opportunity to program the machine in a higher order language, ALGOL. These were simple engineering programs of perhaps 50 odd lines of code, very small programs indeed.

In l966, while an instructor in nuclear weapons effects, at the Defense Atomic Support Agency (now called the Defense Nuclear Agency) in Albuquerque, New Mexico, I needed to do some blast calculations to support some of the teaching that I was doing. We were located immediately adjacent to the Sandia Laboratories which was the principle Atomic Energy Commission contractor responsible for the arming, fusing and firing components that went into our nuclear weapons.

The national nuclear program has always been a big user of computers in their research and development work. They were always on the forefront of computing technology and bought the biggest and best that were available from the computer industry. Just at the time I needed to do these calculations, Sandia Labs had just received the largest available scientific computer from UNIVAC, a Model 1108. It was installed next door to our facilities. At first they had far more computing

capacity in that machine than they needed for their own work. So they advertised that their machine would be available for use by other people on the base for their own scientific calculations. They even offered to teach people to program the machine in FORTRAN.

I signed up for the course. In about 15-20 sessions, I learned enough about FORTRAN programming so that they would grant me access to their machine.

At that time, one had to prepare a deck of IBM punch cards and submit the job the day before; the computer operators would put the job into the run stream for the computer that night. In the course of learning how to program in FORTRAN, I never did learn the procedure to punch up the job control cards to initiate the actual execution of the FORTRAN program. So I found I had some unplanned lessons to learn when I got kicked off the machine about ten times before I could finally generate the right sequence of job control cards for my FORTRAN program.

I also quickly got acquainted with debugging programs because even when my job was initiated, I'd get kicked off because of syntax errors in the programs I had written. It turned out to be a quite lengthy period of time before I finally got a successful program to compile, run and generate the data that I was interested in.

The really significant thing that came out of this was that I got a very pretty certificate of completion for the FORTRAN programming course. Naturally, I wanted this to reflect favorably in my Army records, so I mailed it off to my personnel file kept at the Pentagon. Thought no more about it for a number of years.

Computer Duty in the Army

In 1972, after completing a tour in Viet Nam and another two years commanding troops at Ft. Knox, Kentucky, it came time for me to do some duty in the Pentagon. The Army personnel organization did not have any jobs in the nuclear business at the time but here's where my FORTRAN Programming certificate came into play. In reviewing my records the personnel people decided, based on my computer programming experience, that I was eminently qualified to take charge of the Army's automatic data processing budget. I was going to deal with the budgetary process for the Army's procurement of computers and funding of software development programs.

I didn't know anything about software at this point in time other than the little, bitty FORTRAN, ALGOL and Basic programs I had written. At that point in time the Army was spending close to a 100 Million dollars a year to automate the business functions that they were performing - their payroll, their inventory management, their real property management on bases around the world, their mobilization plans, their force deployment plans, their command and control structure, and virtually everything that had anything to do with business and logistic operations in the Army. This had been automated on the first large- scale computers that came along. Most of this initial work had been done in Assembly

language. At the time I arrived it was just being done over in higher order languages, principally COBOL. We were in the midst of trying to get about 50 to 70 large scale systems completed and out doing their jobs in the active Army. All of this amounted to about 100 Million dollars of expense annually. Hardware was another couple hundred million.

In the midst of all this I began to hear those ominous words about over-runs and slippages. We were having a significant number of those occur with these large Army programs which were running from 50,000 to 400,000 lines of COBOL code.

I really became aware of problems with software the first time I went to the budget table with the people from the Office of the Secretary of Defense. We were going through the justification of our ADP program. The software aspects of this budget were only a third of the total budget and yet it was occupying almost all the time in the hearing. We were looking at the next fiscal year and the five following fiscal years after that. In the case of the Army's SIDPERS (Standard Installation Division Personnel System), we were looking at a system that had become operational the year before. When it first became operational it was up to 118 people; the next year we projected it to fall to about 116, then we were saying we needed 90 people for each of the next five years.

The budget analyst from the Office of the Secretary of Defense, rightly enough, asked "What are the people going to do? The system became operational a year ago. How many bases is it deployed to? Once it's out there, what are you going to use these 90 people for? Isn't it finished?"

Well, there was a big silence in the room. I was new, first time on the job. I didn't know the answer to this. So I looked to my left then to my right - - to those long-term Army civilian employees that had come into the Pentagon with the first computer and were the acknowledged experts. They were strangely quiet as well. Nobody on the Army side of the table knew what the real reason was for those 90 people. The very lame answer that finally dribbled out was "maintenance". So all the dialogue and conversation came to a halt, a big silence. Finally, the analyst from OSD said, "Look, this is a 10 Million dollar item. Unless I can get some satisfactory answers on this, I guess I will have to delete it out. Why don't we adjourn the meeting, [it was almost 4 o'clock in the afternoon] re-convene at 9 o'clock in the morning and perhaps by then you can call a few people that know more about it and we can come up with a satisfactory answer. See you in the morning."

So we scurried off, called the Army Computer Systems Command. We waited by the phone for several hours into the evening. Finally, we got a response back that was lame, inadequate. Upshot: Nobody in the Army really knew what the 90 people were going to do for the next five years, except "maintenance."

So, the next day by 9:15 in the morning we had lost 10 Million dollars from the Army's ADP budget.

The following day, while walking down the halls of the Pentagon away from the budget meeting, my boss, a Major General from the Corps of Engineers, said "You know Larry, this business of trying to plan the resources and the schedule for software projects is a very mystifying business to me. All my experience in the Corps of Engineers has been that we always, even early on in a project - - the big dams, the waterways projects - - we always had some feel for the physical resources required; how many dump trucks, how many cubic yards of concrete, how many shovels, how many people we needed to do the job. For these sort of things we could make some crude estimate, based on a little bit of previous data. Yet any time I try to get similar types of answers on computer programs, things having to do with software, I immediately get a dialogue on the internal architecture of the computer itself, the bits and bytes. Never anything about how long is it going to take, how much it is going to cost, how many people I am going to need, how good is it going to be when I deliver it. We need to have a way to address these kinds of things at our level here at the Department of the Army so we can come to grips with this business of planning and managing software development."

So, that was really the motivation that got me thinking about how software systems behave and how we might be able to effectively model that with a few parameters that would be useful; to get the answers that senior managers wanted.

Within a couple weeks of this budget disaster losing 10 Million dollars, by pure luck, I happened to stumble across a small paperback book in the Pentagon bookstore. It had a chapter on managing R&D projects by Peter Norden of IBM. Peter showed a series of curves which I will later identify as Rayleigh curves. These curves traced out the time history of the application of people to a particular project. It showed the build up, the peaking and the tail off of the staffing levels required to get a research and development project through that process and into production. Norden pointed out that some of these projects were for software projects, some were hardware related, and some were composites of both. The thing that was striking about the function was that it had just two parameters. One parameter was the area under the curve which was proportional to the effort (and cost) applied, and the other one was the time parameter which related to the schedule.

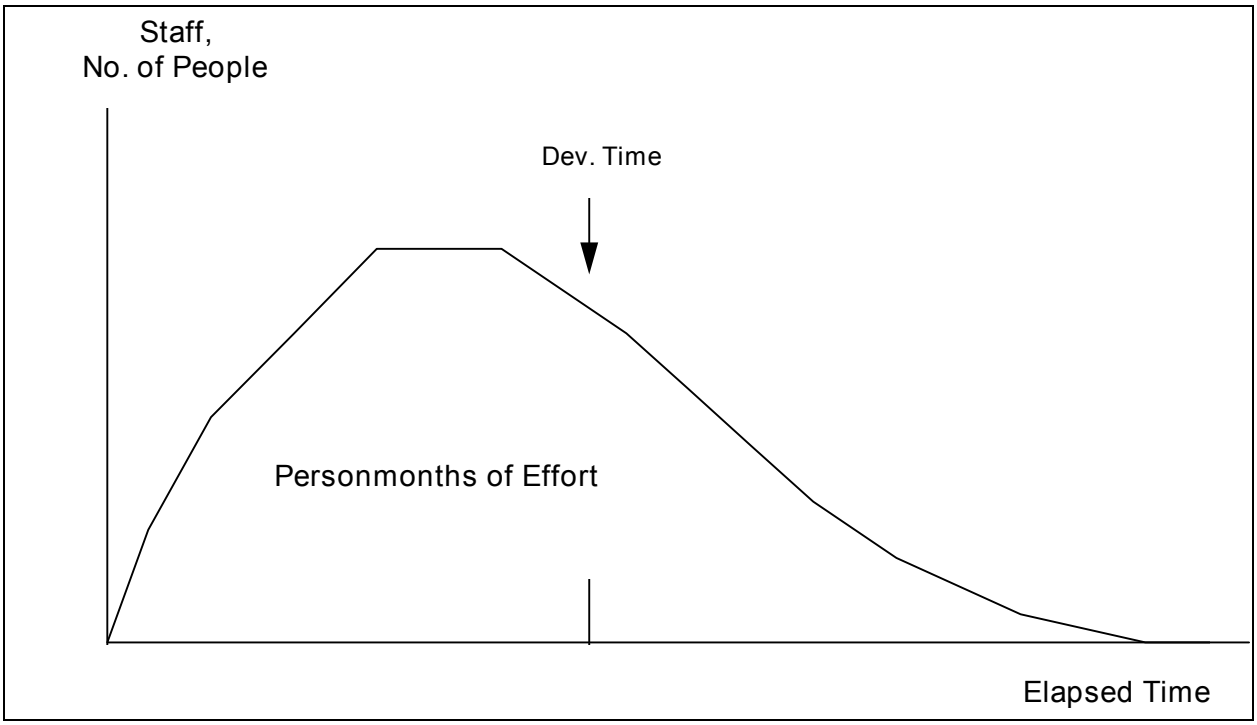

Figure 1. A typical Rayleigh staffing curve.

I found that I could take these Rayleigh curves and easily adapt them to the budgetary data that I had on the Army software projects. That data was collected by fiscal years. We had the data on the application of number of person years for each of the fiscal years throughout the history of any project in our budget. So, I went ahead and quickly plotted up all the software systems that we had in inventory and under development. I was able to determine what the Rayleigh parameters were in terms of the effort and the schedule, and once knowing that, I could make projections out to the end of the curve which was essentially to the end of the budgeting cycle. Within a month I was able to get about 50 large Army development projects under control and able to do credible budget forecasts for those projects - - at least five years in the future.

The next time the budget hearings came around a year later we were in the down sizing phase at the end of the Viet Nam war; budget cuts were endemic, we were asked to cut the application of effort on a number of existing systems. The turn around was short; we had to respond to what the impact this would have within a 24 hour period.

Now, having the Rayleigh curve and understanding that methodology, I was able, with a pocket calculator programmed with the Norden Rayleigh function, to quickly make estimates of what would happen if we reduced the projections for several of the projects. It was easy to show that the aggregate of these cuts would wipe out the capability to start any new software projects in the Army for the next 3 years.

The result of that budget hearing - we did not lose any money.

Naturally, the next important question that arose was, "How do I use these Rayleigh equations to generate an estimate for a new project? It's nice to be able to pick up those that are already underway, but is there some way I can find out the time and effort for a new project so I can build one of these budgeting and staffing profiles for getting the work done?"

I looked into that. Right away the notion arose that somehow we had to be able to relate the size of the Rayleigh curve - - its associated time and effort - - with the amount of function that had to be created. How do people building software for the Army - - in its own in-house organizations like the Army Computer Systems Command, and their contractors - - how do those people think about the functionality that they are creating?

I found out that they thought about the lines of code they had to write, they talked a lot about the number of files they were creating, the number of reports they were generating, the number of screens that they had to bring up - -.those types of entities that were clearly related to the amount of functionality that had to be created. Clearly these functional entities had to be related to the schedule and effort to get the job done.

I spent the next year and a half to two years of my Army time, about a third to half my daily schedule, doing analyses of data. The first set of data I worked with came from the Army Computer Systems Command. It was 15-20 systems. I attempted to do some mathematical curve fitting relating the size of those systems in lines of code, in files, reports, screens to the known development schedules and the associated person months of effort.

The first approach was to use simple regression analysis of functionality, lines of code, as the independent variable and then person months of effort as the dependent variable. I did the same thing with schedule.

Next I did some multiple regression analysis where I related effort to combinations of lines of code, files, reports and screens. The statistical parameters that came out showed they might be useful for predicting, but they were not extraordinarily good fits. Certainly more work and investigation was needed before drawing any conclusions.

By this point in time, 1975-1976, I had been in contact with other investigators working in this area. Judy Clapp from the Mitre Corporation had done some studies on 10 to 15 scientific and engineering systems that were being done for the Electronics Systems Division of the Air Force Systems Command at Hanscom Air Force Base. Felix and Walston at IBM Federal Systems Division had published a paper in the IBM Systems Journal and had amassed a database of about 75 projects that gave a good feel for a range of different parameters related to software. All of this information was very useful in trying to establish relationships between lines of code, pages of documentation, time, effort and staffing.

In trying to do this analytical work I had to go back about 20 years to my academic training at the Naval Post Graduate School to refresh my memory on how to do the

statistical analyses, how to do work with data, and to come up with some logical inferences and conclusions as a result of that. There was a lot of re-learning to build up skills that had become very rusty in the course of not using those techniques for many years.

Software Equation

One of the very promising experiments was trying to do some multiple regression analysis relating the size of the systems in lines of code to the schedule and the person months of effort applied. I did these curve fits first with the Army data, then with the Electronics Systems Division data followed by the IBM data. I was lucky in that I got some very nice fits in about 20 of the Army data systems. Concurrent with these curve fits, I also did some theoretical work on integrating Rayleigh curves and tried to establish the parameters of integration from a little bit of the historic data from the Army and IBM. I found there was good consistency in being able to generate the key parameters for the Rayleigh equation - - the work effort (area under the curve); and the schedule parameter. These different, independent approaches at getting a parameter estimating equation were both leading me in the same direction and producing similar results.

What ultimately fell out of this is what I now call the [QSM] software equation. It related the amount of function that had to be created to the time and effort required to do it. It looked like this:

Quantity of Function = Constant x Effort x Schedule

In the curve fitting process, I found that there were exponents associated with both the time and effort parameter and there was also a constant that got generated in the process. I thought a lot about what the physical meaning of this constant was. Somehow it seemed to be related to the efficiency of the development organization, or the technology which they were applying to their software development practices. The first name I used to describe this empirically determined constant was a Technology Factor. I used that term in the first early papers I wrote on this and published by the IEEE Computer Society in the 1976-1977 time frame. I have continued to use that parameter to represent the efficiency of the software development organization. I have, over the years, re-named it several times - - first a Technology Factor, then to a Technology Constant, then a Productivity Constant. Our most recent name for this is the Process Productivity Parameter which I think is probably the most descriptive term to use in describing what its real relationship is to the software development process.

This was the genesis of my software equation which we still use today. It has proven to be a very good measurement and estimating tool for me and QSM over the past 20 years.

This software equation was a macro model that linked the amount of function to be created to the management parameters of schedule and effort required to produce it. The empirically determined factor represented the productive capability of the

organization doing the work. This also suggested that this was a very good way to easily tune an estimating process because if you knew from your completed historic projects, their size, time and effort, you could easily calculate what that process productivity parameter was. Then you could use this calculated parameter in making an estimate for a new project. So long as the environment, tools, methods, practices, and skills of the people did not change dramatically from one project to the next, this process of playing back historic data became a very useful, simple, straight-forward calibration tool.

Manpower Buildup Equation

The other key relationship that emerged in these studies was the direct relationship between time and effort. Clearly, these were the parameters of our Rayleigh equation. But was there anything we could learn from the basic data as to what this behavior characteristic was? Again, more curve fitting. I found that there was a distinct relationship between the effort and the development schedule. It turned out that the effort was proportional to the schedule cubed. This was also something that had been noted and discovered earlier by Felix & Walston and several other investigators who had done research in software cost estimating.

This finding was especially important because it now gave me the basis for making estimates. I had two equations and two unknowns. The first was the software equation involving size, time and effort and the process productivity parameter. The second equation (which I now call the manpower build-up relationship) linked effort with the third power of the development time.

This latter equation required some parametric determinations. We found a parameter family that seemed to relate to the staffing style of the organization. Those organizations which tended to use large teams of people build up their staffing rapidly. This produces a high value of the ratio of effort divided by development time cubed. Those organizations that worked in small teams took a longer period of time to complete their development. This was typical of engineering companies that tended to solve a sequential set of problems one step at a time. For such companies I saw that the relationship of their effort divided by development time cubed produced a much smaller number for the buildup parameter. This was telling us is that there were different staffing styles that organizations adopted. The parameter of effort divided by development time cubed was really a measure of the manpower acceleration being applied to a software project.

It became evident in studying these different types of organizations that those organizations that use large teams of people seem to finish their projects a little bit faster than those that used the small teams and took a little bit longer to complete their work, all other things being equal. This suggested there was some sort of trade-off going on between team size and how long it takes to get the work done.

The other significant idea that I started working on during this period in the Army was the notion of the Rayleigh curve being a good process control vehicle for projects while they were under way. The idea was to take real data as it was happening on a project and then update and adaptively forecast the right Rayleigh curve to project onward from wherever the project was at the time. This would let you dynamically predict cost, schedule and defects to completion of the project.

I did some early curve fitting investigations but ran into some problems and snags that prevented this idea from being fully realized at this point in time. Nevertheless, there was enough work and enough positive results to suggest that this should to be pursued and had considerable promise. One other consideration that was evident at this time was that not many people were interested in dynamic control. Most organizations were having so much trouble trying to come up with the initial forecast that the idea of learning about how to control an ongoing project was not high on their priority list. This suggested that trying to reach good solutions in dynamic measurement and control was premature.

Duty with Army Computer System Command

After my tour of duty in the Office of the Chief of Staff and with the Office of the Assistant Secretary of the Army, Financial Management in the Pentagon, I was transferred to the Army Computer Systems Command at Ft. Belvoir where I continued to work on the software estimating problem and some useful ways in which the Army could apply these techniques. I wrote several papers during this time which were presented at forums sponsored by the IEEE Computer Society. There appeared to be a growing interest in software estimation.

These papers produced contacts with other people in this area, such as Laz Belady of IBM and Manny Lehman from City College in London. Manny suggested the idea of doing a conference of interested parties to discuss the progress that was being made. I was able to get the Army Computer Systems Command to sponsor such a meeting which was held in the Spring of 1976 at the Airlie House in Virginia. 50-60 participants from government and industry came together and presented their latest thoughts and observations relating to software estimating. Peter Norden was able to come and present his latest thoughts, as well as John Gaffney, Ray Wolverton and other notables in the field.

During most of this period of time I continued to do small scale modeling experiments using programmable calculators from Hewlett-Packard as the computing device. I found this was preferable to time sharing services and using the large scale computers available from the Army because it was so easy to conceptualize things on paper and immediately write a small program to implement it and see what the behavior pattern was. Moreover, I was not at the mercy of programmers and waiting for them to write the programs that were needed. I could do that myself, take the machine home and continue my work into the evening without interruption. I found this a very efficient way to build and test algorithms. When the algorithms worked on the HP calculator they could immediately be

implemented bug-free in FORTRAN or Basic on a medium to large machine for a more comprehensive analysis.

During the 1976-1977 time frame, the Department of the Army decided to take the early work I had done on determining parameters for the Rayleigh curve and put those ideas out for broad distribution throughout the Army. They did this in the form of a Department of the Army pamphlet. These parameter estimators were based on the number of files, number of reports, number of screens to determine the effort and the time parameter for the Rayleigh equation. This happened about the time that I was doing most of the development and testing work on the software equation, but it wasn't ready for full scale implementation and use at that point in time. The Army decided to proceed ahead with their information pamphlet using the earlier work.

During the first half of 1977 I did considerable development work on the implementation of the software equation involving the linking together of the effort and time parameters of the Rayleigh equation and relating those two management parameters with the amount of functionality that had to be delivered. I did extensive work with a family of parameter values for the technology factor, later called the process productivity parameter. I also tried to learn over what range of software application types this factor pertained to. We had some reasonable data for the Army's business systems, some reasonable values for some of the Air Force Command & Control and Radar Control projects from Hanscom Air Force Base and some engineering applications from the IBM FSD suite of data. Much of my research time during 1977 was devoted to validating and testing reasonableness ranges for parameters that seemed to describe these different application environments.

GE Days

At the end of August 1977 I retired from active duty in the Army and shortly afterwards went to work with the Space Division of General Electric Company in their office located in Arlington, Virginia. I was with General Electric a little less than a year but continued to develop the model, apply it to estimates that were used by General Electric in some of their software development activities related to the space program, and a considerable amount of time working on estimates related to large scale manufacturing software that was being built in support of the GE Aircraft Engine Division in Evendale, Ohio.

One important aspect of the software equation arose during this period when I established contact with Professor Victor Basili at the University of Maryland. He was doing work with NASA Goddard Space Flight Center and was one of the principles in their software engineering laboratory. In my conversations with Professor Basili, I found that they had collected a substantial body of data related to scientific data reduction systems for satellites being launched and controlled by NASA. Most of this software was being written in FORTRAN; the size regime was significantly smaller than the systems I had been working with in the Army,

notably in the range 15,000 to 80,000 FORTRAN lines of code. I observed that the Rayleigh staffing profiles were quite different than I had seen in the Army. Most of the Army Rayleigh curves reached their peak staffing at about the time the software was ready to ship. The NASA Goddard software reached its peak staffing intensity considerably before the software ship time. By then the staffing profile had come down and the number of people was at a relatively low level at the time the software went into operational service. This suggested that the Rayleigh model was more complex than I had originally thought and would require an additional parameter that would control where the peak of the Rayleigh curve occurred as a function of how big the software product was.

So I spent some time providing an additional parameter in the software equation to control this peaking and tried to interpret what the physical meaning of that parameter was. Ultimately, I came to call it a complexity factor. It seemed to kick in when the size of the system was in the neighborhood of 15 to 18,000 lines of code. Most of the change in the Rayleigh shape occurred between 18,000 lines of code and 70 thousand lines of code. Beyond that size I found that the behavior of the Rayleigh shape seemed very much the same as that I observed in the Army. That is, it reached its peak staffing at about the time the software product was ready to put into operational service.

During the time I was at General Electric I met Ann Fitzsimmons who was a FORTRAN programmer working for GE. She was very helpful in building some early prototype models of the Rayleigh equation, the software equation, and simple prototype estimating models. Ann became interested in pursuing an opportunity to turn these ideas into a commercial product. She suggested that we leave General Electric, organize a company and try to build a commercial estimating tool. This was based on her observation that there was considerable interest in such a product, because we had been getting a number of inquiries and phone calls, both within General Electric and externally from other companies that we had met through conferences or through papers that she and I had written individually and which had been published in the technical and trade press.

So while I was mulling over these suggestions from Ann, I was developing a business plan and looking at how I might raise money to form up a company and start a business from the ideas we had come up with for estimating tools. Ann precipitated the whole thing by announcing that she had quit GE and was ready to go to work full-time building this system. It took me about two more weeks to make up my mind, resign, and start the process of getting Quantitative Software Management Inc. underway.

Noisy Data

My Army and General Electric experience pointed out rather clearly that there was a great deal of noise in the data we were collecting. In particular, organizations were unable to report the time more accurately than a whole month. Sometimes the person months would be rounded off to the nearest 5 or 10 person months. This

suggested that as I considered the notion of creating commercial products, this had to be researched and addressed. I would have to learn how to deal with this uncertainty and be able to reflect it in the answers and risk buffering that would need to be done.

QSM Days

The first version of SLIM® was built on a DEC System 20 time sharing machine belonging to American Management Systems. It was written in FORTRAN by Ann Fitzsimmons and was about 8,000 lines of code. It was designed to be as simple as possible to use and require a minimal amount of user supplied input information.

The notion of being able to specifically tune the model to the development organization was very much on my mind at this point in time. So, what we did was include a history (calibrate) function in which the user could collect a little historic data - - as simple as the number of new or modified lines of code, the development time and the person months of development effort that had gone into producing a completed project. This could then be put into the software equation and a process productivity parameter determined from this.

We also decided that we would use the concept of a Productivity Index, a simple series of integer numbers from 1 to 18 as a surrogate for the engineering family of numbers called technology factors. This was designed to make it easy to use and simple for managers to understand. Once the user had a Productivity Index from 3,4,5 or more previous projects and was tuned to the right environment, then all they had to supply was the SLOC size estimate in the form of a range - - the smallest, the largest, the best guess. The user could either do this for a whole system, or break it up into a number of subsystems and SLIM® would roll up the individual subsystems for him/her.

The earliest version of SLIM® generated a minimum time solution. This was the fastest possible solution. It would also be the most expensive. Uncertainty was present; therefore, it was important to do a Monte Carlo simulation where the uncertainty that existed with respect to the size and the Productivity Index could be factored into the simulation and mapped through to the outputs of development time, peak staffing, development effort and cost. These uncertainties in the form of standard deviations could then be applied in a risk sense so that if someone needed a 95% cost biased number to quote to a prospective customer, that could be easily obtained. SLIM® provided this in the form of risk tables. Similar capability was provided for schedule and staffing profiles.

The software equation provided a solution in the form of time-effort pairs. There were a range of different staffing profiles possible. If one took a little longer time, a smaller staff could be used; This let the effort go down. Since the relationship between time and effort varied as the fourth power of schedule, a small change in time produced a big change in effort. This appeared to be exploitable by management so within SLIM® we built a range of 'what-if' functions that had a

great deal of power in being able to use the concept of management constraints and trade-off. For example, let's say the minimum time solution turned out to be 21 months and 388 person months of effort and built up to a peak staffing of 29 people. But we might find out that only 15 people were going to be available to work on this project. The implication here is that it will take a little bit longer time and quite a bit less effort if a solution could be found for a peak staff of 15 people.

We provided a trade-off function that allowed the user to directly input a peak staff of 15 people and the output produced the appropriate time-effort pair associated with that staffing profile(250 PM, 25.7 Months).

A whole range of such trade-off scenarios were presented - - design to schedule, design to cost, design to peak manpower, design to a certain level of risk on schedule, etc. All of these were very simple trade-off relationships in a time sharing environment. A user picked the appropriate one from the menu and was then prompted for the appropriate inputs; the computer immediately calculated the solution and brought it up on the screen or printed it out onto the terminal.

With this version of SLIM all of the output was in tabular form. There was no graphics capability. There were no remote output devices that could handle data quickly enough to make high quality graphics a feasible capability to provide. This was because at this point in time we had 300 baud terminals which were very slow in graphics mode. Similarly, plotters were hardly able to be serviced at these data communication rates.

We started building the first version of SLIM in September of 1978. We had a working proto-type version of it by December of 1978 and were able to provide beta versions for prospective customers by January 1979. The system ran well on the DEC System 20. It was accessible from anywhere in the country through Telenet and the reliability of the system was quite high. The time to solve a problem using this system was typically 30 to 45 minutes to get every answer that one could think of.

A family of implementation functions were provided as output. Once a user made a decision for a schedule, effort and peak staffing that they desired for their solution, then complete staffing profile month by month, including the high level functional front end and the main software build and the maintenance work could be provided throughout the entire life cycle. The person months of effort could be given in this same tabular form. If you wanted the cost for just a single fiscal year, you could get that cost. If you wanted the cost over the whole life cycle, that was possible. If you wanted just the development effort for the main software construction phase, that was possible as well. It was a very flexible tool. Anything that you wanted as a function of elapsed calendar time, month by month, was possible to obtain.

HP 85 Version of SLIM

We used the DEC System 20 for about two years. About that point in time the new HP-85, typewriter size, desktop machine with built-in Basic language became

available. This appeared to be an ideal piece of hardware for us to build a portable version of SLIM. Going from FORTRAN to Basic was not a problem. The chief advantage one had with the HP-85 desktop computer was that it had built in graphics. It had a small-format, built-in printer as well, that had the capability of turning graphics output on its side and printing out a very acceptable, readable graph of all of the functions that were important in the SLIM environment.

About 1980-1981 we came across a body of data from the Rome Air Development Center that had defect information that led us to construct a reliability model and that was tied in with the schedule, effort and size of the software product. This gave us a capability to make a reliability prediction model that would estimate the defects that should occur in development. Defect prediction was useful from as early as the start of systems integration test out to the time the product was delivered to the customer, and even out into the operations and maintenance phase after that. The reliability model seemed to be very Raleigh-like so we built it as a time-based Raleigh defect model.

We discovered that the defects were very closely correlated with the productivity index, such that if an organization was very efficient (a high productivity index) then it also had a correspondingly small number of defects. The other interesting observation was that if you used a large team of people (a high manpower build-up profile) then you also got a correspondingly large increase in the number of defects. We built the model in such a way that the projected number of defects would scale corresponding to these parameters - - the staffing profile and the a productivity index. We included the reliability model in our first HP-85 version of SLIM.

The HP-85 was very easy to use, very easy to write code for, and easy to make modifications to code. The quality of the output was not ideally suited to business users needs. The 32 character format for the tables was not standard; but soon HP came out with the HP-86 and 87 series of computers which had the same processor and the same Basic language but had the capability to provide full-page, 80 character, output and a large scale graphic capability driven out to pen plotters. Soon, very high quality output was available from these companion pieces to the HP desktop line. We also found that these desktop machines got around people's concerns about storing their proprietary data on a time sharing machine where they worried about their competitors gaining access to their data.

The reliability function gave good results from the outset and has been an excellent predictor of what one might expect in the way of defects month-by-month while development is going on. It gives a good indication of when the product is going to be good enough to put into operational service.

I felt this was a very important because it got development organizations away from the strong tendency to deliver a product early before sufficient defects had been removed from the product. When this happened the product did not work well or run long enough to do its job. In most cases this was because the product had not been sufficiently de-bugged and tested to guarantee good levels of performance and

reliability. With the reliability function we now had the capability to measure this while development was going on and let people control defect elimination far better that they had been able to do previously.

Linear Program

I also developed the notion of being able to apply a linear program to the software estimation model. The idea is we have the software equation and we have an array of different constraints. We could use the constraints in the same way people did in a linear program to bound a feasible region. From this we got a minimum time solution which was also the most costly and had the lowest reliability. At the other end of the spectrum we got a minimum cost solution which took the longest time and had the highest reliability. In between these two solutions were all other feasible solutions. One might choose to adopt one of these in between solutions because of other concerns.

So Linear Programming became an extraordinarily powerful tool to identify this feasible region and then pick and choose other possible solutions within this range, depending upon the priority of management constraints. It was important to be able to tie management constraints into the normal operating routine in solving estimating problems. A picture of the linear program in the two management dimensions is shown in the following diagram.

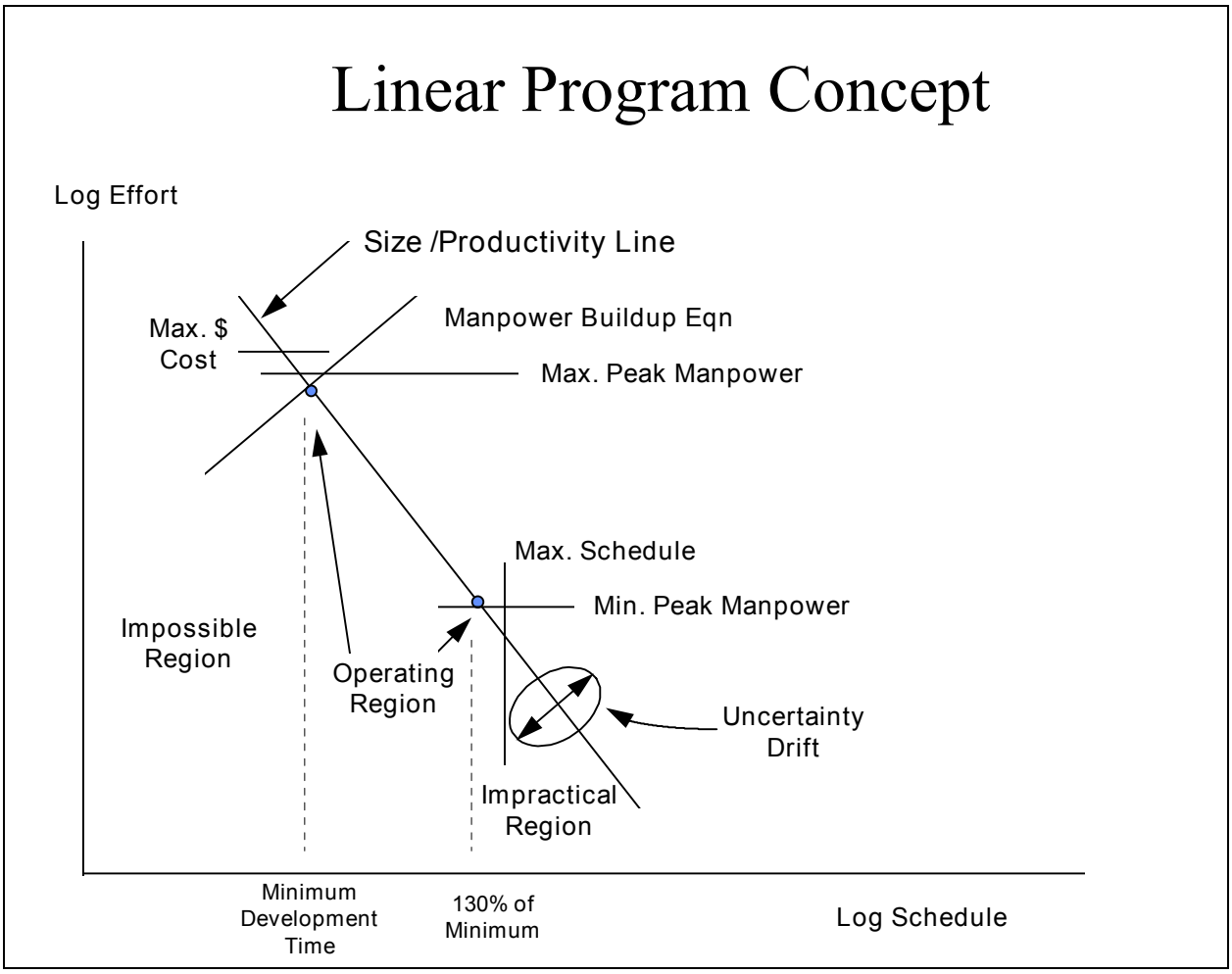

Figure 2. Linear Program concept.

The HP-85 capability with graphics also led us into the notion of looking at the historic data in a graphical format. We found that we had a fairly substantial database of completed projects by that time, maybe 500-600 systems. When we put those all into the history mode of SLIM we found that there was a nice pattern that emerged. We identified the spectrum of person months of effort as a function of system size that had gone into all the historic projects; we did the same thing for numbers of people working on projects, and the development schedule. These graphs were very informative in outlining what the feasible region was. We also put statistically generated trend lines on a log-log scale through these data points. We could take away the historic data points, leave the trendlines in place, and then take a small set of data - 3,4,5 projects from a specific organization and plot them against the trendlines. From this we could see how that organization behaved with respect to the industry average. For example, if a particular organization was taking longer than the industry average, was costing more, then we found that their productivity index was significantly lower than the industry average. This let us say "We see that this particular developer is not as efficient; he takes longer, he costs more, he generates more defects."

This graphical technique gave us a quick way to compare development organizations with each other and with the broader context of the whole industry represented by the trendlines of our large database. This concept became the basis for our next product, PADS, the productivity analysis database system.

PADS® - Productivity Analysis Database System

We started work on PADS® about 1982. We introduced it into operational service in 1984. It was a desktop based system, initially done on the IBM PC under DOS and it utilized the graphics capability that was provided by the IBM PC family.

PADS® provided the capability to keep track of an organization's productivity over time by continuously computing the productivity index and being able to associate that at particular points in time and with the environment, the tools, the skills and the management capabilities of an organization. So, beyond being able to say this is the productivity of XYZ Corporation, PADS would let you say these are the environmental influences that probably had a large impact on why XYZ Corporation is where they are now.

With PADS it was possible to study the introduction of new technology, new products, new management practices and see if those produced an impact on the productivity index. For example, we studied a number of developers when they went from developing in batch mode in COBOL to an on-line interactive development environment. The immediate impact was to boost the productivity index by 2 to 3 integer values - - a very significant reduction in the management parameters. For example, 3 productivity indices would bring down development time by about 26% and cut the development effort by a factor of 59%.

PADS provided the capability for long term studies of software behavior. It became a repository for an organization's software metrics data. It had an excellent capability to do economic analysis - - to see what the high pay off investments and practices.

SIZE PLANNER

By 1984-1985 we found that we were getting considerable number of questions related to sizing. People were happy with the results they were getting from SLIM on schedule, effort, cost and implementation plans but they still had trouble coming up with the estimates of size that needed to be fed into SLIM as input.

In my consulting experience I had come across a number of ways of approaching this. I decided it would be good for us to develop a sizing model and include it with the estimating package, SLIM for people that felt they needed it. A large number of customers had no problem with sizing. So, there was no need to get them to buy it when they had no need for it. That guided us into making it a separate product. We would sell it to just those people that felt they needed some supplemental help in coming to grips with the sizing input to SLIM.

We built an independent, DOS-based product for the IBM PC which we called SIZE PLANNER. It was a fairly small utility of program designed to come up with combined estimates from a number of different approaches.

For example, a user could make estimates in function points, lines of code, programs, and sub-systems. Or, very early in the process, the user could use fuzzy logic sizing based on historical data in QSM's database. SIZE PLANNER could even handle re-used code, some of which was modified, some of which was unmodified. SIZE PLANNER would convert this all into an equivalent line of code estimate in the form of a Low, a High and a Most Likely that was suitable to input into SLIM. It was introduced for sale in 1987.

SIZE PLANNER did a good job. I used it quite frequently in consulting. A good number of our customers that felt they wanted a cross-check for their size estimates, or did not feel comfortable with their existing methods, adopted it and found they got excellent results. Essentially, SIZE PLANNER was a front-end for SLIM, or other estimating tools.

SLIM-Control®

By 1986 there was considerable interest in being able to control ongoing projects. Many organizations would start a development and then find they couldn't stay up with the original plan formulated, or they couldn't get the people as fast as they anticipated. After the first 30% of the project they would sense that they were falling behind but didn't have any idea how much. They didn't know when the project was really going to be completed. If they could get all the people they wanted, could they throw more people on it and catch up? These were the problems that we wanted to be able to address with a new product which we named SLIM-Control®.

The concept was to use statistical process control ideas from the manufacturing area of expertise and tie it together with the notion of adaptive forecasting. This would let us feed in the actual data coming from the project, compare it with the plan, and then generate a new plan that would realistically project when the project was going to get there, what the staffing should be, what the cost would be to that new end point, and make a reliability prediction as well. All of this would be consistent with the underlying SLIM model.

SLIM-Control was difficult to do. It was hard to get the algorithms to work well together. It was also difficult to model the impact of playing "what-if " games. For example, if I am half way through a project and I know I am falling behind where the predicted slippage is 2 ½ months, what would happen if I added an additional 15 people to the project? How much of the lost time could I make up?

These were the issues we were trying to model. Since everything was happening dynamically month-by-month, the algorithms had to be responsive to where you were along the time line. The model had to have different sensitivities at different points in time because things that were done earlier in the process would have far

more impact. Adding 15 people after all the code was written doesn't help much; there is not a great deal one can do to influence the project and play catch-up at that point in time.

So, it was not a simple process to model. It took us about 3 years to complete the development of SLIM-Control. When it did get out into our user community, we found it was well liked, did an excellent job of prediction and filled an important void in the ability to deal with software development projects once they were underway. It suggested intelligent things to do that could be quantitatively determined from what had happened so far. SLIM-Control came on the scene in 1989. This was the last completely new product introduced within the QSM family of tools to date.

Windows Products

By the early part of the 1990s we saw some new trends that suggested major changes to our product line. Most important were the advent of the Windows operating system, higher resolution screens and much faster processors. Suddenly the opportunity to make very major improvements to the user interface and provide higher quality color graphics output became possible. That became our next goal -- take our product line and convert it over to the Windows environment and make the user interface much more intuitive and natural for the user.

All this meant that the user now had the opportunity to solve the problem the way he wanted to approach it, not in the serial, sequential fashion that we had forced him to do with the old DOS products. With the DOS products the user had to think the way the developer did. He did not always do that naturally. It presented a frustrating dilemma at times. Windows, VGA resolution and the high speed of the new processors provided the ability to get much more information on the screen. With a mouse the user could attack the problem in any way he liked, his own way. He could solve the problem in a way that made the most sense to him. That was the driving motivation for how we did our products over in the Windows environment.

With the Windows version of SLIM we were able to innovate by providing new ways for analysts to look at their solution and compare it with the industry trendlines taken from PADS. The user could even superimpose his own data on the industry trendlines. This could be useful when a marketing manager proposed an impossibly short schedule. When this solution was generated with SLIM (perhaps requiring a productivity index that was 4 or 5 values higher than the organization had every done before) it could be shown graphically and the probability of this solution could be seen right on the screen. The very short schedule would be very much out of line with anything the organization had been able to do before. So, with the graphical view it became immediately obvious that this was not a good solution. It was not likely to happen.

Similarly, we were able to put a graphical consistency check in for schedule, effort, staffing and defects. This let the analyst and his boss look at the solutions and see if they were consistent with what other people were doing as well as with their own historic data.

In addition to this there was considerable interest in doing sensitivity profiling in the form of "What-Ifs." The question might come up "what if this product were to grow by an additional 25,000 lines of code? What would that do to our schedule? What would that do to the cost and the effort? We were able to put in a sensitivity capability to look at these issues graphically and see what the incremental differences were. We could do that in terms of size, productivity index or staffing. All of this provided a very quick capability to explore 'what-if' situations that might be posed by management.

The other major Windows innovation was putting dynamic risk gauges on the problem solving screen. If one wanted to play "what-if" on the schedule one could grab a handle on the staffing diagram and make it longer or shorter. In the lower left hand corner of the screen we were able to display a probability scale for each of the key management parameters -- schedule, effort, maximum peak staff, minimum peak staff and Mean Time To Defect. As the user changed the plan with respect to schedule and its effect on staffing and effort, he could instantly see the way in which the risk gauges moved. So, if I happened to have a 24 month constraint on schedule and I stretched the schedule out, I would see my probability gauge for schedule go down. This gave us a capability to keep the different risks balanced; being able to see it visually all at one time gave a much better appreciation of the interaction of these different management impacts.

Summary

I have been involved in software estimating for 20 years. Much has occurred in the field over that time. I have found it exciting to try to understand the software development process and how to model that. Thinking back, I believe looking at it as a time based, parameter-driven, macro-model has been a good approach that has been close enough to reality to produce excellent results with enough flexibility to adapt to the new development paradigms that have arisen. I am confident that it will continue to do so for a reasonable time into the future.

I have found that the software estimating and control problem was a lot more complicated than I originally thought it would be. But that has kept it challenging and interesting.

Moreover, it has been gratifying to have worked with major corporations and government organizations in the US and abroad; to have tackled some of their tough problems in software planning and management and to have been able to contribute to solving their problems and to have had a positive impact on their business. This had made all the hard work worthwhile.

Publications

Publications which I have written or have been co-author include the first book done for the IEEE Computer Society entitled "Software Cost Estimating and Life Cycle Control: Getting the Management Numbers" published in 1980. This was done with Ware Myers who has been my writing sidekick since that time. This book was an assembly of my key thoughts on software estimating at that particular time, along with a collection of germinal papers on software estimating either done by myself or other authors.

The next book which I co-authored with Ware Myers was Measures for Excellence: Reliable Software On Time Within Budget published by Prentice Hall in 1992. This was a complete statement of my entire philosophy of estimating control and measurement as I understood it at that point in time, 1990-1991. It also included some capability to implement the QSM techniques by manual methods, with a calculator. Also shown was an extensive collection of computer generated output from SLIM and SLIM-Control as an illustration of the way in which we have implemented the key functions in our commercial tools.

The most recent publication is a management oriented book done by the IEEE Computer Society entitled, "Executive Briefing: Controlling Software Development", again by myself and Ware Myers. This was published in 1996. It's a 12 chapter book focused at senior management. The objective was to present the key ideas with a minimum of mathematics and diagrams, with just the key management concepts to get across the idea that it is possible to control software projects. That there are a few simple things you need to know about how software projects behave. These concepts are straight forward and work with all the traditional, straight-forward, well-understood management practices.

In addition to these books I have written a large number of papers, by myself or jointly with others, related to specific topics in software estimating. Copies of these are available from Quantitative Software Management, 2000 Corporate Ridge - Suite 900, McLean, VA 22102.