A successful project performs as required, is completed when promised and is implemented within a promised budget. Estimating software development projects is an "art" that has plagued software developers since they began to write code. "How long will that take?" "Why did it take so long?" "When will this project ever be completed?" Numerous books have been written on the subject, and with the completion of our own individual projects we are usually ready to prepare our own "lessons learned" on the topic.

We are always looking for a better solution to estimate the cost and time for developing software. We have used the "seat of the pants method", the "prior experience model" as well as the latest in PC-based project estimation tools. This panel discussion gave software developers a chance to share their own experiences and shed light on some new approaches to deriving better estimates. Three SAS software experts joined the panel to share their current experiences, successes and failures.

Panel Members:

Joel Hoffman is Senior Coordinator, Clinical Information Systems at Pfizer Central Research. He is responsible for coordinating the development of a clinical trials reporting system over Oracle and Rdb databases. This is an international effort requiring the scheduling of resources and the implementation of software between Sandwich, England and Groton, Connecticut. He focuses his discussion on "why" cost and time schedules are difficult to estimate.

Jeff Polzin is the manager of Core Development at the SAS Institute. He is responsible for the components of the SAS System that build the SAS supervisor, the Data Step Supervisor and the Display manager. He presents some background information about how SAS software is developed, and the events that make estimating, even a schedule, a difficult task. At the SAS Institute, the only constraint is time.

Judith Mopsik is the Operations Center Director for Software Applications and Training at Atlantic Research Corporation. Her staff not only develop SAS application systems, but also have the responsibility for providing clients with the cost and schedule estimates for developing these software systems. Producing results on time and within the cost estimate are the major challenges to this organization. She begins the discussion with a formal introduction to estimating project scheduling.

Judith Mopsik: Project Estimation

Project estimation is based on a plan. The plan has a discrete start, finish, and scope. Building a house, publishing a book, and even planning an event are all examples of projects requiring some method of estimation. All have identifiable beginnings and endings, and are comprised of a series of finite tasks.

The functions of project management center around planning the work, scheduling the work and implementing the work. Hence, a successful project performed as required, was completed when promised and was implemented within a promised budget.

In the planning phase, we identify the general requirements of the project, break it down into discrete activities, determine the sequence and interdependence of the activities (what activities must be completed before other ones can be initiated), estimate the resources required to do the work (who will do that work; what equipment will be needed; what outside support/suppliers will be required), and find the "critical path" for completing the job. In the scheduling phase we attach actual calendar dates to the activities and determine the overall project dates.
In the implementation phase we employ a variety of techniques to juggle resources, monitor progress and costs, and ensure that the critical activities are completed on time.

One key to a successful project plan, and the ensuing estimates, is a comprehensive Work Breakdown Structure (WBS). A WBS is organized hierarchically, with each succeeding level depicting the project components in more refined detail. The top level generally defines the tasks in terms of managerial areas of responsibility. Specifically, what are the major project areas. The next level shows discrete activities required for each general area of responsibility. The final level details the specific work or activities required to meet the project goals. If the WBS extends down to the level at which work is actually performed, the estimates have a higher probability of being "close". Using the foundation of the WBS we decide how much time and what resources will be required to complete the job. Finally, probabilities can be assigned for achieving both sets of estimates. This will determine the probability of success.

Compromises must be considered as new events occur to change the original "assumptions". In developing software, quality is the critical measure. It cannot be compromised. Hence, "features" or promises must be scaled back if the resources change, the budget is cut, or the schedule is shortened. A successful plan begins with the "details". You can then optimize the schedule, reduce costs, or change your assumptions. You cannot know where to compromise until you know what all of the activities are. You then have a basis to discuss, challenge, accept or ask for even more detail. Finally, you can assign a probability to each assumption and make an even better estimate.

As you can see, there is an infinite set of possibilities that can enter into a cost or time estimate. It is not an easy job, and there is no single, correct answer. The process becomes the basis on which you can apply your best estimate.

Jeff Polzin: The Time Estimate Process for SAS Institute Product Development

SAS Institute spends approximately 45 percent of its gross income toward the research and development efforts of the SAS System. Individual product development is not separately budgeted, as it is in many corporations, but it is supported by an aggregate of the R&D budget. All development groups have available to them identical and common hardware resources. As previously indicated, because of the unique focus on research and development funds from a common financial pool, budgetary cost estimates associated with product development are not a major burden normally included in estimation of project development efforts in other corporations. Time is the most important factor that we face in determining future product availability.

To gain a perspective about the time estimate process for the SAS System development, one must understand the organization of the development staff, the distribution of the SAS System on hardware platforms, and the determination process for changes and enhancements of the SAS System.

The development staff can be separated into three levels of responsibility utilizing different project management methodologies. The model is closely related to the inverted triangle structural representation of the SAS System. Each tier represents a distinct layer of responsibilities and relationships required for the research and development of the SAS System. These tiers are referred to as the Applications, Core, and Host components of the SAS System. Information about resource and programming requirements generally flow from top to bottom. Refinements of those requirements flow from top to bottom and bottom to top.

The top tier comprises the SAS application product areas. These areas include statistics, basic procedures, SAS/IL, SAS/Graph, SAS/OR, SAS/EATS, and full-screen applications among others. They are organized and operate similar to academic environments, which, for us, has allowed for greater creativity, independence, and competition among the individual departments within the applications product areas. This competition, however, does not include competition for financial resources. Most of the application product development departments utilize the process of prototyping to extend a current product area or to create a new product. Often a new product develops as a special interest project by one or more staff members. Most of the application groups have testers associated with the development of their applications.

The middle tier of responsibility is the Core layer, composed of two development departments, CORE and IDB. The CORE department is responsible for
overall system architecture and the development and maintenance of the supervisor, user interface, and DATA step components of the SAS System. The IDB (Internal Data Base) department is responsible for the SAS data set I/O engine supervisor supporting the basic engine components and the SAS/SHARE product. Both departments in CORE are managed separately but they are dependent upon each other to provide the required interfaces for the other two tiers of the SAS System.

During the developmental cycle, a testing group works with the Core layer to assure the integrity of the code under development. A combination of prototyping and formal design methodologies are utilized when developing changes and enhancements for the Core tier. This area's primary role is to provide a "portable operating system" for the Applications use.

The bottom tier is the HOST. The responsibilities in this layer can be separated into two areas. The first area is the hosting departments that are responsible for providing the host platform support upon which the SAS System rests, the interface with the operating system, and the hosting and initial testing of the portable code. The portable code is the code developed by the Application and Core layers on the development machine.

The second area of host responsibility is the Quality Assurance department, containing individual sub-departments responsible for testing specific product and machine environments. The QA department is administratively part of the host department but its association provides a better opportunity for a closer relationship with all of the research and development staff.

An important note about the QA department should be addressed here. When a release is delayed for delivery into the field, the delay has been generally blamed on the QA department. This blame may originate internally at SAS Institute or externally from our user community. Fortunately, the QA department exists to insure that the SAS System is delivered to our customers with the fewest number of defects.

For most research and development environments, product delivery concentrates upon a small domain of end points. These end goals may include a single machine environment or a small end-user group for a product with specific purpose. The SAS System, however, targets a complex array of machine environments with a varied assortment of products for a wide complex and varied assortment of end-users. An awareness of the complexity of machine environments that the SAS System supports is needed.

The host groups must implement support for a variety of hardware platforms. Some of these platforms have common characteristics and others are very distinct in nature. On the IBM side, we must support MVS, VM/CMS, VSE, and AIX including non-XA, XA, and ESA OS variants. Other machine platforms that are supported include DEC VMS, Data General AOS/VS, and PRIME Primos. Also, the ever increasing number of UNIX based machine platforms, currently over 15 different platforms, are being supported or will be supported in the future. The success of the Personal Computer marketplace with the PC/DOS, WINDOWS and OS/2 GUI implementations has brought the low cost of desk top computing into reach for many business, academic, and consumer markets.

Finally, how does SAS Institute decide what to include in a release of the SAS System? The most important decision makers are our customers. The use of the SASWare Ballot is the vehicle by which our customers directly impact the development of changes and enhancements to the SAS System. Each development group, from top to bottom, utilizes these results to make decisions that are incorporated into the short term and long term development goals that makeup each release of the SAS System. The marketing department is also an important asset for goal determination by providing valuable information about future horizontal and vertical markets that the SAS Institute could provide specific product support. The last part of the goal determination process is the setting of strategic goals for the short and long term evolution of the SAS System as a whole. SAS Institute has unique relationships with many organizations and hardware manufacturers that have allowed us to keep abreast of developments in hardware and user interfaces.

Now that we have an understanding of the organization of the development environment, the complexity of the hardware distribution, and the determination process used in goal setting, let us now analyze the main topic for discussion, "How much time will it take?"

As any development manager knows, estimating the completion date of a project is like playing Russian roulette, either you shoot yourself in the head or you
don’t. To overcome this fear, most development estimation is based on a mixture of approaches: throwing a dart at a calendar, the reading of tea leaves or the use of a crystal ball. SAS Institute does not perform global project management, but relies upon information flow and cooperative relationships among the development groups to perform the necessary tasks to enable us to provide the SAS System to our customers.

When determining time requirements for a release we have to take into account the impact of major functional and architectural changes with respect to all the development staff. In addition to that, time estimates are required for projects that are functionally independent from one another, as in the case for most of the application product development groups. While performing this estimation process, one must keep in mind the presence of good o’l MURPHY, this invisible ubiquitous presence has always been able to throw a lug wrench into the best laid plans. But what is truly apparent at SAS Institute is one fundamental rule, 80-20, 80% of the work will be completed in 20% of the time and the last 20% takes the remaining 80% of the time.

Let us now look at the time estimation process that is utilized in each tier of the organizational model. The time estimation chore has traditionally been left up to each application product development group that has been provided with a project list of functional and architectural changes for which everyone must make accommodations. Together with this information and their own private set of projects, each application product development group estimates the resources and time requirements needed to complete the task at hand. The methods used in this process range from formal scheduling methods to practical experience from previous releases. The independence of the groups has been a mainstay for the success of the SAS System.

The Core and Host levels have required different approaches due to the interdependency of their relationship with each other and the application product development groups. Formal methods for project scheduling have been utilized in the past with a varying degree of success. Most scheduling methodologies require a rigid set of facts to be incorporated into a model from the onset. Many others are not flexible enough to accommodate the specifications of the projects that must be undertaken. The complex nature of the SAS System and its hardware platforms has always been and will continue to be a problem. Changes in hardware architecture, new or old, must be accommodated along with the impact of the ever changing operating system support software. In addition to these areas, the application product development groups only magnify the problem with last minute development issues, e.g. "I gotta have" with the implementation details of "All ya gotta do". Many times the formal systems have just broken down from exhaustion. The best method that we have been able to live with has been to analyze the list of projects and make conservative estimate for the time needed to complete each and every item. We have generally met these deadlines when we exclude the ever present last minute projects, and remember MURPHY. It is these last minute projects that are the greatest headaches in meeting the schedule to get the system ready for the QA process.

The aggregate results of these estimations from the application product, Core, and Host development groups provide for one part of the estimation process. The other part of the time estimation process revolves around QA. The QA department can only handle a relatively limited number of machine platforms at any one time due to staff and hardware resources available. To accommodate this situation, families of machines are created using a couple of important factors, including distribution of a machine or operating system within the customer base and marketplace presence.

Remember the 80-20 rule, this is where it comes into play. Up to now we have completed 80% of the work in 20% of the time; and at this point, we begin the last 20% of the work with the remaining 80% of the time. The major projects that are generally undertaken at this time deal with the last minute items: resource recovery, performance (even when it was part of the goals for the release anyway), new functionality requirements for application product development, accommodations for new hardware architectures or operating systems environments whether it be for the current family or one down the pipeline.

Many of the last minute projects deal directly with the process in which all development groups quantify the individual projects. Some last minute projects are related to emergency situations where we could not perceive a situation until all components were brought together. Some last minute projects are important enough to put in immediately rather than wait for a future release.
What does the future hold? One major effort that is currently being discussed and implemented deals with the maintenance of the SAS System in the field. Previous approaches have dealt with providing maintenance in the form of ZAPS. This approach has been very expensive in terms of manpower and time involved in writing, testing, and distributing the ZAPS. In many cases a fix for a defect is impossible due to the size and scope of the change required to address the problem. Beginning with the release of 6.07.03, the volume of zaps provided to the field will be greatly reduced so that we can incorporate a new plan of attack. The maintenance of 6.07.03 will be updated on a regular basis incorporating only source code changes. Providing source code maintenance will require the development staff to be very conservative with respect to the type of changes and additional functionality that can be accommodated in a maintenance release. This methodology still has some issues to be resolved, but, over the next maintenance release or so, this policy will be incorporated into the distribution mechanisms for the SAS System.

As with the project management processes utilized here at SAS Institute, many departments will continue with the procedures already in place, either with formal modeling approaches and derivatives or with essentially ad-hoc methods. Other departments, specifically the CORE department, are having to refine methodologies that will allow us to better accommodate the rapid pace of change that the SAS System will be undergoing in the next few years. We are currently in the process of developing an in-house project scheduling system that is tailored to meet specific needs of this environment. An important strategy that is being incorporated now is to move the testing effort earlier in the product cycle so that more and more of it occurs before code has been finalized or even begun; this investment of time results in time savings at the far end of the development cycle.

Joel Hoffman: THE IMPACT OF CHANGING CONTEXT ON ESTIMATING DEVELOPMENT TIMES

Two adages often used when estimating the time needed to develop software are:

"If you think it is easy it is hard; if you think it is hard it is impossible," and

"To estimate the time needed to complete a software project, take your first estimate, double it and increase by a level of magnitude."

This paper shares some of the factors that impact the development of software and makes the development difficult/impossible and increases the time of development by orders of magnitude. Specifically, it will be suggested that a reason for many of the difficulties lies in the fact that SAS is such a well known, easy to use tool programming tool, and, as such, the role of the developer has changed from writing code to making code context free. Another reason lies in the assumption that all software development inevitably aims at a moving target by changing the context of the users by providing new computing functionality.

DEVELOPING CONTEXT FREE CODE

SAS is a very commonly used computing language, (as is obvious by the gathering here in Hawaii). For the development of systems, this is both the good news and the bad news. It is the good news in that developers and users share a basic common language for communicating ideas, and because this common language is a computer language, the code serves as an unequivocal, unambiguous specification of the task at hand.

For example, a typical ambiguous user specification of:

\[ \text{Divide the sample into groups of old and young with the dividing line at 65 years} \]

when received from a user who knows SAS is changed to the unambiguous:

\[ \begin{align*}
\text{IF } \text{AGE LT 65} & \text{ THEN } \text{GROUP } = \text{ 'YOUNG'}; \\
\text{IF } \text{AGE GE 65} & \text{ THEN } \text{GROUP } = \text{ 'OLD'};
\end{align*} \]

The use of this code would seemingly expedite the development of software but may, in fact, hinder it by making less obvious the assumptions underlying the code. The text in the first instance describes in words the dividing of a sample into two groups based on age. There is no SAS function "dividing line." Thus the programmer must test the meaning of this criteria with the users and move it from the context of a study design or analysis plan written in English to the context of the program written in SAS. In so doing, the additional requirements of the programming
language will bring to light the need for greater specificity:

Are people aged 65 classified as young or old? Should reported age or derived age ((current birthday)/365.25) be used?

These questions encourage both the user and developer to test the assumptions of their statements as the user request is formalized into the project code.

When the developer receives the seemingly unambiguous SAS code from the user, the use of existing code subtly changes the nature of the task. Instead of identifying all assumptions about the algorithm, the developer must:

- first identify all the assumptions held by the users,
- programmatically test the validity of the assumptions,
- provide work-arounds when assumptions are not met, and
- provide reports to users on the status of the assumption testing.

Stated another way, as compared to the first instance, the task in the second instance is to free the code from the context in which it has been written, rather than creating a new context for the code. This can be especially difficult because the code available from users is often written in a very restricted context (one or two users) while the code required for the software project may need to be used enterprise wide. Thus, for the two lines of code above, the developer must still identify the assumptions:

1. recorded age vs calculated age, test assumptions:
2. are the data available
3. are the data in the correct units
4. are the data stored in the correct format
5. is the variable name used the same across data sets
6. are the data stored in the same data set across projects

work-arounds for each of the assumptions:

7. if age is missing then set group to x
8. if age units = year then ...; else if age units = months then ...
9. if UK data then date format = day/month/year
10. are data in the variable AGE, or are data in the variable AGE_REC...
11. is age in DEMOG data set or is age in the PAT_INFO data set.

The magnitude of these tasks becomes apparent when the code received from users goes from two lines for two variables to 200 lines for 200 variables.

As a side note, I think there is a distinction between the issues being addressed here of context free code, and the more commonly thought of generic code. In my view, generic code is best thought of as code that will work under any circumstance as along as the circumstance is known. Context free code fulfills the same requirements, but has the additional requirement of "understanding" the circumstance or context attached, and then making the required changes so the code will work.

From the above, it should be apparent that the availability of user SAS code may increase the develop time rather than reduce it as the context in which the code is used moves from the single user to global systems use.

DEVELOPING THE CONTEXT FREE USER

There is another change in context that is also likely to affect the time needed to develop systems in SAS. This is the context in which the user finds him/herself as the system is rolled out and the user is now presented with functionality based on his/her old code but in a very different context. This new context will change the user's requirements and the functionality which is needed. McCracken and Jackson suggest:

an analogy with the Heisenbert Uncertainty Principle: any system development activity inevitably changes the environment out of which the need for the system arose. System development methodology must take into account that the user, and his needs and environment, change during the process.

Just as above where we have shown that much work is required to change user code when moving it from a single user tool to a globally useful tool, so to moving the user from the environment where his tool was useful to the new environment where global information is now available is likely to raise new requirements for the user in order to have a deliverable system. The iterations required to take the original user requirements and specification to the
new required functionality is likely to increase the time needed before the system can be delivered.

In the above, I have tried to suggest the importance of context when estimating the time needed to develop SAS systems. In a way, I have suggested that SAS, because of its wide use, presents particular problems to the manager attempting to estimate the time needed for delivery. I think it would be more fair to perhaps suggest that this is not a problem particular to SAS but rather is an emerging situation as computing tools become both widely and widely used and the role of systems development increasingly moves from development of new systems to integration of user functionality into existing information structures. The more the focus of systems development turns to integration, the more developers will need to focus on making code context free and expecting the need of users to expand the context in which their tools are used.

Conclusion

Estimating the cost and schedule for a software development project is one of the most difficult and error-prone tasks in software engineering. We have explored only a few of the reasons for inaccurate estimates in software engineering and presented a few techniques for developing better schedule estimates.

The entire topic can be summarized as follows:

  If we want peak system performance (all desired features, complete quality assurance, and steady progress), and we want the system delivered on time (that is, we meet our promised schedule), then we probably cannot determine the Cost. If we fall behind schedule, or uncover additional "features" we just add more resources (at an increase in cost). As Mr. Hoffman pointed out, there are too many context sensitive issues and hence too many compromises that may degreade "performance".

  If we must adhere to a specific schedule and deliver the system on time and within budget, then the performance of the system will probably be compromised, or at best we will deliver only a partial system, with new "features" added after the first delivery of the system.

If we cannot compromise on the performance and we cannot modify our budget estimate, then the time frame for delivering the system cannot be determined. We limit our resources (maintain our commitment on cost), and probably sacrifice the quality of our results.

If we can identify all of the components of a software project, and if these do not change during the life time of the project development, then we can make a very good estimate of how long it will take certain types of people with specific skills to complete the work. However, to meet a schedule within a budget, nothing can change, and nothing can go wrong. It has been said that a perfect estimate will be high 50% of the time; a perfect estimate will be low 50% of the time. An estimate is an estimate, is an estimate. It is not a promise.

References:


The authors may be reached at:

  Joel Hoffman (203) 441-4981
  Judith Mopsik (301) 258-5542
  Jeff Polzin (919) 677-8000