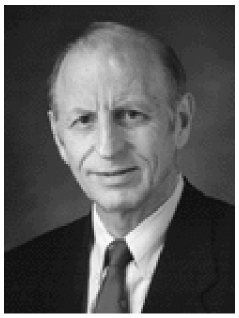
In writing this column, I plan to discuss various topics of interest to software professionals and managers. In general, I will write about issues related to engineering productivity, quality, and overall effectiveness. Occasionally, I will digress to write about a current hot item, but generally I will be pushing the process improvement agenda. Because my principal interest these days is getting organizations started using the Personal Software Process^{SM} *(PSP*^{SM}*) and Team Software Process*^{SM} *(TSP*^{SM}*), readers should know that a not-so-hidden agenda will be to convince them to explore and ultimately adopt these technologies.*

Why Does Software Work Take So Long?

Watts S. Humphrey



Have you ever started what you thought was a two- or three-day job and have it stretch into a week or two? Before deciding you are just bad at estimating, look at how you spent your time. You will find you spend much less time on projects than you imagine. For example, on one project, several engineers used time logs to track their time in minutes. They averaged only 16 to 18 hours a week on project tasks. They were surprised because they all worked a standard 40-hour week.

This information soon turned out to be helpful. They were on a critical project and were falling behind. When they looked at the data, they found the design work took 50% longer than estimated. They knew they had a choice: either do the tasks faster, or put in more time. While there was pressure to race through the design, skip inspections, and rush into coding, the team resisted. They knew this would probably result in many errors and a lot of test time.

To meet their schedule, they needed to average 30 task hours a week. They all tried valiantly to do this, but after Christmas, they realized that just trying harder would not work. They went on overtime and are now starting early in the morning, working late some evenings, or coming in on weekends. While they now average 30 task hours a week, they have to work over 50 hours a week to do it. They are also back on schedule.

Because this team had detailed time information, they could recognize and address their problem in time to save the project. The data identified the problem and pointed them toward the solution. Without good data on where your time goes, your estimates will always be inaccurate and you won't know how to improve.

Working harder

When people say they are working harder, they actually mean they are working longer hours.

Barring major technology changes, the time it takes to do most tasks is relatively stable. The real variable is the time you spend on the tasks. But to manage the time you spend, you have to track it, and practically nobody does. Consider the following:

1. Our lives are filled with interruptions.
2. Software people do many kinds of tasks, and only some contribute directly to our projects.
3. Our processes are often informal and our working practices ad hoc.
4. Even if we wanted to, it is hard to do demanding intellectual work for long uninterrupted periods.

Interruptions

One engineer told me she had recently started to track her time and found she was spending much more time on interruptions than on her real work. For example, on one task of 108 minutes, her interruption time was over 300 minutes. This lost time, however, was not in big hour-long blocks but from an incessant stream of little 5- and 10-minute interruptions.

Interruptions come from many sources:

- telephone calls
- other engineers asking for help
- a coffee or rest break
- supply problems (i.e., printer or copier out of paper)
- equipment problems (the network dies)
- a power failure or a snow storm (everybody leaves)

Every interruption breaks your train of thought and takes time away from your work. With unplanned interruptions, you lose your place in the work and, when the interruption is over, you have to reconstruct where you were. This also causes errors.

For example, when I am in the middle of a design, I am often working on several things at the same time. While thinking through some logical structure, I realize that a name is misleading, a type must be changed, or an interface is incomplete. If I am interrupted in the middle of this, I often have trouble remembering all these details. While I have been unable to control the interruptions, I have found that maintaining an issue log helps me remember what I was working on when interrupted.

Non-project work

Most engineers also spend a lot of time on non-engineering tasks. Examples are

- handling mail
- setting up or updating their computing
- systems
- going to meetings
- looking for some specification, process,
- standard, or manual
- assisting other engineers
- attending classes

Few software development groups have adequate support. No one sets up or maintains his or her development system, few have groups to handle product packaging and release, and there is no clerical or secretarial support for mail, phone, or expense accounts. What is more, even when they have such support, many engineers don't know how to use it. This means that most of us spend more time on clerical and support work than on our principal development tasks. And every hour spent on these tasks is an hour we can't spend on development.

Lean and mean organizations

Often our organizations pride themselves on having very small support staffs. An almost universally accepted management axiom is that overhead is bad and should be eliminated. In the resulting lean and mean organizations, the engineers do their own clerical work. This is not an effective way to use scarce and expensive software talent.

By cutting overhead, management also eliminates the support staffs that funds in the overhead budget support. While some of these groups are not the least bit interested in supporting the engineers, many are. Eliminating them can have enormous costs. Among these costs is the time every engineer must spend sorting through email, answering the phone, getting supplies, doing expense accounts, and filing mail and documents. In addition to the lost engineering time, this also means that most mail is not answered promptly if at all, phones go unanswered, supplies are wasted or overstocked, and little if anything is properly filed or can be quickly found when needed.

Perhaps most expensive and annoying, every software engineer in such “lean and mean organizations” must set up and maintain his or her personal computing environment. Just because we have the skills to do this doesn't mean we should. Most of us could repair our cars or paint our houses if we chose to, but it would take us longer than using someone who does this for a living. And we have other things to do. Why should we have to handle our own computing support? The principal reasons that engineers spend less than half their time doing the tasks they were trained and hired to do is that, without adequate support, they have to support themselves. What is more, few engineers enjoy or are very good at being part-time clerks and technicians.

Ad-hoc working and planning

When no one has taken the time to define and document the organization's practices and methods, they must be maintained informally. When you come to a task that you haven't done before or at least not recently, you look around to see how it should be done. It takes time and a lot of interruptions to find someone with the right experience and get their help. While this is vastly preferable to bulling ahead without exploring prior experience, it does cut into the working week.

A related but slightly different problem concerns planning. When projects don't make detailed plans, and when engineers don't know precisely where they fit into these plans, they must do what I call continuous planning. In continuous planning, the key tool is not the PERT chart or Microsoft Project, it is the coffee machine. When you finish a task, you go to your continuous planning tool to have a cup of coffee. While there you decide what to do next. In the process, you talk to any other engineers who are also doing continuous planning and see what they think. You will usually get some good ideas, but if you don't, you can always interrupt someone.

The common view is that planning takes too much time. By not planning, engineers can immediately start on their programming work. While this immediate progress will feel good, you won't know where you are. Like driving in a strange country without a map, you have to stop at every turn and decide where to go next. All these short stops will take far more total time than a properly thought-out plan would have taken in the first place.

You also need an occasional break

Finally, creative development is hard work. When designing a product or a system, we need uninterrupted time. But we cannot design complex products for more than a few hours at a time. The same is true of testing, reviewing, inspecting, coding, compiling, and many other tasks.

One laboratory decided to set up a dedicated group of experts to inspect a large and important product in final test. Every module that had test problems was sent to this group. For a while, they cleaned up a lot of defect-prone modules. Then, one of them later told me, they could no longer see the code they were inspecting. Everything began to blur. They even saw source code in their sleep.

Designing, coding, reviewing, inspecting, and testing are intensely difficult tasks. To have any hope of producing quality products, we must occasionally take breaks. But, to be reasonably efficient, and to do high-quality work, we need to control our own breaks, not take them when the phone rings or when somebody barges into our office or cubicle. Studies show that when engineers spend all their time on their principal job, their performance deteriorates. Some reasonable percentage of time on other tasks such as planning, process improvement, quality analysis, or writing papers can improve engineering performance. You will get more and better work done in the remaining 75% of your time than you would have accomplished in 100% of dedicated time.1

So, keep track of your time

To manage your personal work, you need to know where your time goes. This means you need to track your time. This is not hard, but it does require discipline. I suggest you get in the habit of using the time recording log, shown in Tables 1 and 2.2 When doing so, enter the tasks and the times when you start and stop these tasks, and also keep track of interruption times. If you do this, you will soon see where your time goes. Then you can figure out what to do about it. Manage interruptions Next, interruptions are a fact of life, but there are many ways to deal with them. Use “DO NOT DISTURB” signs and establish an ethic where everybody (even the managers) respects them. Forward phone calls or even unplug or turn off the phone. Also consider getting permission to work at home for a day or two a week.

Another way to manage interruptions is to get in the habit of using an issue-tracking log. Then, when you think of something you need to do, make a note of it in the log so you will remember to do it later and you won't forget it when the phone rings. While you will still have to handle these issues, you are less likely to forget them and you can do them at a planned time.

Also, use this same principle with interruptions. When someone calls in the middle of a design problem, tell them you'll get back and then make a note on a sticky so you don't forget.

Learn to use administrative support

Learn how to use support. While few engineers have a support staff to help them, many who do don't know how to use them. If you have a support person, think about every clerical-type task before you do it. Can this person do it for you? Even though it may take longer at first, use them whenever you can. At first the result may need to be redone. But be patient and help the support people understand your problems with their work. It will pay off in the long run.

Plan every job

Perhaps most important, learn to plan. Plan your own work and urge your teammates and the project leader to start planning. Proper planning takes time, but it can save much more time than it costs. You will end up planning anyway, but it is much better to do it in an orderly way, and not at the coffee machine.

Vary your work

You can do demanding work only for so long. I lose my ability to do intense creative work after an hour and a half to two hours. I need to stop for a break or even to switch to some other kind of work. Further, during these intense sessions, frequent short interruptions offer no relief. It then takes an extra effort to reconstruct my thought process.

What I suggest is to intersperse various kinds of work throughout your day. Do creative work when you are most fresh and productive and then switch to your email or an administrative task. Then perhaps do a design or code review possibly followed by a process-improvement task or data analysis. By varying the task types, your creative work will be of higher quality and you will actually get more done.

Define and use a personal process

When you regularly make plans, a defined process will save a lot of time. The process provides a framework for gathering historical data and a template for making plans. And, by using historical data, your estimates will be more accurate.

Get and use historical data

Finally, if you don't have administrative or technical support, use your time log to see what this lack costs you. Then tell your managers and show them your data. It might help them see the cost advantages of adequately supporting their engineers. Remember that the amount of work you produce is principally determined by two things:

1. the time the tasks will take
2. how much time you have available for these tasks

To manage your work, you must know where your time goes. Only then can you judge how much work you can do and when you will finish it.

Acknowledgements

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 I received from Dan Burton, Alan Koch, and Bill Peterson.

In closing, an invitation to readers In these columns, I plan to discuss software issues and the impact of quality and process on engineers and their organizations. I am, however, most interested in addressing issues that you feel are important. So please drop me a note with your comments and 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

Notes

1. For a brief discussion of this issue, see my book *Managing Technical People, Innovation, Teamwork, and the Software Process*, Addison Wesley, 1997, page 186. A more complete discussion is in Donald C. Pelz and Frank M. Andrews, *Scientists in Organizations*: *Productive Climates for Research and Development*, Wiley, 1966, pp. 56, 65.

2. Recording Log, see my book *A Discipline for Software Engineering*, Addison Wesley, 1995. This log is also discussed in *Introduction to the Personal Software Process*, also by me and published by Addison Wesley in 1997.

Table 1: Time Recording Log

Engineer ________________ Date ________________

Program ________________ Module ________________

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments

Table 2: Time Recording Log Instructions

Purpose	• Use this form to record the time spent on each project task. • Either keep one log and note the task and product element for each entry or keep separate logs for each major task.
General	• Record all the time you spend on the project. • Record the time in minutes. • Be as accurate as possible. If you need additional space, use another copy of the form.
Header	Enter the following: • your name • today's date • the project name • the name of the program or other • product element If you are working on a non-programming task, enter the task description in the comments field.
Date	• Enter the date when you made the entry.
Example	• 4/13/98
Start	• Enter the time when you start working on a task.
Example	• 8:20
Stop	• Enter the time when you stop working on that task.
Example	• 10:56
Interruption Time	• Record any interruption time that was not spent on the task and the reason for the interruption. • If you have several interruptions, enter their total time.
Example	• 37 — took a break
Delta Time	• Enter the clock time you actually spent working on the task, less the interruption time.

Table 2: Time Recording Log Instructions (Continued)

Example	• From 8:20 to 10:56, less 37 minutes or 199 minutes
Phase	• Enter the name or other designation of the phase or step you worked on.
Example	• planning, code, test, etc.
Comments	• Enter any other pertinent comments that might later remind you of any unusual circumstances regarding this activity.
Example	• Had a requirements question and had to get help.
Important	• Record all worked time. • If you forget to record the starting, stopping, or interruption time for a task, promptly enter your best estimate.

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*$^{\text{SM(}}$1997). He holds five U.S. patents. He is a member of the ACM, an IEEE fellow, and a past member of the Malcolm Baldridge National Quality Award Board of Examiners. He holds a bachelor's degree in physics from the University of Chicago, a master's degree in physics from the Illinois Institute of Technology, and a master's degree in business administration from the University of Chicago.