

## Making Team Plans

Watts S. Humphrey



At the team kick-off meeting, management told the engineers that the company critically needed their new product in nine months. This group was introducing the Team Software Process<sup>SM</sup> (TSP<sup>SM</sup>), and I had convinced management that the team should make a development plan before they decided on the schedule. Management had agreed, and we scheduled a meeting for two days later to review the engineers' plan. Now, the 12-engineer team was assembled and ready to make their plan. They had a lot of questions.

- How do they make a plan when they don't know the requirements?
- How detailed should they make the plan, and how much of the project should they cover?
- Suppose the plan doesn't finish in nine months; what do they do then?

And finally,

- Since they knew so little at about the product, could any plan they made now be useful?

These questions are the subject of this column.

### **How do they make a plan when they don't know the requirements?**

The team was very concerned about the vague state of the requirements. While a couple of the engineers had a general idea of what the product was to do, they did not see how they could make a realistic plan without much more detail. I was coaching this team and pointed out that they could make an accurate plan right after final product delivery. Their plan would be most accurate then, but it would be least useful. On the other hand, they most needed a plan at the beginning of the project, but it would necessarily be least accurate. So, while they did not yet know very much about the product, they agreed to make the best plan they could produce.

### **Plan accuracy**

Clearly, the more you know about a product's requirements and design, the more likely you are to make a good plan. Thus, plans will always be least accurate at the beginning of a project, and they should get progressively more accurate as the work

progresses. This suggests three things. First, you must plan, even if you don't know very much about the job. Second, you should recognize that the initial plans will be the least accurate. And third, you need to continually remake the plans as you learn more about the work.

### **How detailed should they make the plan, and how much of the project should they cover?**

Planning is much like designing a product. It is a good idea to start with an overall architecture or process and then to lay out the entire structure. What development tasks are required, in what order, and how long will they take? Until you have an overall framework, a detailed plan could address the wrong tasks or focus too much effort in the wrong places.

For example, I was assigned to a project some years ago. There had been many small schedule changes, but everybody thought the project was on schedule. Nobody had ever produced an overall plan. However, when we did, we found that testing time had been dangerously reduced. The project was in serious trouble.

Without an overall plan, it is hard to see the cumulative impact of many small schedule slips. They all add up, however, and without an overall perspective, the latter phases will invariably be squeezed. So, while detailed plans are essential, they must be made in the context of an overall plan that runs from the start date all the way to the final product delivery. Therefore, the first step must be to make an overall plan.

### **Start with the process, then list the products and make an estimate**

Once the engineers agreed to make an overall plan, they had to decide on what development process to use. By starting with the organization's overall process framework, they defined their specific project process in less than an hour.

Next, they defined the products to be produced by each process phase. They estimated the sizes of the requirements and design documents and postulated an overall product structure. They judged what components would be required and how big each component was likely to be. Each engineer contributed to these discussions, and they compared this job with others they had worked on. It was surprising how much relevant information the 12 of them had.

Next, the team had to figure out the effort required to develop each of these products. Again, every engineer contributed his or her views. In some cases, they had real data for similar jobs. In other cases, they made overall judgments based on general recollections. In the end, they came up with estimates for every product. While some of these estimates were guesses, they were informed guesses made by experienced engineers who had previously done similar work.

## **Make the schedule**

The last overall planning step was to produce the schedule. Here, the engineers estimated how many hours they each had available for the project each week. Since many had prior obligations that would continue, they allowed time for this other work as well. When they were done, they had an estimate of the total hours the entire team would have available for each project week. Then they spread the work over these hours to produce the schedule.

By this time, the engineers had a pretty good idea of how big the job was. Thus, they were not surprised that the project would take much longer than the 9 months that management wanted. The full schedule actually turned out to be 18 months. At this point, the team had defined the complete process that they would use, produced a product list, made product-size estimates, and generated an overall plan—all in one afternoon. While they were still concerned about the plan's accuracy, they knew this was a big job, and there was no chance they could do the work in 9 months. They also had a lot of data to back up their 18-month schedule.

## **Next came the detailed plan**

The next step was to look at the work that lay immediately ahead. On the morning of the second day, the team made a detailed plan for the requirements phase. First, they examined the requirements process and broke it into the steps needed to produce, review, and correct each requirements product. To make sure their detailed plans fit into the overall plan, they started with the overall estimates and then estimated the engineering hours for each step. They then named an engineer for each task, and each engineer then used the same overall planning data as the starting point for a personal plan for the immediate next phase.

When the team put these plans together, the result was a shock. The combined detailed plans took much longer than the top-down plan for the same work. How could this be? The same engineers had made the plan and they had used the same product list, size estimates, and development rates.

The problem was *unbalanced workload*. The lead engineers were involved in every step of the work, and the less experienced engineers often had little to do. While the lead engineers could likely produce the best products and everyone felt that they should participate in every product review, this made them a serious bottleneck.

After some discussion, the team agreed to unload much of the lead engineers' work. By balancing the workload, the less experienced engineers got much more to do and the lead engineers concentrated on the most critical parts of the job. The final balanced plan produced the same schedule as the overall plan, and the team now felt they had a sound basis for doing the work. At this point, it was noon of the second

day, and the team had all afternoon to assess project risks and to prepare a presentation for the management meeting.

### **What happened**

When the team presented their plan the next morning, management was impressed with the plan, but unhappy with the schedule. They really did need the product in 9 months, but, after considerable discussion, they were convinced that the 18-month schedule was the best that the team could do.

The team followed this plan in doing the job. The requirements phase took several weeks longer than planned and the design phase also took a little longer. But, the team stuck to their guns and did a quality job. As a consequence, there were fewer late requirements and design changes, implementation took less time than planned, and testing took practically no time at all.

In the end, the team finished the job 6 weeks ahead of the original 18-month schedule. Because of the well-thought-out design and the high product quality, marketing was able to contain the customer problem, and the product was a success.

### **Teamwork**

Teams have a great deal of knowledge and experience, and when they are all involved in producing their own plans, they will invariably do a first-class job. After all, they will do the work, they have the most at stake, and they will derive the most benefit from having a realistic plan. With a detailed plan, teams know precisely how to do the work, and they feel obligated to finish on the dates to which they committed.

### **Closing comments**

First, early plans are invariably less accurate than those made later. The reason is that engineers often overlook tasks, they don't allow enough time to clear up requirements problems, and they assume that they will work full time on the job. Also, in many organizations, management fails to protect their teams from the normal turmoil and disruption of a running business. Thus, when you consider all the pressures in working software organizations, the early team plans are almost always aggressive. Thus, even if an earlier date is critically important, it is invariably a mistake to cut these initial plans. If the problem is severe, the team should make a new plan with different resource assumptions or work content. The best approach, however, is to wait until the end of the requirements phase to replan. Then everyone will better understand the work, and they can make a more accurate plan.

Second, there are lots of estimating tools and methods. While I am partial to the PROBE method described in one of my books,<sup>1</sup> estimating is a largely intuitive process. So, use whatever methods help your intuition. However, do not rely on some magic tool to produce the plan. While the detailed printout may look impressive, plans are only as good as the thought that went into them. Remember that the principal benefits of planning are the engineers' shared knowledge of how to do the work and the team's commitment to the plan. Use whatever tools and methods you have available to help make the plan and to check your results, but use these tools only to support your planning, not to replace it.

Third, to help you work efficiently and to coordinate your work with your teammates, you need a detailed plan for the work immediately ahead. While you can rarely produce a detailed plan for an entire development job, you should start with an overall plan and then produce a detailed plan for the phase you are about to start.

Fourth, when management is unhappy with your team's plan, don't change it without making a new plan. When you do, however, make sure you get different resource and work-content assumptions. Without changes in their planning assumptions, teams invariably think of previously overlooked tasks and end up with a longer schedule.

Finally, remember: if you cannot plan accurately, plan often. Plans are only as good as the knowledge on which they are based. As you gain new knowledge, produce new plans. As long as the previous plan is useful, however, don't bother making a new plan. But, the moment the plan ceases to provide helpful guidance, make a new plan.

### **The commercial**

While the methods I have described are not complex, they are not obvious. That is the purpose of the Team Software Process that we have developed at the SEI. It provides the guidance that teams need to follow these methods on the job. The catch, however, is that to use the TSP, engineers need to be trained in the Personal Software Process<sup>SM</sup> (PSP<sup>SM</sup>), and their management needs overall training and guidance on how to lead and guide TSP teams.

### **Acknowledgements**

First, I would like to thank Walden Mathews for asking some perceptive questions about planning. The answers to his questions provided the basis for this column. Also, in writing papers and columns, I make a practice of asking associates to review early drafts. For this column, I particularly appreciate the helpful suggestions of John Goodenough, Mark Paulk, Bill Peterson, Marsha Pomeroy-Huff, and Dave Zubrow.

---

<sup>1</sup> *A Discipline for Software Engineering*, Addison Wesley, 1995.

## **In closing, an invitation to readers**

In these columns, I discuss software issues and the impact of quality and process on engineers and their organizations. I am, however, most interested in addressing issues you feel are important. So please drop me a note with your comments, questions, or suggestions. I will read your notes and consider them when I plan future columns.

Thanks for your attention and please stay tuned in.

Watts S. Humphrey  
watts@sei.cmu.edu

## **About the author**

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and six books. His most recent books are *Managing the Software Process* (1989), *A Discipline for Software Engineering* (1995), *Managing Technical People* (1996), and *Introduction to the Personal Software Process<sup>SM</sup>* (1997). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago.

The views expressed in this article are the author's only and do not represent directly or imply any official position or view of the Software Engineering Institute or Carnegie Mellon University. This article is intended to stimulate further discussion about this topic.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

<sup>SM</sup> IDEAL, Interim Profile, Personal Software Process, PSP, SCE, Team Software Process, and TSP are service marks of Carnegie Mellon University.

® Capability Maturity Model, Capability Maturity Modeling, CERT, and CMM are registered in the U.S. Patent and Trademark Office.