The Diagnosis-Resolution Structure in Troubleshooting Procedures

David K. Farkas
University of Washington
farkas@u.washington.edu

Abstract

Troubleshooting procedures are prevalent in the computer industry and in many other industries and subject areas. In the computer industry, they appear in manuals and help systems and, especially, as "articles" in the KB (Knowledge Base) that comprises a core component of support websites. Developing successful troubleshooting procedures is both a technical and a rhetorical task. These procedures take diverse forms and vary greatly in complexity. Troubleshooting procedures, however, almost always have a diagnosis-resolution structure consisting of configurations of symptoms and solution methods. Examining this structure enables us to meaningfully classify the very diverse instances of this genre, reveals key design issues, and can help us identify productive research questions. Complex troubleshooting procedures present the user with multiple symptoms. A set of symptoms may correspond directly to particular causes or may comprise a tree of symptoms. The resolution phase consists of one or more solution paths each consisting of one or more methods. When feasible, solution paths and methods should be variable rather than fixed sequences and should empower users to choose among solution paths. Keywords: writing, documentation, procedures, troubleshooting.

Introduction

Troubleshooting procedures are important and highly prevalent in the computer industry. They very often appear as "articles" in the KB (Knowledge Base) that comprises a core component of the support websites maintained by vendors of hardware and software products and web-based services. Help systems and manuals may also include troubleshooting procedures. Troubleshooting procedures are important in many other industries and subject areas, though they may go under different names. A first-aid manual, for example, is a set of troubleshooting procedures. My focus is troubleshooting procedures in the computer industry, in particular, complex troubleshooting procedures.

There is a significant, though scattered, literature pertaining to standard procedures but very little about troubleshooting procedures. They are familiar but largely unstudied. The professional organizations that are most directly associated with this specific technical communication genre are the Association of Support Professionals and the Technology Services Industry Association (TSIA/SSPA). These organizations conduct research and disseminate information regarding both support content and real-time phone and text dialogs. But their main concerns are the business dimension of technical support and the general features of support websites and call centers. There is little attention to the specific characteristics of troubleshooting procedures. My discussions with those who develop troubleshooting procedures suggest that they have developed their knowledge and skills largely on their own, aided by existing company practices and expertise.

I was extensively involved in a large-scale Microsoft effort to redesign their model for KB troubleshooting procedures during the spring and summer of 2007, and I have continued my investigation of this support genre since then. In this paper, I define troubleshooting procedures and briefly sketch out how they are developed. Then I analyze the genre's underlying architectural structure of diagnosis and resolution, showing both simple and complex configurations of symptoms and solution methods. These configurations are in part constrained by the nature of the technical problem; but they are also the consequence of design decisions. Understanding structure enables us to meaningfully classify the very diverse instances of this genre, reveals key design issues, and can contribute to experimental research insofar as structure is central to many of the most useful research questions we can ask.

Defining troubleshooting procedures

Standard procedures are task-focused. They state a user goal ("Encrypting files") and provide the steps for achieving this goal [1]. They assume a normally functioning system and assume, not always correctly, that the user is consulting the procedure as she begins the task. In contrast, troubleshooting procedures articulate and try to solve a problem other than the user's lack of familiarity with the normal operation of the system. In most cases this problem is a bug, incompatibility, or component failure:

When I save SWF files, they save with meaningless file names and the file sizes are unusually large

EZGrab 3.0 freezes or closes unexpectedly

My computer no longer plays audio or produces any sound from the speakers or headset

This distinction between troubleshooting procedures and standard procedures requires some refinement. First, some troubleshooting procedures (and other KB content) are written for situations in which the system *is* functioning normally. For example, a troubleshooting procedure may address an unexpected limitation of the product: A user cannot make something happen and thinks the product is malfunctioning, whereas the product was simply not designed to carry out this task. Second, in some cases, a user's lack of knowledge is framed as a troubleshooting problem and included in a KB: "I cannot encrypt files". This troubleshooting procedure, whose steps will closely resemble those of a standard procedure, serves the user who has gone to the KB on the assumption that a system problem is the reason she cannot encrypt her files. Finally, many standard procedures include a step or note that anticipates and addresses a minor impediment that will stymie some users [1]:

If you don't see the Format button, click More.

In this step, the introductory clause states a symptom and the main clause states the resolution. This step, then, is a "mini" troubleshooting procedure.

In certain instances, in particular when diagnosis is especially difficult, a preferred alternative to a troubleshooting procedure is a two-way dialog, perhaps via forum posts, email, live chat, or a telephone support call with the vendor's support technician. These dialogs, although "noisier" than carefully crafted KB content, also follow a diagnosis-resolution structure.

Developing troubleshooting procedures

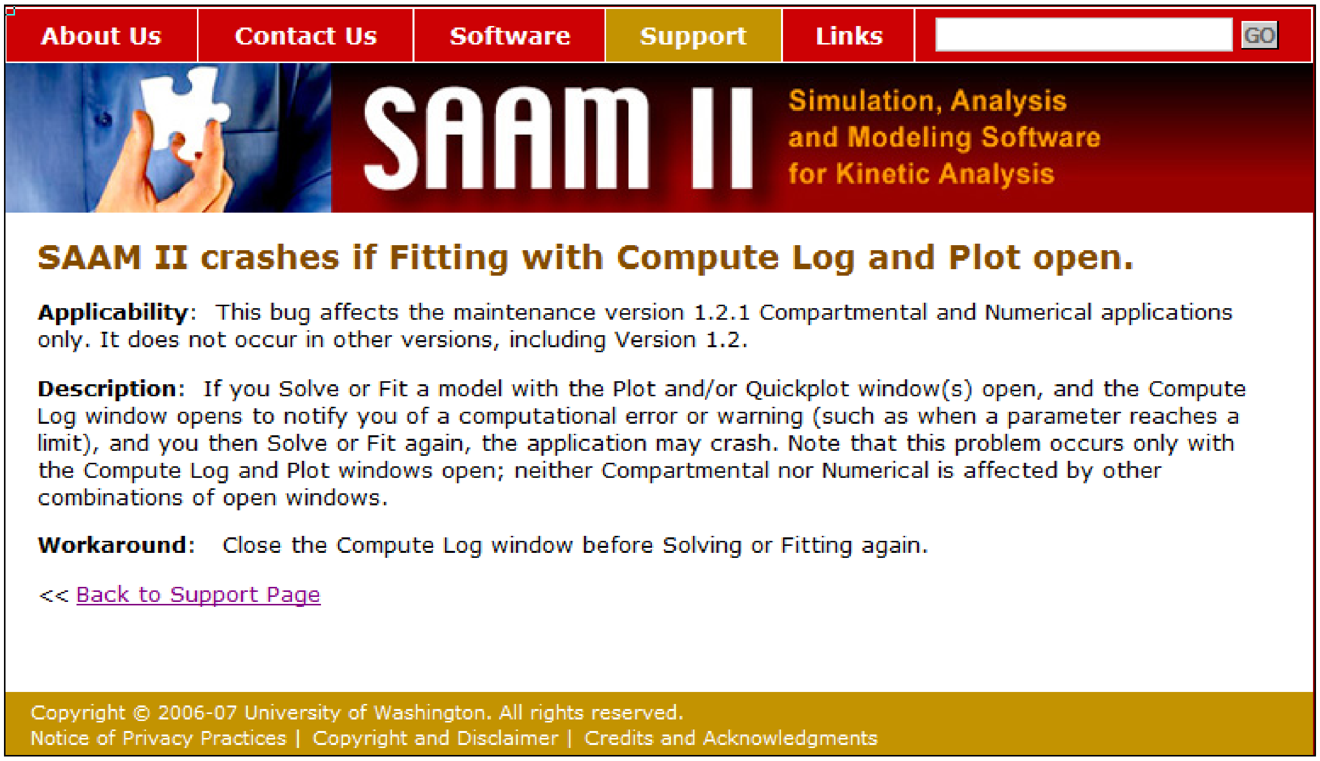
Although generalizations are difficult given the size and diverse nature of the computer industry, a broad sketch of the development process provides necessary context for the analysis that follows. The development process varies greatly according to such factors as the product or service, the user assistance genre (KB, help system, support bulletin, etc.), the company (size, budget, maturity of processes), the problem category and severity, and the range of users being served. One safe generalization is that when indications of a problem first reach a company, the problem must be analyzed, a plan must be devised for a troubleshooting procedure (and possibly other responses), and the content must be created and tested. This effort is in large part technical: For example, users who have upgraded to the newly released EZGrab 3.0—but not new purchasers of EZGrab 3.0—report that the product is saving damaged SWF files. It seems that the problem arises when users have previously saved a SWF file with certain other graphics applications. It will be necessary for the EZGrab company to determine the exact nature of this conflict and what solutions are possible—perhaps a more thorough uninstall of EZGrab 2.0, perhaps a change in the Windows Registry, perhaps downloading and installing a patched version of EZGrab 3.0. But the effort is also rhetorical: it is often a daunting challenge to create procedures that users, especially less sophisticated users, are able and willing to follow [2].

Because the development of troubleshooting procedures is both a technical and a rhetorical task, it is best carried out collaboratively by a range of professionals. Field reps, support technicians answering hotline calls, and forum moderators will likely have the most complete understanding of what brings on the problem, what exactly is going wrong, and what are the technical backgrounds of the various segments of the customer base. Developers are intimate with the product code. Writers and editors know how to present the information—how to encourage the user, how to reduce the effort needed to understand and follow the steps, and when and how to offer users alternatives in carrying out a procedure. Finally, writers should play a central role in designing the template or model to be used as new procedures are written, a model that is optimized for the kinds of troubleshooting procedures the company produces most often.

Diagnosis and resolution structure

Almost all troubleshooting procedures lead the user through phases of diagnosis activity and resolution activity. Either phase can be brief (sometimes very brief) or lengthy. The diagnosis phase may consist of a single symptom or a complex configuration of symptoms. The resolution phase may consist of a single method (a set of steps to take) or many methods in a complex configuration. One important distinction is that one or more methods comprising a distinct approach to solving a problem can be regarded as a solution path.

Although the main structural pattern is diagnosis followed by resolution, at times the diagnosis phase may contain steps for resolution and the resolution content may include diagnosis steps. That is, the procedure may switch the reader back and forth between diagnosis and resolution. Because understanding this diagnosis-resolution structure is the key to understanding troubleshooting procedures, I now show how this structure is manifested in a broad range of troubleshooting procedures.


About Us
Contact Us
Software
Support
Links
GO
SAAM II
Simulation, Analysis and Modeling Software for Kinetic Analysis

SAAM II crashes if Fitting with Compute Log and Plot open.

Applicability: This bug affects the maintenance version 1.2.1 Compartmental and Numerical applications only. It does not occur in other versions, including Version 1.2.

Description: If you Solve or Fit a model with the Plot and/or Quickplot window(s) open, and the Compute Log window opens to notify you of a computational error or warning (such as when a parameter reaches a limit), and you then Solve or Fit again, the application may crash. Note that this problem occurs only with the Compute Log and Plot windows open; neither Compartmental nor Numerical is affected by other combinations of open windows.

Workaround: Close the Compute Log window before Solving or Fitting again.

<< Back to Support Page

Copyright © 2006-07 University of Washington. All rights reserved.
Notice of Privacy Practices | Copyright and Disclaimer | Credits and Acknowledgments

Figure 1. A simple troubleshooting procedure in a help system [3].

The diagnosis phase

A user comes to a troubleshooting procedure with a problem. The problem may be slow performance or a crash. The problem may be continuous or occur only under certain conditions. The problem may be a strange sound or blinking light that is not currently affecting performance but suggests future trouble.

From the standpoint of diagnosis, the user's problem, including associated conditions that the user may or may not have recognized, are symptoms. The title of the procedure is part of the diagnosis phase and must state a key symptom (or a cluster of symptoms) in a way that the user can recognize. The goal of the diagnosis phase is to enable the user to match her symptom to a symptom described in the troubleshooting procedure. In so doing, the user and the procedure jointly achieve a diagnosis of the underlying cause that is sufficiently specific to direct the reader to one or more solution paths in the resolution phase.

Figure 1 shows a short and simple troubleshooting procedure in a web-based online help system. The title is a concise though not complete statement of the symptom: the system is apt to crash if the user tries to solve or fit a model with the Plot or Quikplot window open. A user who is experiencing crashes should be able to determine whether this statement corresponds to the behavior of her system. The Applicability and Description sections elaborate on the symptom as expressed in the title by specifying associated conditions: the problem pertains only to certain versions of the product, and the system is apt to crash only if the user attempts to solve or fit after receiving an error or warning message. Now the user can confirm whether the behaviors described in this procedure match the problematic behaviors of her system. If so, a diagnosis of the cause has been achieved and a simple work-around is provided. Note that in this procedure the cause is explained only at a superficial level: there is a bug. The SAAM II developers almost certainly understand the bug at the code level. In many procedures, however, a fuller explanation of the cause is necessary.

This troubleshooting procedure consists of a symptom (problem with associated conditions) that is specific and easy for the user to identify. Furthermore, it provides a single, reliable solution method (a simple work-around). More complex problems, however, present more difficult symptoms. Very often the symptom is broad in scope with numerous and disparate associated conditions and causes. For example, there are numerous reasons why a particular user may be unable to connect to a web-based service: a hardware malfunction (perhaps nothing more than an unconnected cable), a conflict between the client software and a process the user is running, a problem with the user's local area network, or just user error logging in. This broad symptom will be stated in the procedure title:

I cannot connect to the service

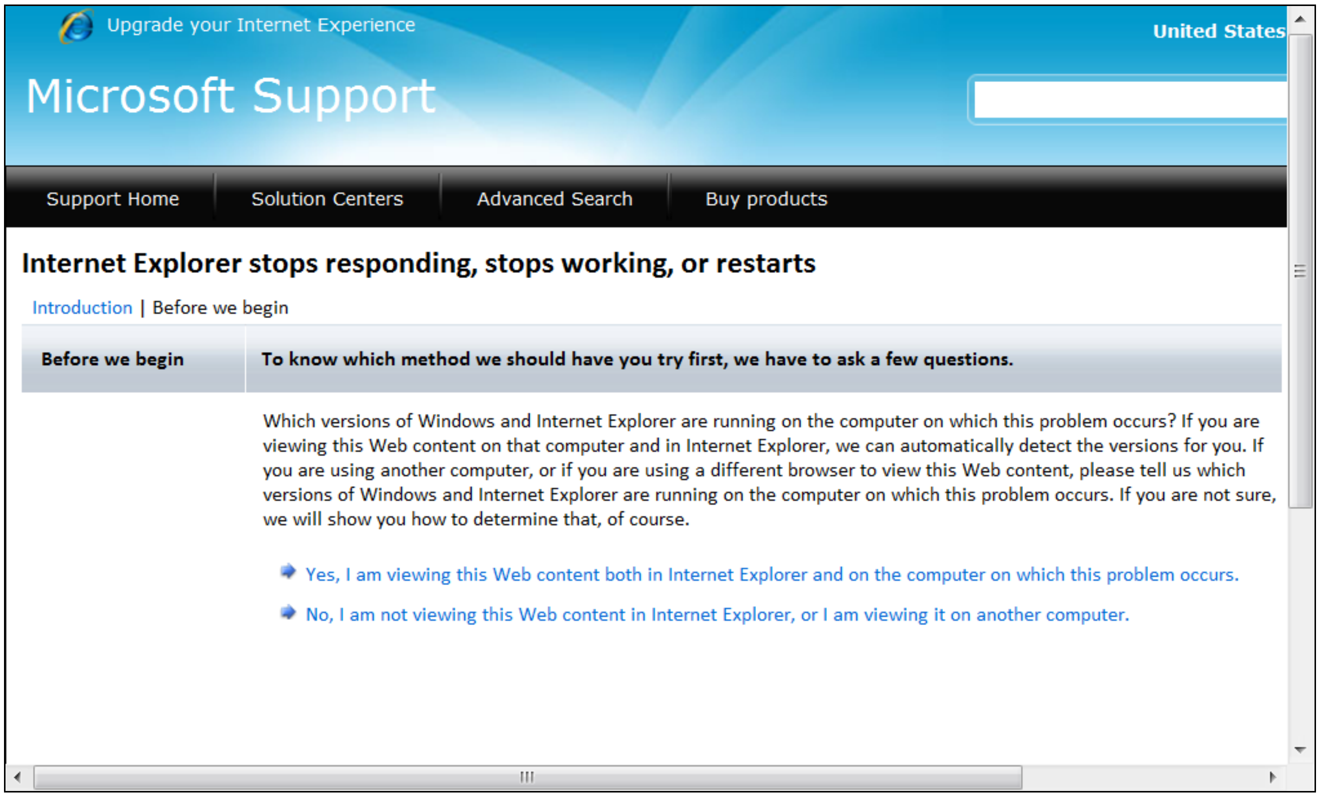
Following such a title there will very likely be a set of more specific symptoms. In many instances this list of more specific symptoms is a "flat list." By "flat list" I mean that these symptoms don't lead to further symptoms. Instead, each of these symptom has a clear-cut cause (perhaps even a one-to-one correspondence of symptom to cause) and therefore a solution path, possibly consisting of only a single method. A problem exhibiting a flat list of symptoms is a relatively favorable situation for those who are writing a troubleshooting procedure and ultimately for the users.

In other instances, the set of symptoms is divided still further into a tree-like structure. The user is asked to identify increasingly specific symptoms until she reaches a symptom that is specific enough to indicate a cause (or at least a set of candidate causes), at which point the procedure transitions to the resolution phase. Symptom trees can be effectively presented in a wizard-like sequence of panels. Figure 2 shows one of the initial panels in a wizard-like KB article that addresses problems in Internet Explorer. This KB article has special functionality. Assuming that the user is currently using the computer on which IE is causing problems, the next panel will identify which version of IE and which version of Windows the user is running and will ask the user to confirm this. In so doing, the KB article is moving down through the symptom tree and getting closer to the cause.

Among the symptoms that may appear in a troubleshooting procedure are error messages. Error messages, especially when they are specific to a particular problem, are very useful symptoms.

If a problem exhibits two very different symptoms, it will be necessary to write two entirely different troubleshooting procedures. For example, if EZGrab 3.0 becomes damaged, it may save abnormal SWF files (let us say with meaningless file names and very large file sizes), and it may close unexpectedly even when the user is not trying to save SWF files. Because the user may encounter or may notice only one of the two symptoms, it will be necessary to write two troubleshooting procedures, each with a title that corresponds to one of these very different symptoms.

On the other hand, there are many instances when a problem exhibits symptoms that can be readily described together: Magic Accountant closes unexpectedly or freezes. Now the option to write either one or two procedures—no longer constrained by the nature of the technical problem—becomes a rhetorical decision. Let's



Internet Explorer stops responding, stops working, or restarts

Introduction | Before we begin

Before we begin | **To know which method we should have you try first, we have to ask a few questions.**

Which versions of Windows and Internet Explorer are running on the computer on which this problem occurs? If you are viewing this Web content on that computer and in Internet Explorer, we can automatically detect the versions for you. If you are using another computer, or if you are using a different browser to view this Web content, please tell us which versions of Windows and Internet Explorer are running on the computer on which this problem occurs. If you are not sure, we will show you how to determine that, of course.

- Yes, I am viewing this Web content both in Internet Explorer and on the computer on which this problem occurs.
- No, I am not viewing this Web content in Internet Explorer, or I am viewing it on another computer.

Figure 2. A wizard-like troubleshooting procedure that begins with a symptom tree [4].

imagine that each of these Magic Accountant symptoms has a different cause and a different solution path. A single procedure will be relatively long and complex; on the other hand, reducing the number of articles, especially in a large KB, makes it easier for users to search the KB and find the most appropriate article.

The resolution phase

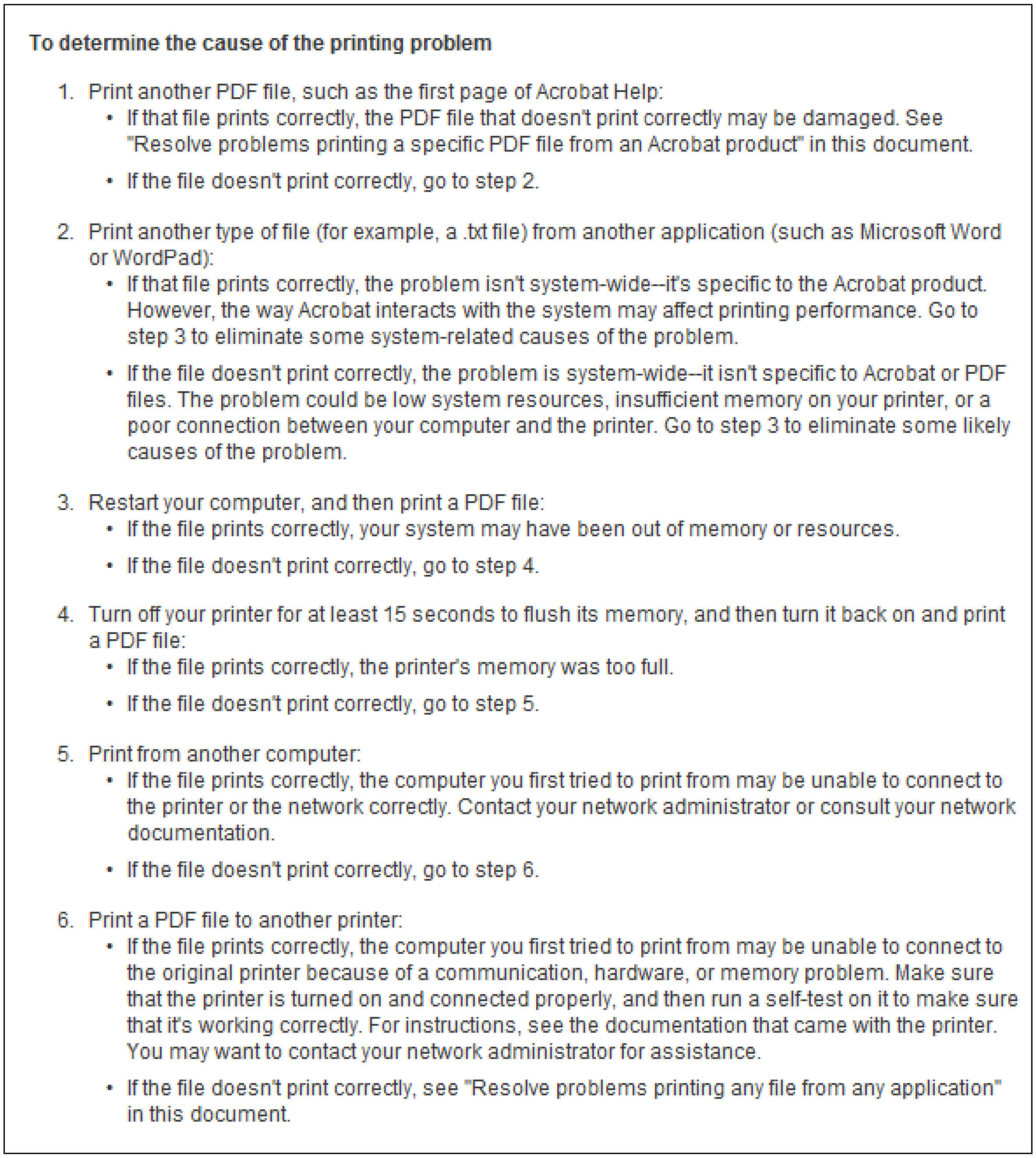
As noted above, the resolution phase consists of one or more solution paths (a particular approach to fixing the problem). There is always one solution path, even if it is nothing more than a single one-step method. Often, however, there are multiple solution paths, and a solution path may consist of a sequence (perhaps a lengthy sequence) of methods. For example, if the client software for a web-based subscription music service is exhibiting a particular symptom (e.g., playing the wrong song), one solution path for addressing that problem might consist of three lengthy methods intended to repair the corrupt database in the client software. The user will first follow Method A. If Method A is not successful, the user is directed to Method B. If Method B is not successful, the user is directed to Method C, which may also fail. The second solution path consists of a single method: downloading new client software. The second solution path is easier and is very likely to succeed, but the user will lose her playlists. The KB article, therefore, starts with the first solution path.

Complexity in the resolution phase very often arises when the exact cause of the user's problem cannot be pinpointed in the diagnosis phase. If a single, specific cause can be identified, it should be possible to provide only a single method. (When you know exactly what has gone wrong, you know exactly what—if anything—can be done about it.) Unfortunately, however, there are apt to be many more user variables than it is feasible to describe as symptoms in the diagnosis phase. It is impossible to know all the system states that might arise from the user's hardware, operating system, applications (e.g., another subscription music service), configuration of the software, and so forth. In a sense, then, providing multiple methods is another means to get at the cause. Broadly speaking, a method that (when properly followed) does not succeed is a kind of belated diagnosis; it rules out (or, at least, argues against) a suspected cause. In some instances, therefore, it makes sense to abbreviate the diagnosis phase in favor of a lengthier resolution phase.

The sequence of methods within a single solution path (or the sequence of solution paths) may be fixed (as described above) or else variable. That is, it is possible that a certain outcome of Method A means that Method B is useless and that the user should bypass Method B and go directly to Method C—or to an entirely different solution path. This distinction between fixed and variable sequencing has a direct analogy in medical treatment. A physician may have a fixed regimen of treatments for patients suffering from a particular illness. In other cases, the sequence is flexible; the outcome of Treatment A (perhaps a lower white blood cell count) dictates moving directly to Treatment C. To design variable sequencing requires a more specific understanding of the cause of the problem. But there is much to be said for keeping the user from following methods that have no chance of working.

Thus far, we have been considering instances in which the diagnosis and resolution phases are distinct. But this is not always the case. For example, it is possible that an outcome of a method necessitates further diagnosis. In other words, diagnosis steps may be embedded in the resolution phase. In other cases, resolutions may be embedded in the diagnosis phase. Figure 3 is the first section of a long, complex Adobe Support Center TechNote for Adobe Reader and Acrobat. It addresses the broad symptom of PDF files that do not print. This section narrows the symptom by trying to pinpoint one of several associated conditions: Is it a general printing problem? Is it a problem with the Adobe product that generated the PDF? Is the problem limited to this PDF file? To narrow the symptom, the user must perform tests, some of which resolve the problem. We see, then, that resolution actions (restarting, turning off the machine) are embedded in the diagnosis phase. In still other cases, other kinds of content will be embedded either in the diagnosis or solution phase; for example, a certain symptom or a certain outcome of a method may dictate an interim task such as gaining administrator rights.

We have thus far been considering resolution configurations in which the procedure dictates the user's next action. But it is often highly desirable to empower the user to choose among solution paths. Individual solution paths entail trade-offs among time and effort, likelihood of success, risk of creating new problems, and the nature of the resolution. When feasible, users should be invited to make these choices. For example, in the case of the music subscription service, the user should be fully informed and empowered to immediately download the new client rather than try to repair the corrupt database. A particular user may not have downloaded many tracks and playlists and so may not want to go through numerous steps in an effort to repair the corrupt database. If the goal of a troubleshooting procedure is to repair a damaged word processing file, certain methods may entail the loss of formatting while others promise to retain formatting. Someone trying desperately to save the text of his novel will be willing to lose the formatting and will try every available solution. Someone trying to save an elaborately formatted document may only be interested in methods that retain the formatting. To return to the medical analogy, physicians should inform patients about the available treatment options and empower patients to make their own choices.



To determine the cause of the printing problem

1. Print another PDF file, such as the first page of Acrobat Help:
   - If that file prints correctly, the PDF file that doesn't print correctly may be damaged. See "Resolve problems printing a specific PDF file from an Acrobat product" in this document.
   - If the file doesn't print correctly, go to step 2.
2. Print another type of file (for example, a .txt file) from another application (such as Microsoft Word or WordPad):
   - If that file prints correctly, the problem isn't system-wide--it's specific to the Acrobat product. However, the way Acrobat interacts with the system may affect printing performance. Go to step 3 to eliminate some system-related causes of the problem.
   - If the file doesn't print correctly, the problem is system-wide--it isn't specific to Acrobat or PDF files. The problem could be low system resources, insufficient memory on your printer, or a poor connection between your computer and the printer. Go to step 3 to eliminate some likely causes of the problem.
3. Restart your computer, and then print a PDF file:
   - If the file prints correctly, your system may have been out of memory or resources.
   - If the file doesn't print correctly, go to step 4.
4. Turn off your printer for at least 15 seconds to flush its memory, and then turn it back on and print a PDF file:
   - If the file prints correctly, the printer's memory was too full.
   - If the file doesn't print correctly, go to step 5.
5. Print from another computer:
   - If the file prints correctly, the computer you first tried to print from may be unable to connect to the printer or the network correctly. Contact your network administrator or consult your network documentation.
   - If the file doesn't print correctly, go to step 6.
6. Print a PDF file to another printer:
   - If the file prints correctly, the computer you first tried to print from may be unable to connect to the original printer because of a communication, hardware, or memory problem. Make sure that the printer is turned on and connected properly, and then run a self-test on it to make sure that it's working correctly. For instructions, see the documentation that came with the printer. You may want to contact your network administrator for assistance.
   - If the file doesn't print correctly, see "Resolve problems printing any file from any application" in this document.

Figure 3. A complex symptom tree that includes resolution actions [5].

Although there are limits to the number of solution paths and methods that can and should be provided, the scope of troubleshooting procedures should not be unrealistically narrow. For example, an internet service provider (ISP) consistently receives reports that some of their customers can open but not directly download email attachments. Without much investigation, the ISP responds with a troubleshooting procedure stating that the problem lies with the user's virus protection software. But the ISP is choosing to ignore (at least temporarily) the possibility that there is another reason why customers are encountering this problem.

In many instances, especially when the problem was brought on by a user error, it is necessary to explain how to prevent a recurrence, which is often equivalent to explaining the cause of the problem. For example, the EZGrab company has learned that the problem with EZGrab 3.0 occurs when some users, wanting to run both EZGrab 2.0 and 3.0, circumvent the procedure for uninstalling version 2.0. The EZGrab troubleshooting procedure, therefore, must not only fix the problem but must make clear that users cannot run both versions of EZGrab.

In some cases, verification steps or a complete verification method is part of the resolution phase. For example, it may be advisable to guide the user through repeating the actions that brought on the problem or perhaps restarting their system. Through this means, the user will either confirm that the procedure was a success or will learn that it was not—in which case the verification steps will hopefully direct the reader to a promising solution path. In some cases, each method concludes with one or more verification steps; alternatively, each method may conclude by directing the user to a single verification section. This decision to create a separate verification section is one more variation in the diagnosis-resolution structure of troubleshooting procedures

When all solution paths fail, the procedure may direct the user to another resource, such as another troubleshooting procedure or phone support. At times it is necessary to express regret that nothing further can be done to solve the problem.

Conclusion

Troubleshooting procedures, even brief ones, exhibit a complex architecture based on diagnosis and resolution. This architecture reveals important underlying similarities among procedures that may look very different from one another. For example, two seemingly diverse procedures with different formatting and other characteristics might both employ a tree (or a flat list) of symptoms, a variable (or fixed) sequence of solution paths, or resolution content embedded in the diagnosis phase. The structural perspective, then, is like an X-ray view into the architecture of troubleshooting procedures.

Furthermore, the structure of troubleshooting procedures will be central to many of the most useful research questions we can ask and many of the most important design decisions. For example, when do numerous methods, variable sequences, and the empowerment of users to make their own choices become overly burdensome? Given the need to motivate users to follow procedures [2] [6], how does the architecture of troubleshooting procedures affect motivation? What is the relationship between empowering users to make their own choices and users' perceptions of the rhetorical stances we assume when we write troubleshooting procedures [7] [8]?

The diagnosis-resolution structure is closely tied to modularization, an important direction in the design of troubleshooting procedures. A promising means to train new writers, especially those who have stronger technical than rhetorical skills, is to explain troubleshooting procedures as a set of modules, consisting of mandatory and optional components that have specified characteristics. A modular approach also facilitates document re-use, including incorporating parts of a troubleshooting procedure—in particular a solution path or method—into a forum post that responds to a user's query. Finally, users may well benefit from modularized presentation in which modules and their components are visually distinct and have clear-cut roles.

When we modularize, we have many choices. For example, do we write a single verification section (a highly modular approach) or else add verification steps to multiple methods? When does the resolution phase become one large module and when do we modularize at the more granular level of solution paths? Almost certainly, as we entertain options for modularization, we will find that the basis for modularization will be the underlying architecture of diagnosis and resolution.

References

[1] Farkas, D.K. The Logical and Rhetorical Construction of Procedural Discourse. *Technical Communication*. 46(1): 42-54, 1999.

[2] Douglass-Olberg, C., D.K. Farkas, M. Steehouder, J.D. Kieras, A. Roesler, N. Dalal, R. Baker, and D. Brunet. The New Face of Procedural Content: A Real World Approach, extended abstract, *Proceedings of SIGCHI 2008*. Florence, Italy, April 7-10, 2008.

[3] SAAM II Help System.
http://depts.washington.edu/saam2/support/screencrash.html, accessed 4/19/10.

[4] Microsoft Support.
http://support.microsoft.com/gp/pc_ie_start, accessed 4/19/10.

[5] Adobe Support Center TechNote. Troubleshoot Printing Problems in Adobe Reader 9 and Acrobat 9 (Standard, Pro, Pro Extended) in Windows.
http://kb2.adobe.com/cps/403/kb403914.html, accessed 4/19/10.

[6] Loorbach, N., M. Steehouder, and E. Taal. The Effects of Motivational Elements in User Instructions. *Journal of Business and Technical Communication*. 20(2): 177-199, 2006.

[7] Coney, M.B. Technical Readers and Their Rhetorical Roles. *IEEE Transactions on Professional Communication*. 35(2): 58-63, 1992.

[8] Coney, M.B. and C.S. Chatfield. Rethinking the Author-Reader Relationship in Computer Documentation. AMC *SIGDOC Asterisk Journal of Computer Documentation*. 20(2): 23–29, 1996.

About the Author

David K. Farkas (http://faculty.washington.edu/farkas) is a Professor in the Department of Human Centered Design & Engineering at the University of Washington, Seattle (US). His main professional interests are software user assistance, presentation graphics (slideware/PowerPoint), consumer-decision infographics, and the design of innovative documents (QuikScan, SwitchBack). He served as the program chair or co-chair of this conference in 1988, 1994, and 2007.