Addressing the Problems of Program Documentation

Claire M. Perkins
Kaibab Industries
4602 E. Thomas Rd
Phoenix, AZ 85018

Documentation has long been a thorn in the side of many data processing departments. Tedious to prepare, it is often left until the last stage of system development and done sloppily, incompletely or not at all. Major maintenance problems occur when the only documentation for a system exists in some programmer's head -- and he is no longer around.

As programmers, we have all felt the lack of quality documentation at one time or another when we were expected to maintain a system with which we were totally unfamiliar. We probably did everything short of creating a voodoo doll of the programmer who left us this mess of spaghetti code with not so much as a data flow diagram for a roadmap. Unfortunately, someone somewhere out there was probably thinking much the same thing about the systems we left behind.

As programming managers, we have felt the lack of quality documentation each time we've watched minor maintenance requests evolve into major time consumers. We have sweated over deadlines and smoothed users' ruffled feathers as time ticked away while our staff struggled to find the right source code, the right compile job and all the relevant parts and pieces that make up the system. And we have sworn that we would put an end to the problem by creating the missing documentation, or updating and organizing what little documentation we had. But not this time, not this project, because we were already pressed for time.

As end-users, we have often thought we might just as well have asked for a new system when we asked for maintenance work from the DP department. We just couldn't understand why a simple request should take so long and cost so much. When the programmer in charge of the project suggested that we should allow even more time for the project so that documentation could be created, we were convinced that the DP department's motives were completely self-serving.

Everyone, it seems, has felt the effects of the problem, but like the weather: "Everyone complains, but no one does anything about it."

While the weather is something beyond our control, the documentation problem is not. Each of us; programmer, programming manager and

end-user, has the power to improve the situation. Okay, so where do we start? Well, like all good analysts and programmers, we start with the basics: defining terms and defining the problem.

WHAT IS DOCUMENTATION ?

Documentation is a collection of documents. Getting out my good old Funk and Wagnalls I see that a document is "something written or printed that furnishes conclusive information or evidence." Conclusive means "putting an end to the question." I guess that means that those old binders full of paper which usually create more questions than they answer don't qualify as documentation.

There are many categories of documentation to be found in the typical DP shop. Some categories of documentation and the things they may contain are as follows:

> Policies and Procedures--used to define the guidelines under which the department should operate.
>
> Project Documentation--includes project plans, estimated time and cost for project completion, critical path diagrams, status memos and meeting minutes.
>
> System Documentation--includes high level HIPO charts, a narrative system overview, and a list of all the programs, job streams, data files and reports contained in the system.
>
> Program Documentation--includes a narrative description of the program, some kind of flowchart or other diagram illustrating logic flow, a maintenance history, and narrative descriptions of any complicated sections of code.
>
> Operations Documentation--includes instructions for streaming and/or monitoring jobs, instructions in case a job aborts, report distribution instructions, and a list of contact people for each of the systems maintained.
>
> User Documentation--includes instructions for data entry, pictures of screens used and reports generated by the system, and helpful hints for problem resolution.

Different DP shops will have different combinations of the kinds of documentation listed above. This is due, in part, to a different style of work and a different range of needs from shop to shop. It is also

due to the fact that programming is a relatively high turnover field. As your programming staff changes and evolves, so does the prevelant style of documentation.

This trend seems to continue because documentation is often considered to be the sole responsibility of the programming department. No one has attempted to standardize the process, so the content, accuracy and format of the documentation is left to the discretion of the programmer.

WHOSE DOCUMENTATION IS IT, ANYWAY ?

Documentation belongs to everyone in the company. End-users, operators, managers and programmers all benefit when the documentation of DP systems and procedures is accurate, current and complete. Since most smaller companies cannot afford to staff a full-time technical writer, the actual creation of documentation may fall to the programmer(s). But the entire responsibility for defining documentation standards and absorbing the cost of creating and maintaining quality documentation should not rest with the DP department.

Everyone who benefits from good documentation has to commit to the idea of defining and enforcing documentation standards. Each part of the company needs to realize how they would benefit in the long run by taking the time and effort to confront the problem. If your DP department consistently supplied quality documentation for all DP systems, these are some of the ways that everyone would benefit:

Policies and Procedures--would ensure that all necessary documentation was being created and maintained.

Project Documentation--would provide ongoing communication about the resources required to carry out a request, as well as a historical record on which to base future time estimates.

System Documentation--would provide an overview of each system, giving new programming staff a shorter learning curve toward efficiently maintaining those systems.

Program Documentation--would make minor maintenance requests the short and simple things they should be, and make major modifications less complicated than they might otherwise become.

Operations Documentation--would allow the operations staff to do their job more quickly and efficiently.

User Documentation--would provide end-users with quick reference information, allowing them to solve some of the problems that come up without having to incur the cost of programmer or operator time.

Generally speaking, better documentation would lead to faster, better service from the entire DP staff. Whether you work in an environment where DP costs are billed directly to each department, or absorbed as overhead, time savings will always mean money savings.

Virtually every DP shop has some level of documentation available, yet often that documentation is not doing its job of "putting an end to the questions".

WHAT ARE THE PROBLEMS ?

I've run into all of the following problems to some degree:

Paper Overload-- This is when the documentation for the system you have just inherited consists of six three-inch binders, thirty-seven assorted manilla folders and a box or two of program listings. Every note that was ever taken during the life of the system has been saved...somewhere. Good luck finding anything that makes sense!

Inaccurate Information-- This includes misinformation, misleading information and obsolete information. The scary part is, this kind of documentation often looks very complete, very organized and very "official". It may have been top quality documentation at one time, but unfortunately it was so pretty that no one wanted to mess it up by updating it as the system changed.

Missing Pieces-- Somehow, the one piece of information that is vital to your understanding of the system is not available. Either it did not seem important at the time the documentation was created, or it seemed like the kind of thing that was just generally understood, or it used to be there but it disappeared at some time and was never replaced. By the time you figure it out you are so frustrated, you probably won't add your discovery to the existing

documentation. Let the next guy struggle through it like you had to!

Unorganized Information-- Lack of organization can cause the same problems as missing documentation. The information may all be there, but if you can't find it, it still doesn't do you any good.

Unstandardized Documentation-- When generations of programmers and analysts have created documentation when and how they saw fit, you may end up with a wealth of perfectly valid documentation that is simply impossible to digest. The format, content and style of the documentation is upredictable from system to system. The lack of consistency prevents any intuitive familiarity with a system you've not worked with before. Every new system you are assigned to maintain is going to have its own set of problems. Does it have to have its own style of documentation too?

No, it doesn't. None of the problems listed above are unsolvable. Actually, they are all related. They are all surface problems, symptoms of a set of deeper, interrelated problems. They stem from the lack of defined documentation standards, which in turn stems from the unwillingness to invest the necessary time and money in first defining the standards and secondly living by them.

WHY STANDARDIZE ?

Many people fight standardization. They feel that it stifles their creativity, or they equate standardization with bureaucracy. This is especially true of programmers and analysts whose job is to creatively solve problems.

But let's face it. Documentation is supposed to be a factual representation of our systems, not an exercise in creativity!

The real reason most people resist standardization is because it is forced upon them. If you don't give people any say in the rules they have to live by, they are going to fight tooth and nail against them. They are going to fight whether they agree with the rules or not, simply because they feel imposed upon.

Involve the people who will create and use the documentation in defining the documentation standards. This approach fosters cooperation and

commitment. Also, these are the people who can best define exactly what it is that is needed.

Build a team of qualified "experts", pulled from the programming staff, the operations staff, DP management and the end-user population. Assign them the task of defining documentation standards. The project should basically follow these steps:

I Define your needs
II Explore your options
III Detail your requirements
IV Select your tools
V Streamline
VI Write it down!
VII Put it to work
IX Revise and refine

Defining your needs is a critical first step. Appendix A lists the components we felt were important to include. This may give you a starting point, but it is essential to the success of the project that you define your own needs as clearly and completely as possible.

What is important from the user's viewpoint may be quite different from what is important to the programmer. The first time through, don't overlook anything. Keep track of all the ideas that come up. It may be helpful to group the ideas into categories like system documentation, program documentation and user documentation.

Exploring your options means looking at all kinds of available tools. Tools range from paper and pencil to sophisticated case tools and PC packages. Your shop may already have some forms that have been used successfully, but maybe they would be easier to use if you set them up as Vplus forms on the HP3000. Or maybe there is a PC package out there that produces the same kind of document, but provides some additional benefits like an integrated data dictionary system. Your final documentation system could be completely on paper, completely on-line on the HP3000, completely on floppy disk, or any combination of the above.

Detail your requirements. Now that you have a list of general needs and an idea of the tools available to you, you can become more specific about the form your documentation will take. There are many difficult choices involved, but the object is to match the right tool(s) to your requirements. At this stage, you want to define the essential ingredients for your shop's documentation. You want to define not only the general areas of documentation, but the actual details required in

each of those areas. For instance, if you have decided that you need documentation on data files you may decide that that will include record layouts, blocking information, cross-referencing to all programs which update the file, or many other possibilities.

Selecting your tools now becomes much easier, because you can eliminate any that will not support your requirements. Having narrowed your range of choices, you must now find a balance between such considerations as ease of use, cost, training requirements and effectiveness. These choices apply whether you are comparing one PC package to another, an automated tool to paper and pencil methods, or anything in between. Pick the single tool or combination of tools that best meets your requirements.

Streamline. Review the list of requirements and the selected tool(s). Now is the time to take a hard look at your decisions. It would be nice to have every one of your documentation needs met, but is it practical? The reality is, if your standards are too cumbersome they will be worked around. Make some compromises, if necessary, so that you don't doom the standards to failure even before implementation.

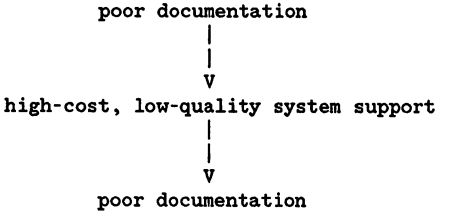
Write it down! Your team has done a lot of work to define these standards. Don't leave their implementation to chance. Prepare a document that explicitly defines the new standards. Provide samples where possible. Provide instructions for using all the tools you have selected. Publish the standards and distribute them to everyone who is expected to follow them.

Put it to work. You need to plan for the implementation of the new standards. If they represent a major change from the old style, you may need to ease into the changes. People will have questions, and you will run into situations that have not been planned for and don't quite fit into your guidelines. It's not a bad idea to hold some status and review meetings to keep things on track.

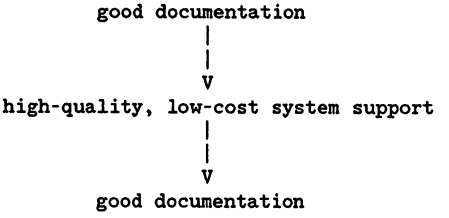
Revise and refine. Keep in mind that just because you have defined some standards, you have not set anything in concrete. You'll need to work with the new standards on a day-to-day basis for quite some time before you can really see what's working and what's not. Don't paralyze yourself waiting for the perfect tool or the perfect solution. Give it your best shot, then work with it for a while. Over a period of time, the standards will be revised and refined until they really are meeting your needs.

Once you have a really good, workable set of documentation standards and guidelines, you will begin to see real gains made in the area of

quality documentation. That first step leads to increased programmer and operator productivity, increased program and system quality, and better user support from the entire DP staff. Appendices A and B have been designed to give you some ideas and to help you get started. Keep in mind that the documentation effort needs continuing support from everyone involved: DP staff, management, and end-users. Given that support, you can break out of the cycle of:



and into the more desirable cycle of:



APPENDIX A
Documentation Requirements

Policies & Procedures

* Documentation Standards	Defining the required documents
* Coding Standards	Mainly used for COBOL, defining standards like paragraph numbering, format of IF constructs, use of intrinsics, copylibs and macros
* Production Account Standards	Defining security conventions, naming standards, account structure
* Project Management Guidelines	Guidelines on making time estimates, recording progress, making status reports
* Administrative Policies	Policies regarding things like work hours, vacation, sick leave

Project Documentation

* Service Request	Every request for service from the programming, operations or tecnical services staff is recorded on a numbered service request. The request is used as a vehicle to record estimated and actual time spent on a project, and to record the project sign-offs.
* Statement of Scope	A narrative on the scope of the project, used mainly on larger projects
* Project Plan	Includes plan of action, time and cost estimates for the project. Can be free form, or follow a preprinted outline of the typical project phases (analysis, code, test, etc...)

APPENDIX A
Documentation Requirements

* Project Log	Details action taken by date and amount of time spent
* Status Memos	Required on a regular basis for any substantial project
* Meeting Minutes	Minutes for any meeting involving the user and/or the project team

System Documentation

* System Flow Diagram	High level input/process/output
* Cross-reference of Jobs, Programs, Files, Reports	Defining the interdependence of processes and files
* System Problem Log	Includes date and time of problem, who reported it, who worked on it and how it was resolved
* Contact List	List of contacts for the system including programmer and/or analyst, main user contact, controller

Program Documentation

* Flow Chart, Warnier or HIPO	High level illustration of main logic flow
* Program Narrative	Narrative description of purpose
* Maintenance History	A running report of changes made including date, version number programmer and one-line description. Should include reference to the service request number.
* Imbedded Narrative	Imbedded comments anywhere they are needed to clarify the code

* Compile Instructions	Which production compile job should be used and/or related copylibs, macro files or special prep requirements
* Current compiled listings	Stored in hanging folders, sorted by system

Operations Documentation

* Job Descriptions	Narrative of job purpose
* Job Run Instructions	Operator instructions for streaming and/or monitoring the job
* Job Restart Instructions	Operator instructions in case of job abort or system failure
* Report Distribution List	List of who is to receive report(s)
* System Contacts	List of contacts for the system including programmer and/or analyst, main user contact, controller

User Documentation

* User Manuals	Include system overview, data entry instructions, sample screens and reports, help section
* On-line Documentation	Where possible use self-explanatory input screens and/or on-line help facilities
* Data Services Handbook	Defines who's who in the DP what services are offered, and how to get help

APPENDIX B
Pitfalls and How to Avoid Them

Programmer Resistance--First, be sure to involve the programmers in defining the standards. Secondly, select the tools carefully to ensure ease of use. Avoid defining standards which appear to be no more than red tape to the people who must create and maintain the documentation. Keep the programmers involved in fine-tuning the documentation process.

Management Resistance--Sometimes it's the programmers who feel the problems of unstandardized documentation the most. Management may not be terribly cooperative, however, if they are approached only with complaints about the current state of things. Do some of the ground work before approaching management. Define the problem(s) in detail, look into some of the available tools and solutions. Go to management with a plan for change, well outlined and thought through, not simply another complaint.

End-User Resistance--In our shop, the users pay cold, hard cash for our services. They resist the idea of paying for more programmer time to create documentation. You need to involve the users in the definition of the standards so that they understand what you are asking them to pay for. You need to foster the idea that they are investing time and money now in order to save time and money later. Perhaps you can compromise by billing for only half the time spent on documentation.

Overkill--Once you have decided to define standards where there have been none, it is easy to get carried away. Everyone involved in the process of defining standards must keep in mind that in order for the standards to work, they must be easy to follow. Having end-users on the team will help in this respect because they will require that you explain exactly why each document is needed. They will not want to pay for busy work.

Reinventing the Wheel--The definition of standards is important, and for the most part your needs will be unique. It is possible, however, to spend far too much time defining your standards and developing processes to support them. One thing to keep in mind is that the goal is not perfection, or even excellence. The real goal is to make steps in that direction.

APPENDIX B
Pitfalls and How to Avoid Them

Anything can be designed to death, including standards. Set a time limit at the outset of the project. Set a date by which you will have the Documentation Standards document complete and stick to it.

It is also possible to spend more time than you anticipated in the implementation stage if you decide to design your own forms or, as we did, your own on-line system. Be sure you take enough time up front to investigate the tools that are already available before you decide to design your own. The Data Processing field is notorious for reinventing the wheel. If that leads to a new and improved model, that's great! But it's awful disappointing to spend months in development and then discover there was a good solution already available.