

## Using Task Analysis in Documentation Field Research

Kent Sullivan  
Usability Group  
Microsoft Corporation

Task analysis, the “systematic analysis of human task requirements and/or task behavior” (Stammers *et al.*, 1990), is a primary research method used in the human factors and ergonomics fields to identify and understand the components of a particular job, set of tasks, or task in a particular context. Researchers and practitioners in other fields, such as technical writing and usability testing, have recognized the importance of task analysis in designing concise, usable documentation, as well as in helping to create intuitive products. In fact, a technical writer must do some form of task analysis in order to truly create a task-oriented manual. However, the task analysis performed is often implicit in the writing process instead of a formal procedure.

A number of articles about task analysis methods have been aimed at technical writers in recent years, but my research indicates that formal methods are being used very selectively in many companies in the computer industry. (See Berghel & Roach [1990], Bradford [1988], and Leonard & Waller [1989], among others.) While there could be many reasons for this lack of use, my work with several different writing teams at Microsoft points to three possibilities: (1) an assumption that task analysis is primarily valuable when a completely new product is being created, (2) a feeling that task analysis would take too much time for the information it yields, and (3) confusion about what task analysis techniques would be best to use for a given situation and set of questions.

In this paper I address these three concerns by describing some first steps I took in adapting task analysis for a usability field research project at Microsoft. The context of my discussion is one phase of a usability field study conducted on a programming product. Specifically, I discuss:

- 1) The questions the writing team needed to have answered
- 2) The type of task analysis I chose to use
- 3) The type of information the task analysis generated
- 4) How the group of writers (and program designers) used the information
- 5) Ideas for implementing the method

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

### Field Study Questions

The usability field study described in this paper had three major phases and lasted approximately six months. We undertook the study to better understand how programmers were using the latest version of a Microsoft language product. Studies in our usability laboratory had shown us that a “typical programming task” was very hard to define—users of the product were engaged in a wide variety of projects—and that a lab study did not allow sufficient time with any one programmer to truly understand his/her work patterns.

We designed a longitudinal field study to collect information on the use of specific parts of the product’s online and printed documentation. (See Gould & Doheny-Farina [1989] and Sullivan [1989] for discussions of the type of field study techniques we employed.) In the final phase of the study, I chose task analysis to gather information on three broad questions:

- 1) How do users conceptualize their programming projects?
- 2) How do users build existing projects (i.e., compile and link code) and create new projects?
- 3) How do users manage projects?

These questions reflected the fact that the writers’ concerns had broadened over the course of the study. The writers began with questions exclusively about the documentation—the elements for which they had primary responsibility—but grew to include the product and the context in which it was used. Some of those initial questions were:

- 1) How usable is the mix of online and printed documentation?
- 2) Can users learn new procedures using the online help system?
- 3) How do people use README.DOC? (an addendum delivered as an ASCII text file)

Information collected in the early phases of the study indicated that our understanding of the way people were using Microsoft language products to do their programming work was incomplete. As a result, the writers decided they needed to back up and look at what they called “the big picture.”

Another way to categorize the evolution of the writers’ concerns is that the early phases of the field study focused mainly on the

tasks the writers and documentation system anticipated (although task analysis methods were not used). The final phase of the study, however, focused specifically on the tasks the programmers needed to accomplish.

### Type of Task Analysis Chosen

In task analysis, the basic technique—the decomposition of a whole into its parts—is constant across the many implementations, but the focus and level at which the decomposition starts and ends varies widely, depending on the questions to be answered. In choosing the type of task analysis for this field study, I first had to carefully analyze the questions the writers had expressed. Their initial set of questions was much larger, and some of the questions and underlying issues overlapped. As is typical in usability testing, I asked for clarification on the issues, which in turn narrowed the number of questions and sharpened their focuses. (See Dieli [1989] for a discussion of the problem definition process the Microsoft usability group uses.) This careful definition of the exact questions to be addressed had a dramatic impact on the type of task analysis chosen and helped ensure that the data collected matched the writers' expectations.

The type of task analysis I chose was *hierarchical* and borrowed ideas from several sources, including Wigley (1985). In a hierarchical task analysis, each task is analyzed by “breaking it into task elements or goals which become increasingly detailed as the hierarchy progresses” (Stammers *et al.*, 1990). The most general information is placed at the top of the hierarchy, with the more specific information following on lower levels. (See Figure 1 for a diagram of part of a task hierarchy. It is discussed in more detail in the following section.)

As discussed in Stammers *et al.*, (1990), the hierarchical method can allow an economical approach—areas of little interest can be treated superficially without risk to more important areas. This feature was especially attractive to me because of the limited amount of time I had to complete the task analysis. I was a guest of the various companies participating in the field study, so I didn't have unlimited access to the sites or subjects. And, as usual in the computer industry, the clients of the field study (the writers) wanted the information quickly so they could begin using it on the next iteration of the product.

Other task analysis techniques would have been appropriate for different questions. For example, had one of the questions been “What information do users need to have in order to successfully use the debugger?” then a task analysis focusing on *knowledge description* could have been used (Johnson *et al.*, 1984).

To collect this information, several techniques were employed. They are described in Table 1 below:

**Table 1: Data Collection Techniques**

Data Collection Technique & Topic Covered	Rationale
Interview: Description of last 3 programming projects and general programming process	Learn about subjects' job responsibilities and perception of the site's programming process
Observation 1: Use of the language product to create the essential items of a new project	Identify tasks in creating a new project in subjects' work environments
Observation 2: Use of the product to build current project and organize files for workgroup access	Identify tasks in building an existing project and tasks involved in sharing project files; also confirm each subject's job duties
Thinking-aloud protocol: Verbal report of thoughts while under observation	Understand why subjects were doing each task and collect information on mental models

### Types of Information Generated

As described in Table 2 below, four major types (levels) of information were collected: site, job, task, and element. As shown in the table, each of the levels of analysis served a different purpose.

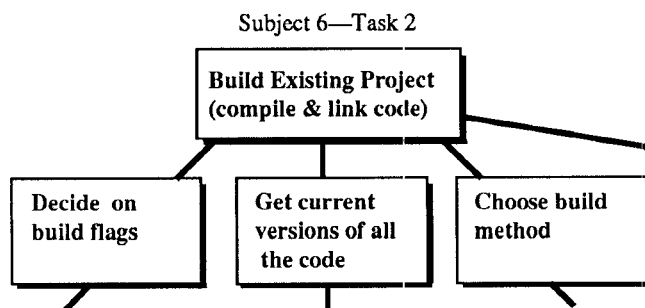
**Table 2: Four Levels of Analysis**

Level of Analysis	Definition/Use for Information
Site	Each of the companies in the study was a site. Information collected helped place the subjects from that site in the overall organization—size of programming group, software products produced, etc.
Job	Each of the programmers at a site had a specific job. Information collected helped define what it really meant to be a “professional programmer” by identifying the different responsibilities of each subject.
Task	Each programmer did a set of core tasks. Information collected helped describe what subjects expected the language product to do for them.
Element	Most tasks could be broken down into a set of “elements”, which are discrete actions or steps executed during the performance of a task. Information collected helped describe each core task and establish how subjects performed each task.

Additionally, the site and job information together helped establish a base for comparing the task and element information among sites. By understanding the similarities and differences of the subjects' jobs and their workgroups we were able to identify patterns where appropriate and keep results separate where not.

In the task hierarchy, the "site" data I collected was typically entered first, at the top, because it was the most general. The other data—job, task, and element—was then entered in the lower parts of the hierarchy. The job, task, and element information re-described the site data in more specific terms. For example, in order to answer question 2, "How do users build existing projects (i.e., compile and link code) and create new projects?", a high-level analysis of workgroups drawing from site and job information was completed first. Then, a detailed look at subjects' interaction with the specific "project setup" commands in the programming product, drawing from task and element data, was done. (See Figure 1 for a diagram of part of a task hierarchy.) The process was often recursive—low level task and element data refocused the higher level categories.

Figure 1: Sample Task Hierarchy (partial)



Once the detailed data concerning how subjects created, built, and maintained their projects was analyzed in the task hierarchy, it was possible to summarize the data in a series of increasingly-specific tables. Each table was also accompanied by a prose discussion. Each of these tables represented the information from one or more nodes on the same level of the task hierarchy.

Tables 3 and 4 show an example of the tables created in order to answer question 2, "How do users build existing projects (i.e., compile and link code) and create new projects?" Table 3 below defines the three stages of a typical programming project for the subjects. This data is on a fairly high level.

Table 4 below describes subjects' interaction with the product's commands for building a project (compiling and linking code) in the "middle" and "late" stages. This data is on a much more elemental level.

The high-level data concerning how the subjects conceptualized their projects was not presented in tables but instead in flowcharts of their development processes. A prose summary of the similarities and differences between sites was also included. The flowcharts clearly depicted the multiple activities that often were occurring simultaneously, which is difficult to do with a table. (Figure 2 below shows part of a sample flowchart.)

## How the Information was Used

Feedback from the writers and other members of the languages product group indicates that the information generated by the task analysis has had impact in four main ways:

- Described workgroup dynamics and elucidated ideas for tools to meet group project demands. Site and job information told the language product's designers that the programmers were working almost exclusively in groups. The information also described how they used different tools to manage group projects. The task analysis also highlighted that programming groups of all sizes, not just large ones, shared responsibilities and code on projects. One or more programmers in each group spent a lot of time managing the group's activities. The task analysis gave the designers information on how they might design tools to ease the project management task and also provide more powerful capabilities.
- Confirmed the existence of troublesome interface components. The task analysis data also confirmed that parts of the interface of the programming product were hard to use. Observations conducted in an earlier phase of the field study pointed to problems with a specific series of menu commands and dialog boxes. The task analysis showed that one core programming task did not map well onto the command structure of the language product. The programmers had a clear picture of what they needed to accomplish but had considerable difficulty choosing the correct commands and settings to complete the task. The task analysis data gave the writers ammunition for advocating change in the product through meetings with the program management for the product.
- Clarified and gave detail on how subjects defined a "typical programming project". Task analysis data clarified and provided useful detail on how the programmers divided their projects into three fairly distinct phases (early, middle, and late) and how they were using the language product in each phase. (See Table 3 below for a more complete summary.)

Table 3: Three Stages in a Typical Project

Stage	Common Actions
<u>Early:</u> Planning/ Experimenting	Write shell of program and a few functions before attempting first compile; spend time setting up build/compiler/link options; do initial compiles to check syntax of code; begin building (i.e. include linking)
<u>Middle:</u> Coding/ Debugging	Work iteratively—make small changes and do builds often to check correctness; use debugger to track down troublesome problems
<u>Late:</u> Testing/ Releasing	Change build/compiler/link options to what is wanted for the final release product; build release version and test; change build options back to "debug" and work iteratively to remove remaining bugs; change options back to "release" and prepare final builds

- Contributed to a raised awareness of the user's perspective.  
The report containing the task analysis results contributed to a raised awareness of users and the successes and failures they were having with the programming product. Several follow-up laboratory usability tests were conducted on the next version of the product to help ensure its interface improved.

Figure 2: Programming Process Flowchart (partial)

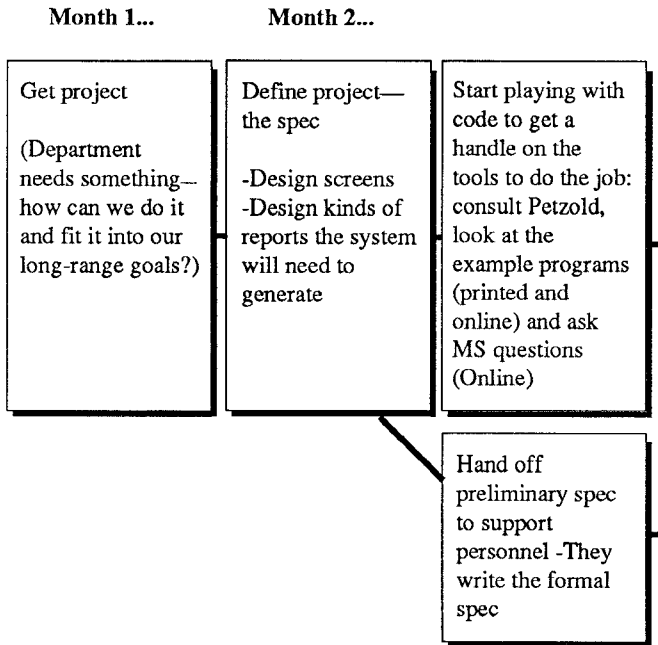


Table 4: Use of Project Building Commands

Command	Use
Build: <PROGRAM>.exe	Used almost exclusively in the middle and late stages
Compile File: <FILE>.C	Used occasionally to check the syntax of a new piece of code and when experimenting
Rebuild All	Used occasionally to (1) make sure all code being tested was up to date, (2) see if a mysterious bug was being caused by out of date code, and (3) prepare final release candidates
Build Target...	Not used—subjects did not understand the purpose

## Implementation Ideas

The task analysis described in this paper provided information on a variety of levels. The method, in general, is very flexible—adaptable to most any set of questions concerning users, tasks, and software. While task analysis data was used to answer broad questions about the programming process, task analysis can also be used to focus on very detailed, interface-specific questions.

Given the method's flexibility, one may wonder how best to use the method. Because documentation writers can use formal task analysis to improve the materials they produce—manuals, training materials, etc.—as well as in the actual product being documented, there are a number of areas which can be addressed. Here are a few general ideas for areas to explore:

- 1) Ensuring that program designers have information about users' expectations and needs. It could be an impossible task to create documentation for a program that doesn't do what users expect or need. Task analysis can reveal what users want to do so that writers aren't forced to compensate for inadequate or needlessly complicated product features.
- 2) Focusing documentation on the tasks the user brings to a product rather than on the tasks the system allows. Formal task analysis might reveal that a "task-oriented" manual is describing the wrong set of tasks. (Do users really have a task they call "Formatting a paragraph"? Or is their task something else, like: "Preparing this memo using the format my boss prefers?")
- 3) Understanding what combination of basic and advanced features of a program users depend on to complete their tasks. Task analysis can help writers decide which features to discuss prominently and which features to save for advanced topics. This information can also help program designers see which features are not being used.
- 4) Clarifying the "typical user's" set of tasks. Formal task analysis can help writers identify where their audience's task set differs from the model the writers have built based on their own intuitions, experience, and informal task analysis (i.e., with co-workers).

## Conclusions

Formal task analysis methods provided valuable information about professional programmers' everyday tasks and how they were using a Microsoft programming product to complete those tasks. The information has had considerable impact on not only the language product's documentation but the product itself. Future versions of the product have been designed and usability tested with the goal of helping users complete their tasks more efficiently and completely. The language products team, as well as the usability group, are enthusiastic about the continued use of task analysis to gather more knowledge about users and their tasks.

A long-standing rule in the computer software industry is that each new version of a software product adds more features, and by extension, more complexity. An unfortunate corollary is the documentation that describes the product also becomes larger and more complex. Task analysis can help both documentation writers and software designers minimize this complexity by identifying the exact features and information users expect and use to do their work.

### Acknowledgements

My thanks go to Marshall McClintock and Mark Simpson for their many ideas and critical reviews; to Mary Dieli, Richard Gold, Patrick Kelley, and Will Sibbald for feedback and reviews; to Lisa Dreger for valuable graphic design ideas.

### References

- Berghe, H. & Roach, D. (1990). Documentation Design Based upon Intuitive Feature Taxonomy and Use Logging. *SIGDOC '90 Conference Proceedings*.
- Bradford, A. (1988). What is a Task Analysis Matrix? *Proceedings of the 35th ITCC*.
- Dieli, M. (1989). Usability Evaluation: Involving Writers in the Problem Definition Process. *Proceedings of the 1989 IPCC*.
- Gould, E. & Doheny-Farina, S. (1989). Studying Usability in the Field: Qualitative Research Techniques for Technical Communicators. In Doheny-Farina, S. (Ed.), *Effective Documentation: What We Have Learned from Research* (Chapter 16). Cambridge, MA: MIT Press.
- Leonard, D. & Waller, A. L. (1989). Usability Planning for End User Training. *SIGDOC '89 Conference Proceedings*.
- Stammers, R., Carey, M., & Astley, J. (1990). Task Analysis. In Wilson, J. & Corlett, E. N. (Eds.), *Evaluation of Human Work* (Chapter 6). Bristol, PA: Taylor & Francis.
- Sullivan, P. (1989). Usability in the Computer Industry: What Contribution Can Longitudinal Field Studies Make? *Proceedings of the 1989 IPCC*.
- Wigley, W. (1985). INPO / Industry Job and Task Analysis Efforts. *Proceedings of the IEEE Third Conference on Human Factors and Power Plants*.