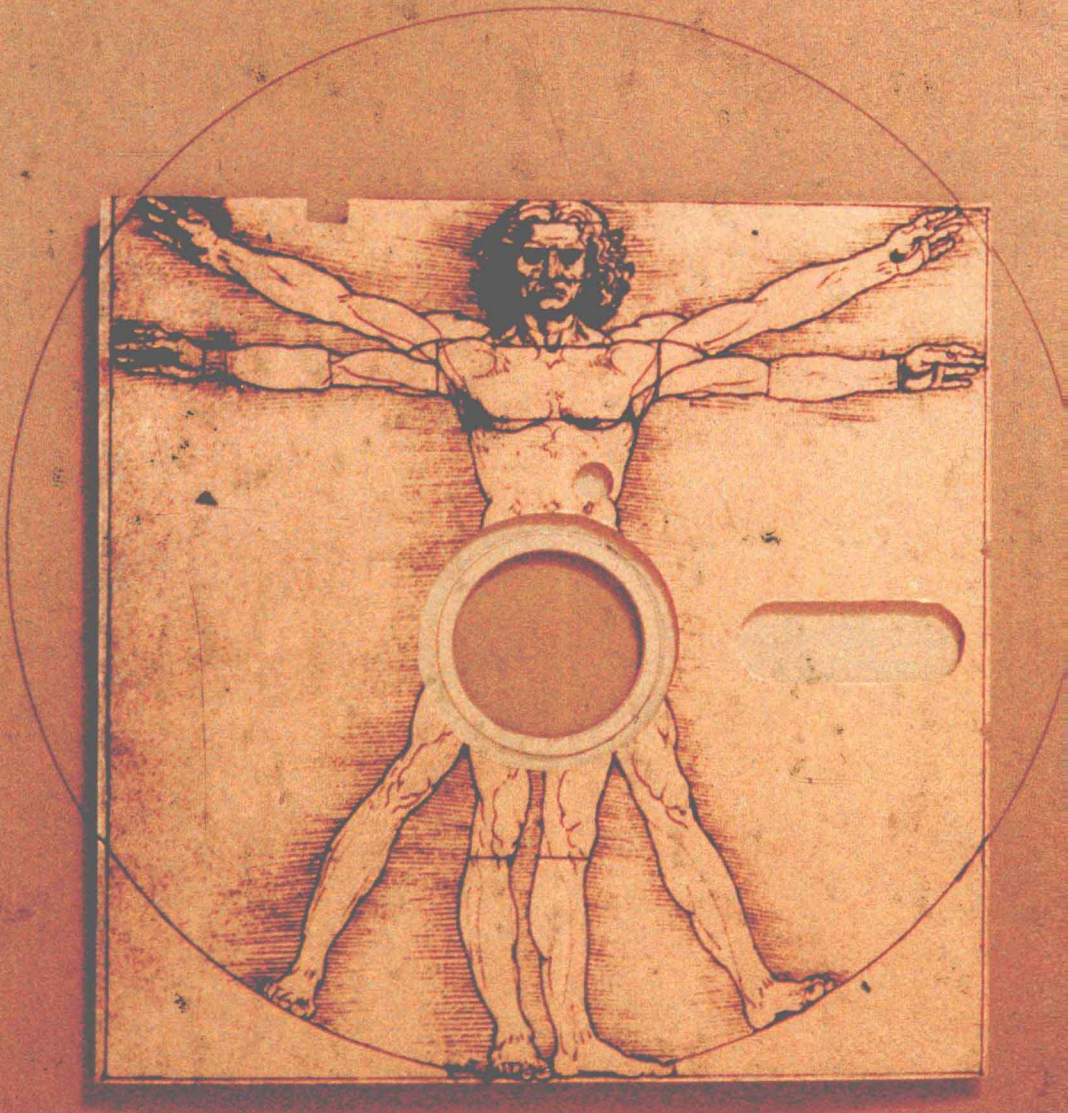
Developing Effective User Documentation

A Human-Factors Approach

Henry Simpson Steven M. Casey



Developing Effective User Documentation

A Human Factors Approach

Henry Simpson

Steven M. Casey

Library of Congress Cataloging-in-Publication Data

Simpson, Henry.
Developing effective user documentation.

Bibliography: p.
Includes index.
1. Electronic data processing documentation.
2. Computer software—Development. I. Casey,
Steven M. (Steven Michael), 1952– . II. Title.
QA76.9.D6S56 1988 005.1'5 87-29690
ISBN 0-07-057336-0

1234567890 DOC/DOC 8921098

ISBN 0-07-057336-0

The editors for this book were Theron Shreve and Marlene Hamerling, the designer was Naomi Auerbach, and the production supervisor was Richard Ausburn. This book was set in Century Schoolbook. It was composed by the McGraw-Hill Book Company Professional and Reference Division Composition unit. Printed and bound by R. R. Donnelley and Sons Company.

Preface

This book was written for people who want to develop effective user documentation for computer programs. It was written for technical writers, editors, documentation managers, programmers, and others involved in documentation development. User documentation is the hard-copy or on-line documentation provided with an application program. It tells the user how to get the program up and running, trains the user in its operation, and provides reference information once the user becomes experienced. It may do additional things as well—such as help build the user's confidence, entertain, or sell the user on the merits of the program—but these are incidental. The main thing is to allow the user to make the program work.

User documentation comes in many different forms. Most people think of it as the manual that comes with a computer program, usually with a title containing the words "user's guide" or "reference manual." The term also applies to quick reference cards and guides, job performance aids, computer-based tutorials and help screens, keyboard overlays, and anything else that will support the user. The various documentation possibilities are complementary rather than competing alternatives. Each has its strengths and weaknesses. Deciding which types to develop to support an application requires thought and analysis. This book is designed, in part, to help you make the right decisions.

A few words about how this book is unique. First, it deals with user documentation from a human factors, rather than a technical writing, viewpoint. It will help you make decisions concerning what type of documentation your program needs, its content, its format, and how it (like the computer program it supports) should be developed, debugged, evaluated, and refined.

Second, it assumes that you already know how to write; it does not dwell on spelling, punctuation, or grammar. These matters are left to texts on expository or technical writing.

Finally, it shows how you can use the tool whose software you are

documenting (the computer) to support the documentation effort. A software revolution has occurred in the last few years, and many of its products—word processors, graphics programs, desktop publishing programs—can make life easier for the documentation developer. Ironically, much of the publishing industry still works with the technology of the past—typewriters, tape, scissors, rubber cement, and so on—and is only slowly beginning to make use of the computer.

This book is divided into 12 chapters. Chapter 1 provides an overview of the various types of user documentation. Chapters 2 and 3 focus on the user and show how to make the "user-documentation match." Chapter 4 describes a systematic process for developing documentation. Chapters 5 and 6 describe tools for management and production during a documentation development project. Chapters 7 through 11 focus on documentation, with discussions of design, architecture, graphics, documentation models, and on-line documentation. Chapter 12 tells how to test, evaluate, and validate documentation.

The best strategy for using this book is to start at the beginning and read straight through to the end. Regardless of the role you play in documentation development—manager, editor, writer, or other interested party—all of the material contained in this book is relevant. It is our sincere hope that it will help you develop better user documentation.

The authors gratefully acknowledge the help of the following software companies for answering questions and providing review software: American Small Business Computers, Ashton-Tate, Breakthrough Software, Communication Dynamics, Computer Associates, Greene-Johnson, Hayden Software, InfoStructures, Innovative Data Design, Lifetree Software, Living Videotext, Manhattan Graphics, MaxThink, Microsoft, North American Mica, Perceptronics, Simon & Schuster, SoftCorp, SSI Software, Symantec, T/Maker Graphics, Target Software, and Writing Consultants. Special thanks to SSI Software, Microsoft Corporation, and Perceptronics for permitting the reproduction of several pages of their documentation. We would also like to thank Dick Noffz, documentation editor, for his helpful commentary. Finally, thanks to the many researchers and authors whose research, analyses, and opinions were incorporated into this book.

Henry Simpson
Steven M. Casey

Contents

Chapter

1

Overview of User Documentation

This chapter defines terms and presents ideas that will be used throughout the book. The first section briefly discusses the evolution of user documentation. The second section describes some common reasons user documentation fails when put to the test in the real world. The third section discusses three different ways of classifying user documentation—system versus user, procedural versus reference, and internal versus external. The fourth and fifth sections provide an overview of the common forms of user documentation. The sixth section discusses documentation systems—the set of documentation components used to support a program. The final section describes the three elements of documentation design. Incidentally, in this and later chapters the terms "user" and "operator" are used interchangeably; they mean the same thing.

Evolution of User Documentation

User documentation is extremely important. It is accurate to say that, for most programs and users, user documentation will make the difference between whether a program can be used effectively or not. In fact, it is reasonable to go a step further and say that, if the program is not documented properly, it may not be usable at all. It follows from this that it is only sound practice to give as much attention to user documentation as to the program itself. Doing otherwise is perilous.

Awareness of the importance of user documentation has undergone a steady evolution over the years. This is due in large part to the increas-

ing growth of the market for software products, particularly microcomputer software. Software publishers have discovered that programs people can use sell more copies. Good user documentation makes programs usable; ergo, there is more interest in making it good, or at least look good. User documentation is increasingly becoming a selling point for software products.

Precedent and history leave much room for improvement. The documentation provided with most computer programs has not been distinguished by its completeness, quality, or ease of use. In fact, much of it to date has been awful. The situation does appear to be changing, however. In the early days of computers, documentation might be written by an engineer or technician who had little writing skill or understanding of the requirements for communicating with the reader. Thankfully, this era of user documentation—its sort of "Dark Ages"—now seems to be passing. The documentation provided with software products is much better today than it was in the past.

Still, the outlook is not particularly bright. The documentation departments of many firms—if such departments exist at all—have little prestige and even less influence on the engineering departments that create software. Technical writers spend a lot of time running after engineers attempting to obtain information needed to create user documentation. One still sees ads for technical writers followed by the words "experience preferred but not essential." Pay scales leave little doubt why few people major in technical writing in college.

One of the authors once became involved with a trade magazine, the management of which had a similar view of its editorial staff. The publisher was literally incapable of constructing a grammatical sentence, but was a terrific ad salesman and made several times the salary of the magazine's editor. The publisher once explained that management justified the disparity on the basis that many people could write, but few could sell. However unfair, this is sound thinking from a business viewpoint, and also shows how little esteem many people have for those who use the English language and their imaginations to explain technical mysteries to readers. It may not represent the prevailing view of the importance of good technical writing, but it does represent a fairly widely held view. We may take some comfort from the fact that the situation seems to be improving.

For all the optimistic signs, it probably remains true that the writing of user documentation is widely considered to be of less importance than the writing of the software that the documentation describes. The attitude is understandable. Think about it the next time you are flying in a jetliner; ask yourself how important manuals are to pilots, navigators, and mechanics. Likewise, as you are being wheeled into the operating room to have your appendix removed, consider the utility of textbooks describing surgical procedures.

Why Documentation Fails

There are several common reasons the creator of a product may develop poor supporting documentation. As you read, see if you have direct experience with any of these problems on the documentation projects on which you have worked.

A Documentation Editor's Perspective

The documentation editor for a major mainframe computer manufacturer identified four major problems with the user documentation developed within his organization. While the problems he identified are not necessarily universal, they are typical.

First, much user documentation is written at the wrong level. The usual case—especially if documentation is prepared by programmers or other technical professionals—is for it to be too technical, too complex, and to assume too much knowledge on the part of the reader. The opposite—writing material at too simple a level—can also occur but is relatively rare. As emphasized elsewhere in this book, the first step in developing effective user documentation is to have a good understanding of the user and, ideally, to conduct function and task analyses that allow definition of user information requirements (see Chapter 3).

Second, many user documents are actually technical documents. That is, instead of helping the user understand the program and how to use it, they tend to focus on the mechanics of software and the methods by which operations are performed. Often, the document is written in the technical language of its author—and not well suited to the end user. The editor said he spent much of his time modifying such material to suit the needs of the users.

Third, many user documents suffer from the "multiple-author syndrome." The authors of different parts of the document may have contrasting styles, use terms differently, discuss the same topics, cover topics at different depths and breadths, and so forth. Often, working these separate contributions into a unified document requires a major editorial effort (as the authors of the present book are well aware, incidentally). In the best of all possible worlds, a given user document has a single author. Since most of us live in one of the other possible worlds, this ideal is seldom achievable. However, negative consequences of multiple authors can be reduced considerably by adequate planning, outlining, and developing documentation standards (see Chapters 3, 4, 5, and 7).

Fourth, many documents suffer from inconsistent terminology—the practice of describing the same thing with different words. Related to this is the use of the same word to mean different things. These problems occur with single authors but are compounded when multiple authors are involved. Prevention is probably impossible, but effects can be mini-

mized by stressing the use of concrete language, minimizing the use of technical terms, and careful outlining. It is absolutely essential that an editor work through such materials to correct the inconsistencies before the document is released to the world.

Resource Allocation

The most inexcusable reason for poor documentation is the belief on the part of its developer that such documentation is of secondary importance and does not justify the expenditure of the monetary, personnel, and time resources necessary to produce a first-rate product. The developer might follow the line of reasoning that computer programs are complex things and the document user must expect to work hard to learn what is necessary to operate the program successfully. Nothing, of course, is further from the truth. In recent years, as software has reached the mass marketplace and come into the hands of many unsophisticated users, there has been an increasing design emphasis on ease of use. Inadequate documentation undercuts this whole philosophy. Indeed, many a good program has been sunk by poor documentation.

The "Myopia Syndrome"

Related to the problem just cited is the software expert's familiarity with his or her product. This might be termed the "myopia syndrome." Software developers become so familiar with their products that they fail to approach the problems of learning and operating a program from the perspective of the new user. When developers are solely responsible for documentation development and that documentation is not evaluated properly by outside users, the chances are good that the resulting documentation will not be adequate. Common problems are the use of unnecessarily complex language, incomplete coverage of topics, and a tendency to assume that the reader knows more than should be expected. The foregoing is not meant as a condemnation of system developers as documentation developers. The real issue is a frame of mind, which in simple terms amounts to an appreciation that the user comes to a new program with much less knowledge than its developer. This gap must be crossed if documentation is to do its job.

Time

Another common reason for poor user documentation is that it is often developed late in the product development cycle, perhaps even after the software itself has been released (Mosier, 1984). Time is short, deadlines are tight, and the documentation is developed under pressures that mitigate against

adequate planning, quality control, user field testing, and other factors important in a documentation development project. The earlier documentation is developed, the better. The converse is also true.

Documentation Development Skills

Quite often, documentation is developed by people who lack the requisite skills for a successful product. It is common to point the finger at programmers and engineers who may, in some cases, produce user documentation themselves or provide the source documentation from which technical writers work. There is an old cliché about engineers not being able to write. One of the authors of this book is an engineer and has heard it often enough. There may be some truth in it, as there is in the notion that all psychologists are a little crazy (the other author of this book is a psychologist), but such sweeping statements really miss the point of why engineers do not usually produce good documentation. They tend to consider — and rightly so— that writing is not their primary function. As is widely acknowledged, they have other primary functions—designing things, debugging them, making them work properly—and writing documentation comes second, if at all. If the engineer has the inclination, time, motivation, and writing skills, he or she may be the ideal author of user documentation.

Who, then, writes the documentation? Technical writers, those "poor, harmless drudges" of the computer industry, as Samuel Johnson might have put it. The skills of these writers vary greatly but, as in most things, you get what you pay for. If you hire cheap, you get the kid with the English degree whose novel was unpublishable, the technician who was unable to move up a rung, or the out-of-date engineer who was unable to hack it in the first profession.

If you are willing to pay for talent, you may hope to find a professional who knows how to put the words together, has an appreciation of documentation as a learning and teaching tool, and who understands that adequate analysis of users and the tasks they must perform underlies all good documentation. In fact, it is fair to say that writing skill, though important, is probably not as important as understanding the last two factors just cited: creating learning and teaching tools, and being able to respond to users' needs in developing documentation.

As a sort of footnote and for whatever it's worth, the acquisitions editor for the publisher of this book, upon receiving the authors' book proposal, remarked that he received many such proposals from people who wanted to write books on how to prepare user documentation and that most of them were badly written. The point is not that we had written such a great proposal—it went through three revisions before finally being accepted—but that so many folks who ply the writing trade aren't good writers. If there's a message there, it's a disquieting one.

Obsolescence

Documentation often fails because it becomes outdated. Printed documentation may be difficult to modify or update because of structure or layout, or because of the way it is generated physically. As we all know, no one uses typewriters, editor's scissors, rubber cement, and such any more for creating and editing documents. (Of course not!) These are outdated techniques that make it terribly inefficient and time-consuming to create and edit documentation. Now we have the word-processing program, along with various automated production tools, graphics packages, and other software, to make the developer's life easier. However, not everyone has yet gotten the word.

To give an actual example, one of the authors recently visited a two-year-old computer company that is developing a very advanced minisupercomputer. An outside consultant had determined that at least 20 separate documents would be required to support this new computer. These included installation manuals, hardware manuals, user's guides for various programming languages, and so forth. The company's documentation manager (actually, the marketing director) planned to hire a team of technical writers and to have them produce text on microcomputers using a popular word-processing program, have the copy typeset, have the artwork produced by a consultant, and then have the documents offset-printed by a contractor. (Incidentally, the word-processing program, which shall remain nameless, is one of the most popular ones, also widely acknowledged to be nonintuitive, difficult to use, and to take a long time to master. Like lemmings, a lot of people continue to use it—presumably, because a lot of other people continue to use it. See Chapter 6 for an in-depth discussion of word-processing software.)

The proposed documentation development method is a great improvement over using typewriters and such. Still, it presents problems in terms of updating because so many separate parties are involved in document production. It would make much more sense to use a desktop publishing program and to produce all documentation in house. But that, as they say, is a new idea in the documentation world. The irony of the situation is that the system being documented is one of the most advanced on the drawing board. (Desktop publishing is discussed in detail in Chapter 6.)

Documentation Development Process

Perhaps the most common reason for poor documentation is the failure to spend the necessary time and funds on the documentation development process (Hendricks, Kilduff, Brooks, & Marshak, 1982) (Figure 1.1). Too often a piece of documentation—for example, a user's manual—is thought of in isolation and regarded as the product solely

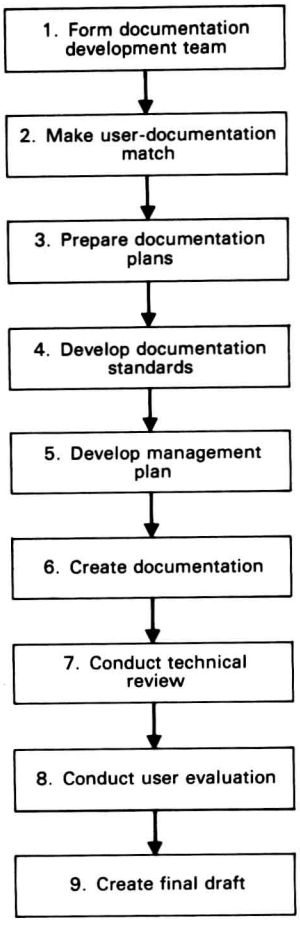


Figure 1.1 Documentation is most effective when developed systematically, according to a well-thought-out documentation development process.

of writing. To develop proper documentation, considerable front-end work is required beforehand; writing should be the last thing done. Documentation objectives must be defined. A documentation plan must be developed. The types of documentation for the project must be chosen. The various user groups must be specified, and documentation must be tailored to satisfy their needs. All of these things should be done before the writing begins, and all too often they are not. In short, the documentation development process entails more than writing. Chapters 3 and 4 discuss the documentation development process in detail; Chapter 2 discusses users.

Documentation developers often fail to assess the needs of users before developing documentation and may fail to do the other front-end analyses required before putting pen squib to paper (or fingers to keyboard). Typically, they are not willing or able to expend the time and funds needed to develop and maintain good documentation. Documentation developers must realize that their task entails more than writing. More importantly, those responsible for allocating resources to documentation development projects must understand the process themselves and allocate the resources necessary for successful products. Without this appreciation, the documentation will be inadequate. Poor documentation influences the user's perception of both the documentation and the software being documented. Write a bad manual, and its user is liable to throw both it and the software out the window or, more likely, send them back to the manufacturer for a refund.

Documentation Systems

The larger and more complex the program, the more likely it will need several separate documentation components to fulfill the various needs of its audience. The documentation developer must decide what components are needed and the function to be fulfilled by each (Hendricks, et al., 1982; Mosier, 1984). Together, the various components comprise a documentation system (Figure 1.2).

A documentation system is a system in the same sense as any other type of system—including a computer system. A system, formally defined, is a set of interdependent components that work together to achieve a goal. (Documentation systems are discussed in detail later in

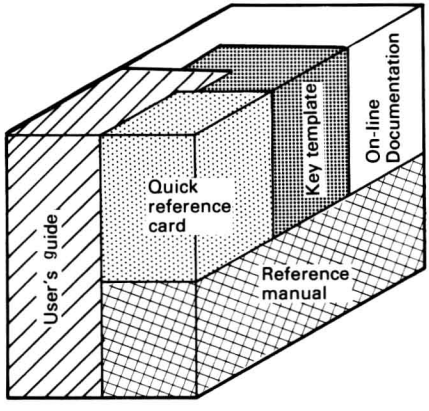


Figure 1.2 A documentation system is a set of interdependent documentation components, each with an assigned role.

this chapter.) Documentation developers frequently fail to recognize the separate roles and interdependent nature of the various document components they develop. For example, they may fail to identify certain components needed for a product or assume that a particular type of document will fulfill the needs of another. For example, it is quite common to open a new software package and discover that the documentation consists of a reference manual—without a tutorial, quick-reference material, or other documentation components needed to learn how to use the program and use it efficiently once the learning phase is over. Obviously, making this mistake can have serious consequences.

Some Ways to Classify User Documentation

This section distinguishes between system and user documentation, procedural and reference documentation, and internal and external documentation.

System Versus User Documentation

There are two classes of program documentation (Figure 1.3). The first is documentation intended for the programmers who create, maintain, troubleshoot, and update an existing program. This documentation consists of such elements as program listings; tables of variables, functions, and procedures; written explanations of parts of the program; and remarks in program code. This documentation is prepared by the program's authors and maintainers and is referred to as "system documentation." Beyond the description just given, this book does not discuss system documentation.

The second type of documentation is "user documentation." This may be in written or other forms. It may be provided outside the program, appear within the program, or be a combination of both. Such documen-

	System documentation	User documentation
For:	Programmers	Users
Purpose:	Design	Operation
	Maintenance	
	Troubleshooting	
	Updating	
Contents:	Program listings	Program explanation
	Tables of variables, functions, procedures	Program operation
	Written explanations	Reference information
	Remarks in code	

Figure 1.3 The audience for, purpose of, and contents of system and user documentation differ.

tation must explain the program, initiate the new user, and provide the reference information the experienced user needs. This type of documentation is the subject of this book.

Procedural Versus Reference Documentation

Psychologists commonly distinguish between procedural and declarative knowledge. Procedural knowledge is knowledge of how to do things —for example, to ride a bicycle. Declarative knowledge is knowledge about things—for example, that bicycles are ridden with the hands on the handlebars, feet on the pedals, and that centrifugal force plays a role. Declarative knowledge is usually a prerequisite to procedural knowledge, but not vice versa. For example, it is important to know about bicycles before attempting to ride one, but such knowledge does not guarantee that one will be able to ride successfully. As we all know, one cannot learn to cook, box, or perform open-heart surgery simply by reading a book; the activity must be practiced until the skills are "in the hands." It is possible for procedural knowledge to exist in the absence of complete declarative knowledge; this is demonstrated by musicians who play by ear, politicians who write legislation on complex issues, and engineering managers who lack detailed technical knowledge.

For all these technicalities, the important point is that a program user must master more than facts to operate a computer program successfully. Facts form the foundation, but procedures tell how actually to do things. User documentation provides the necessary facts in the form of reference information—in reference manuals, guides, or quick reference cards. Procedural information, if provided, usually appears in tutorials or descriptions of procedures in a user's guide.

There is a problem. It is common for user documentation to cover facts adequately but to overlook procedural information. The reasons for this oversight are debatable but probably reduce to ignorance by authors of the importance of procedural information. It is quite understandable that such ignorance should exist. People who learn to do things often forget how they learned them and have difficulty explaining how to do them. (Have you ever seen a description of how to ride a bicycle, pilot an airplane, or play the piano?) Thus, it may seem to the skilled person that the essentials reduce to a set of easily explainable facts. Clearly, there is much more to the matter, and the procedures—however difficult they are to describe—need to be explained. Fortunately, the procedures for operating most computer programs can be laid out in simple step-by-step fashion and don't pose the difficulty of explanation of complex skills. Put another way, how to work *Microsoft Multiplan* isn't as hard to describe as how to play a Beethoven piano sonata.