4 *Scientific Writing*

4.1 Introduction

Scientific articles in scholarly journals, magazines, and conferences are a very special kind of technical writing. Senior, masters, or doctoral theses also fall into this category. These writings tend to have longer "life spans" as they are archived in digital libraries for search and retrieval by other scholars in perpetuity. The archival aspect of these kinds of writings differs from the technical writings that you are likely to produce at work. Some of your work writing will be archived by your company, customers, and perhaps your company's legal counsel, but such writings are not intended for public scrutiny.

Many characteristics of the scientific article (such as reference lists, discussion of precedents, etc.) are found in technical reports that you may be asked to write. For example, you might have to prepare some kind of scientific writing or technical report for a current or future college course.

You may never attempt to publish a scientific article. But here is a piece of unsolicited advice: Even if you do not plan to publish some kind of technical work, reconsider. Having a publication or two on your résumé sets you apart from those who do not have any publications. A published paper shows that you have the fortitude to see a significant project through to completion. The peer review process behind a published paper also validates your expertise in a more convincing way than a claim on your résumé. Any publications, speaking engagements, and extra activities you conduct help to "brand" you as a distinguished professional.

In this chapter, I discuss various kinds of scientific and technical writing. My hope is that you will better appreciate this kind of writing as a reader, and that you may one day consider publishing your own work.

4.2 Technical Reports

Technical reports can take many forms: an overview of some technology, a survey of candidate products for some application, or a forecast of a technological trend.

You can organize technical reports in many ways. Examples include a set of research questions, a survey of recent research, or a list of product innovations. Yet another way is to create a taxonomy of a field, potentially identifying market or research gaps. An example of this approach for real-time imaging is included in Section 4.5.1. Finally, you can use the format that your professor, client, employer, or industry requires.

I wrote a technical report on open-source software for the online journal *Computing Reviews* using the research question format [Laplante 2008] (© 2008, ACM, Inc. Included here with permission):[1]

> In 1983, Richard Stallman created a Unix-like operating system called GNU (a recursive acronym for "GNU is Not Unix") and released it under a license that provided certain rights for use and redistribution—an open-source license. Eight years later, a graduate student at the University of Helsinki, Linus Torvalds, created another Unix-like operating system, Linux, which he also made available for free. Both Linux and GNU are still widely available, and their evolution spurred the creation of many other open-source software (OSS) programs. By 1999, a prodigious open-source software developer, Eric Raymond, published his famous treatise, comparing the development of open-source software to the market conditions found in a bazaar, and describing the development of commercial software as a secret, almost religious experience. The process and culture created by Stallman, Torvalds, Raymond, and others formed the basis for the open-source software movement.[2]

There was more to the introductory material, of course, but the remainder of the technical report was organized simply as a discussion of lines of research in the field, namely:

1. Open-Source Adoption Decision-Making and Business Value Proposition
2. Legal Issues (Licensing and Intellectual Property)
3. Qualities of Open-Source Software
4. Open-Source Community Characteristics
5. Source Code Structure and Evolution
6. Tools for Enabling OSS and Applications
7. Philosophical and Ethical Issues

For each of these questions, I described the current state of research and listed a few relevant references. A comprehensive conclusion and the short glossary shown in Table 2.6 in Chapter 2 completed the technical report.

Technical reports are relatively easy to write because you don't have to "invent" anything. You simply have to do a good job discovering the current state of affairs and then organize that information in a logical way.

4.3 Tutorials

You might be asked to write user manuals, tutorials on new technologies, or a primer on the theoretical foundations of some technology with which you are involved. Alternatively, you might have the luxury of having an in-house or outside technical writer helping you. But even in this situation, experience in preparing explicative material is valuable. Tutorials typically look like excerpts from textbooks.

An example of a tutorial follows. This tutorial covers the Halting Problem, an important concept in computer and systems sciences.

> Here is a theoretical question that has important practical implications in real-time systems scheduling analysis, program verification, and the development of process monitors. "Can a computer program (called 'the Oracle') be written, which takes as its input another arbitrary computer program, *P*, and a set of inputs and determines if *P* will eventually "Halt" or "Does not Halt"? (See Figure 4.1.)

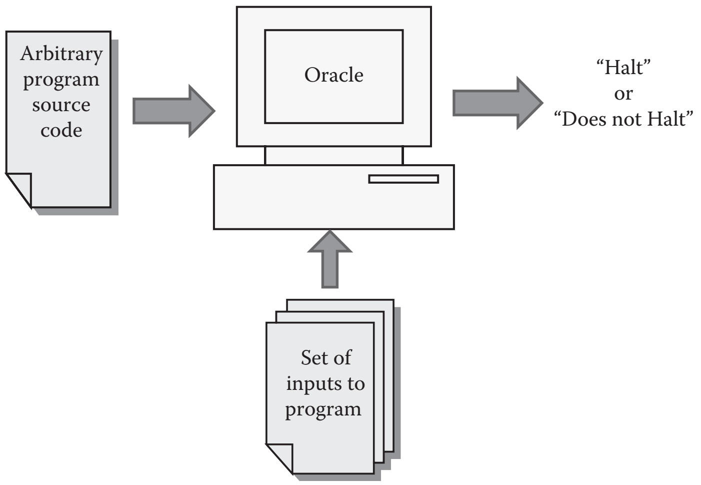


FIGURE 4.1
A graphical depiction of the Halting Problem.

This classical decision problem is called the “Halting Problem,” and it so happens that it is undecidable, that is, no such Oracle can be built. One way to demonstrate that the Halting Problem is undecidable is to use Cantor’s “diagonal argument,” which was first used to show that the real numbers are not countably infinite.

Suppose that we encode program *P* by concatenating its source code bytes. Such a strategy will yield a unique number (a very long one) for every computer program, whatever the programming language. The same encoding scheme can be used with each possible input set. Now, if we could build “the Oracle,” its behavior could be described by Table 4.1.

Table 4.1 represents the internal decision logic of the Oracle by depicting each possible program as a row, every possible set of inputs as a column, and the corresponding outputs from the Oracle. The “☺” symbol indicates that the program will halt on the input and the symbol ☹ indicates that the program will not halt on the corresponding input. For example, the table shows that for Program 1 with Input Set 1, the Oracle would decide “☺,” that is, Program 1 will halt on Input Set 1. Imagine that you have completed Table 4.1, that is, you have accounted for every program and input set in the Oracle’s logic (“☺” or “☹”).

You should see, however, that no matter how hard you try, a new program can always be created whose behavior is different from any other program already accounted for by the Oracle. This new program is created such that its output corresponds to the last row in Table 4.1. That is, if Program 1 outputs a “☺” for Input Set 1, then the new program outputs a “☹” for Input Set 1. If Program 2 outputs a “☺” for Input Set 2, then the new program outputs a “☹” for Input Set 2 and so on. If Program *n* outputs a “☹” for Input Set *n*, then the new program outputs a “☺” for that input set and so on down the diagonal of

TABLE 4.1

Cantor’s Diagonal Argument to Show That the Halting Problem Is Undecidable

	Input Set$_1$	**Input Set$_2$**	**…**	**Input Set$_n$**	**…**
Program$_1$	☺	☺	…	☹	…
Program$_2$	☹	☺	…	☹	…
.			…		…
.			…		…
.			…		…
Program$_n$	☺	☺	…	☹	…
.			…		…
.			…		…
.			…		…
A new program	☹	☹	…	☺	…

> the logic table to the very last program that the Oracle knows about. Since you can never complete the table the Oracle's logic is always incomplete, implying that the Halting Problem is undecidable.

Notice how in this tutorial I have combined narrative text with a figure and a table to illustrate a rather difficult concept. This approach is consistent with my preference for avoiding long runs of sentences.

Tutorials are difficult to write; you may want to describe and define everything because you fear that the reader may be a novice, or you may skip over important details that appear to you to be obvious. Have several readers review your tutorial writing, more than you would have review other kinds of technical writing; it will improve the presentation.

4.4 Opinion

Technical opinion papers are found in office communications or consulting when, for example, you are asked to endorse some new tool, technology, or methodology. Opinion or position papers are fun to write because you have license to use humor, anecdotes, metaphors, and other kinds of writing devices in addition to logic and precedent to support your position. I have written many opinion pieces and I enjoy writing them. I especially like having fun with the titles so that I can grab a prospective reader's attention.[3] Here are a few of the more amusing titles of opinion papers that I have published:

- "Another Ode to Paranoia"
- "The Joy of Spam"
- "The Burning Bag of Dung and Other Environmental Antipatterns"
- "Staying Clear of Boiling Frog Syndrome"

While you have great leeway in writing opinions, you need to be careful not to become careless. You can't make unsubstantiated claims; and even if you are being speculative, you must acknowledge that you are only giving your opinion. You cannot use hyperbole, and you must use humor with great caution—or not at all. I would not write something humorous to a client, even in an opinion piece.

Here is a deconstructed excerpt from a piece I wrote critiquing a practice used in Agile software development—the stand-up meeting. Stand-up meetings are also found, although not always by this name, in many other technical disciplines [Laplante 2003] (© 2003, ACM, Inc. Included here with permission).

> Stand-up meetings are an important component of the "whole team," which is one of the fundamental practices of extreme programming (XP). According to the Extreme Programming Web site, the stand-up meeting is one part of the rules and practices of extreme programming: Communication among the entire team is the purpose of the stand-up meeting. They should take place every morning in order to communicate problems, solutions, and promote team focus. The idea is that everyone stands up in a circle in order to avoid long discussions. It is more efficient to have one short meeting that everyone is required to attend than many meetings with a few developers each [1].

Note that in the article all claims are substantiated, although in this excerpt, reference [1] ["Daily Stand Up Meeting"] is not shown. Further on in the article, I try to point out negative aspects of stand-up meetings by way of comparison with other unhappy stand-up activities:

> I think there is something wrong with a meeting held standing up. Standing is inherently onerous. It is used for punishment in schools and in the military: "Stand in the corner," "Stand at attention," etc. Standing has an implicit element of authoritarianism and theory X control (which asserts that people will respond only to threats).

Then I offer a personal anecdote to make a point:

> I endured regular stand-up meetings for three years. What made the meetings most painful was my boss. His main reason for the stand-up meeting was not to increase efficiency or embrace XP as much as it was to shorten human interaction beyond anything directly related to the work product. This is the same boss who never took me out to lunch because he believed it was a waste of time.

Finally, I close with a bit of logical reasoning:

> For a methodology that emphasizes people over process, stand-up meetings seem contradictory: Don't structure meetings because informal communication is the best; hold highly structured meetings that discourage informal communication.
>
> I have described several objections to stand-up meetings, and although some of my alternatives can also be dismissed, my hope is that those who embrace XP—or consider embracing it—will understand that the stand-up meeting is not necessarily what it seems. I suspect that many team members consider this aspect of the approach the most onerous. Perhaps alternatives should be embraced as part of the XP culture. After all, one of the tenets of agile methodologies is to "embrace change."

This editorial uses a combination of comparison, personal experience, logic, and scholarly reference to build a case. I like using multiple techniques in my opinion pieces, but you may use only one or two.

4.5 Research Papers

4.5.1 Survey of the Field

You may be asked to conduct background research on a technical area and summarize your findings in a report. Your report may also include observations about gaps in prior research and future research trends. The research report differs from the technical report only in that the former is intended for experts to help map the way ahead for future research, while the latter is intended as a primer.

There are several ways to organize your findings. One way is as a dry summary of papers and their contents. Another way is to organize the papers into a historical context, showing trends or epochs in the evolution of the technology. As with the technical report, you can also organize your research report as a set of research questions. Yet another way is to use an existing taxonomy or create your own—a structured decomposition of the field.

For example, here is an excerpt from one such taxonomy-based approach for a real-time image processing research [Laplante 2002]. My goal in this paper was to review the set of publications in the journal *Real-Time Imaging* for the previous seven years and to do a gap analysis, that is, to identify where I thought more research was needed. The report begins:

> To understand the present, we must study the past. To see just how far has the sub-field of real-time imaging advanced, a keyword search was done on various research databases on the keywords "real-time" conjoined with "imaging" for 25 years before the founding of the *Real-Time Imaging* journal (1995) and for the years since its founding. Table 4.2 summarizes the results of these searches. [Laplante 2002]

I gave the findings of my background search in summary form, as shown in Table 4.2.

Rather than give a dry summary of every paper found (there were more than 100), I listed the titles in a very long reference list, and clustered those

TABLE 4.2
Search Results for Various Research Databases on the Key Words "Real-Time" and "Imaging"

Database	Hits in Publications before 1995	Hits in Publications 1995 until January 2002	Total Publications	% Increase <1995 to =2002
INSPEC	322	220	542	–31.68
IEEE Explore	196	349	545	78.06
Academic IDEAL	31	832	863	2583.87
American Institute of Physics	228	48	276	–78.95
Kluwer Online	0	10	10	∞
ACM Digital Libraries	0	323	323	∞
JSTOR	219	98	317	–55.25
Elsevier Science Direct	152	185	337	21.71
Springer	0	131	131	∞
Information Science Abstracts	0	5	5	∞
Wiley Interscience	1	77	78	7600.00
Total	1149	2278	3427	98.26

Source: From Laplante, P. A., *Real-Time Imaging*, 8(5), 413–425, 2002. With permission.

papers by research areas identified by authors in previous works. The space of image processing was thus organized accordingly:

Image compression
Remote control and sensing
Image enhancement and filtering
Advanced computer architecture
Computer vision
Optical measurement and inspection
Scene types
Models
Image data types
Processes

I then proposed a modification to the list based on my own analysis:

> The proposed changes [are] as follows. First, the classification "image compression" has been combined with "image data types" to form the area "image compression and data representation" to recognize the fact that in real-time imaging, the main issue for data representation is to reduce storage requirements and transmission times.

> Next, the classification "scene types" is combined with "models" and renamed to read "multimedia/virtual reality." Such a classification would include multimedia systems, virtual reality, scene modeling and real-time rendering related research.
>
> Next, the term "intelligence" is added to the area of "computer vision" to more clearly capture the various soft-computing techniques that are involved in this area.
>
> Then "computer vision" and "optical measurement and inspection" are combined because the only apparent distinction is that "computer vision" pertains to generalized schemes, while the other can be assigned to specific application domains, such as textile or agricultural inspection.
>
> Finally, the classification "processes" is split into "algorithms" and "software engineering issues." This is due to the belief that it is important for real-time imaging researchers and practitioners to begin to focus on best practices of software. Perhaps because many imaging engineers are not trained in software engineering, or because of pressures to complete the project, basic software engineering practices are often not followed, or followed poorly. This situation seems to exist in both industrial and academic R&D labs.

The analysis led to the following proposed taxonomy for real-time imaging research:

Image compression and data representation
Remote control and sensing
Image enhancement and filtering
Advanced computer architecture
Computer vision and intelligence
Multimedia/virtual reality
Algorithms
Software engineering

Then I analyzed the set of published papers that had appeared in *Real-Time Imaging* to show how the distribution was uneven across the areas proposed in the new taxonomy. These observations led to the penultimate conclusion suggesting new research:

> Relatively little research has appeared in image compression and data representation (8%) and in image enhancement and filtering (7%). Disappointingly, even less work has been published on remote control and sensing (<1%), and on software engineering for real-time imaging systems (1%).
>
> Therefore, more work in these areas should be encouraged.

The approach I took in the *Real-Time Imaging* survey paper was a rather straightforward one, and an approach that you can use in your own work. The steps are simple:

1. Assess the state of current research by organizing existing papers into a known taxonomy.
2. Analyze the set of organized papers to identify any gaps.
3. Propose new research to be conducted based on the gaps.

You can propose any adjustments to the known taxonomy as warranted.

4.5.2 Based on Survey Data

Survey-based research is often difficult to conduct because of the challenges in getting a suitable number of respondents. The mechanics of conducting such research are outside our scope here. Let us instead consider the organization of the written report that summarizes the findings that may result from survey research. Technical survey-type research is very similar to marketing research; if you want to write this kind of paper in your place of work, you would do well to consult with your marketing department for some advice. Marketing professionals are experts at this sort of research writing.

As an example of this type of writing, consider a survey on requirements engineering practices that I conducted with a colleague (portions reprinted, with permission, from Colin J. Neill and Phillip A. Laplante, "Requirements Engineering: The State of the Practice," *Software,* 20(6), 40–46, 2003, © 2003, IEEE).

The introduction to the paper describes how we organized our survey. After describing the need for the survey, we explain the administration process:

> We created a Web-based survey (www.personal.psu.edu/cjn6/survey.html) consisting of 22 questions (summarized in Table 1). We drew our survey participants from a database of prospective, current, and past graduate students of the Penn State Great Valley School of Graduate Professional Studies. We sent them an email invitation (and subsequent reminder) to visit our Web site.

You can also describe a survey sample population graphically, as we did in another paper based on the same survey data (see Figure 4.2).

Returning to the survey, we described the survey questions in tabular form (shown in Table 4.3).

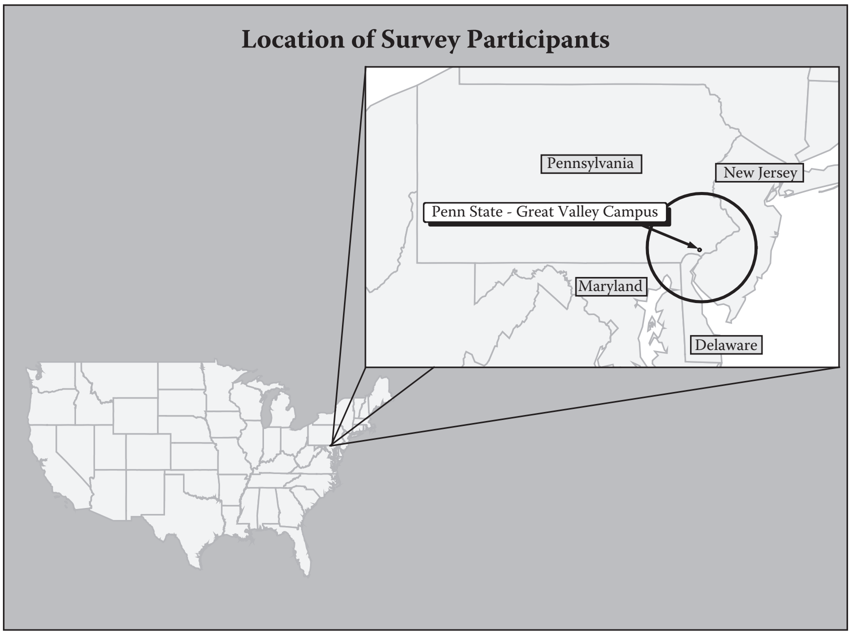

Location of Survey Participants
Pennsylvania
New Jersey
Penn State - Great Valley Campus
Maryland
Delaware

FIGURE 4.2
Penn State Great Valley School of Graduate Professional Studies location and service area. (From Laplante, P. A., Neill, C. J., and Jacobs, C., *Proc. 27th NASA/IEEE Software Engineering Workshop*, December 2002, pp. 121–128. With permission.)

The paper then describes an important component of any survey research, the response statistics:

> We collected survey data through March and April 2002, and although we received a few additional responses after this date, we didn't include them because the analysis had already commenced. So, of the 1,519 invited persons, we had 194 completed responses.

There are many ways to report survey results, but it is common to use bar and pie charts as in Figure 4.3. Here we reported on the survey results for the question "Which of the following development life cycles best describes the one you are using/did use?" The x-axis indicates the primary choices selected by survey respondents, and the y-axis indicates the percentage of respondents who selected that response to the question. The notation "n = 191" indicates that the number of respondents to the question was 191.

The main result of this survey—that more than 30% of respondents still used the Waterfall life cycle model—yielded several other papers and this

TABLE 4.3
Summary of Survey Questions

No.	Question
1	What type of business or organization are/were you employed by during this project?
2	Approximately how many software professionals are/were employed by your organization?
3	What is/was the approximate size of your organization's annual budget?
4	Which of the following application domains does/did this project apply to?
5	Which of the following development life cycles best describes the one you are using/did use?
6	Within the life cycle, do/did you do any prototyping?
7	If your answer is yes, how do/did you prototype?
8	How many full-time staff (IT) are/were involved in the project altogether?
9	How many full-time staff are/were involved in each phase of the project development?
10	What is/was the duration of the project (from inception to delivery)?
11	How is/was the project duration distributed among the following phases?
12	What techniques do/did you use for requirements elicitation?
13	Which of the following approaches are you using/did you use in analysis and modeling of the software requirements?
14	In what sort of notation is/was the requirements specification expressed?
15	Do/did you perform requirements inspections?
16	If your answer is yes, which technique do/did you use?
17	In your opinion, does your company do enough requirements engineering?
18	When you review/reviewed requirements, which of the following approaches do/did you employ?
19	Indicate with a number the size of requirements specification in terms of [the following].
20	The following statements are indicators for software quality and software productivity. Please rate these statements by clicking one box with the following scales.
21	Which of the following best describes your position while engaged in this project?
22	Over the last five years, how many software projects have you worked on?

Source: Neill, C. J. and Laplante, P. A., *Software,* 20(6), 40–46, 2003.

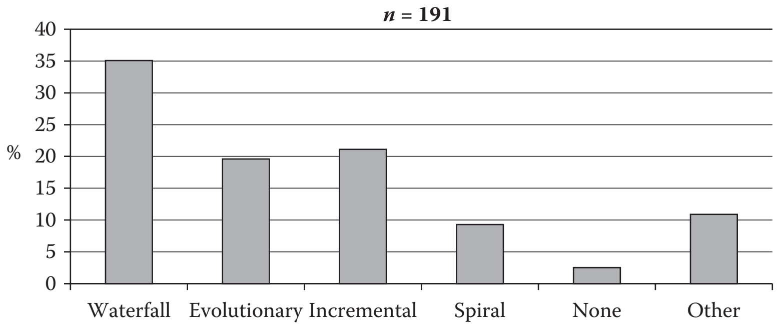


FIGURE 4.3
Reported life cycle model used. (From Neill, C. J. and Laplante, P. A., *Software,* 20(6), 40–46, 2003.)

result was reported in a prestigious international magazine [The Economist 2004]. Many other bar charts, such as Figure 4.3, appeared in the paper. The use of graphical elements, such as those just shown, is discussed further in Chapter 7.

4.5.3 Based on Experimentation

You may be asked to write a research report describing the results of an experiment or a set of experiments. This kind of technical writing is very straightforward and you can use the following format:

1. Describe the background and need for the experiment
2. Describe the methodology of the experiment
3. Report the results of the experiment
4. Interpret the results of the experiment
5. Offer recommendations and conclusions

The key to this kind of writing is to provide sufficient information to allow someone else to repeat the experiment. Readers may believe your results, but they will certainly lose faith if the results cannot be repeated. Even if no one chooses to attempt to repeat your experiment, providing ample background information increases confidence in the results.

To illustrate, I will describe an experiment conducted by one of my students and the subsequent technical report we published in a respected refereed magazine. In this case we wanted to explore the viability of Microsoft's C# language and .NET framework for use in real-time applications [Lutz and Laplante 2003].

After describing the various technologies involved in some detail, and some special concepts in real-time processing, the performance tests were described.

> We conducted two experiments comparing C#'s performance against C. All tests were built with version 55603-652-0000007-18846 of C++. NET, and we built and executed C#.NET on the.NET platform, version 1.0.3705. These versions correspond to.NET's first generally available, nonbeta version. All builds were optimized release builds. The tests ran on an 800-MHz Inspiron 8000, running Windows 2000 Professional, SP 1, with 523 Mbytes of physical memory.

The first test was the execution of 10 billion floating-point operations so we included a graph showing the performance results. We then described the second test, which dealt with memory management:

> We generated and released a linked list containing 5,000 nodes in both C and C# 24 times (48 total). Every other list saw an increase in node size. So for the first two sets of 5,000 nodes, each node contained a simple

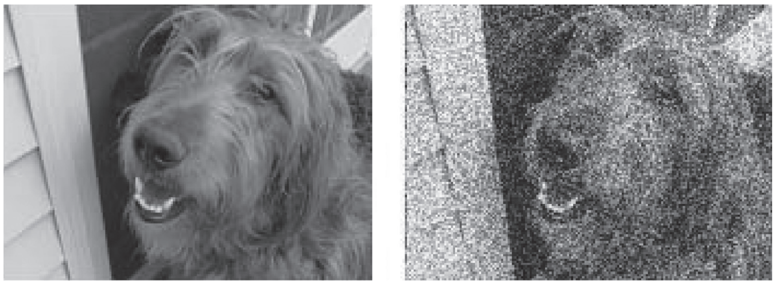
> numeric value and a string of length zero. For the second set of two lists, each node contained a simple numeric value again and a string of 2,500 bytes, and so on, increasing the string size by 2,500 bytes every other list iteration.

Note how we provide sufficient details of the experiment in order to allow interested parties to repeat the experiments. Not only are the computations explained, but also included are the characteristics of the computer on which the experiments were conducted.

Here is another example from a professional paper (thesis) written by one of my graduate students. The paper reports results of experiments with a new technique that I developed for dealing with various types of uncertainty in images [Rackovan 2005].[4] After some preliminaries, the paper describes the experiments:

> The image used for all of the experiments was the well-known "Lena" image (lena.jpg). All of the image filters were implemented with MATLAB and all executions were performed from the MATLAB command line. The images were converted to an array format, each containing a separate grayscale value ranging from 0 (black) to 256 (white). For ease of implementation, this image array was then transformed into a double floating point array ranging from 0 (black) to 1 (white), and then randomly contaminated with 3 different levels of uncertainty. These values of uncertainty were set to −3 (most uncertain), −2, and −1 (least uncertain). The contaminated image was then run through both versions of the mean filter, median filter, dilation filter, and erosion filter. Figure 4.4a shows the original image used in this investigation. Figure 4.4b shows the original image contaminated with uncertainty.

There is an interesting story behind the "Lena" images in Figure 4.4. This image has appeared in many books and hundreds of papers in imaging science journals since 1972. "Lena" is particularly useful to imaging scientists because it has both clearly defined and fuzzy edges and skin and hair tones



(a) (b)

FIGURE 4.4
The Lena test image before (a) and after (b) adding noise.

that present interesting challenges for certain algorithms. Perhaps another reason for the persistent use of the photo is that Lena is attractive—in fact, the Lena picture is a cropped version of a frontal nude that appeared in the November 12, 1972, *Playboy* magazine. The pervasive use of this picture in image processing papers for so many years has created two ethical dilemmas—one relating to the "objectification" of women, the other to copyright infringement—most authors fail to obtain permission to use the image.

In 1992, Playboy Enterprises challenged the editors of one publication, *Optical Engineering,* alleging copyright infringement, but left the matter unresolved [Hutchinson 2001]. Despite these issues, Lena still appears in imaging papers, usually without proper attribution. Very recently I reviewed a paper for an image-processing journal that had an unattributed Lena image. Of course, Lena also appears in this book.

At this point you must be confused because Figure 4.4a and 4.4b do not depict a woman. Figure 4.4a and 4.4b are actually photographs of my dog, Teddy (sadly, he passed away a few years ago). I requested permission from Playboy Enterprises to use the image of Lena, but after several requests, I received no response. So, I had to use a substitute—and Teddy gave me permission to use his likeness. If you want to see the real Lena picture, just Google it.

Getting back to the thesis, another feature is the representation of an algorithm in a programming language, in this case, pseudocode. Pseudocode is a generic name for any code syntax that resembles a programming language but is not intended to be compiled and executed. The idea is that if the reader understands C, Fortran, Java, or some other modern programming language, then they would get the general idea of the algorithm by inspecting the pseudocode. Here is the introductory text and the accompanying pseudocode:

> The algorithm used for contamination of the original image is shown in Algorithm 1. A random pixel array was generated using the MATLAB rand function. This function generated a 512 × 512 array of random numbers ranging from 0 to 1. This array was traversed and the original lena.jpg was given values based on the logic displayed in Figure 4.5.

```
if random_map(i,j) > 0.98
        Iunc(i,j)=-1;
elseif (random_map(i,j) > 0.96) & (random_map(i,j) <=0.98)
        Iunc(i,j)=-2;
elseif (random_map(i,j) > 0.94) & (random_map(i,j) <=0.96)
        Iunc(i,j)=-3;
else
        Iunc(i,j)=Iorig(i,j);
end
```

FIGURE 4.5
Algorithm 1 contamination of original lena.jpg.

Of the 262,144 pixels in the contaminated image, 5,220 (1.991%) had an uncertainty value of –1; 5,242 (2.000%) had a value of –2; and 5,231 (1.995%) had a value of –3. This contaminated image served as the standard input to all the enhancement filters used in this investigation.

The key point here is that when expressing algorithms, technical writers should use pseudocode because readers shouldn't have to learn a new programming language to understand your experiment or conclusions.

In his thesis, as is often the case with technical writings, my student generated many numeric results. These results are most easily summarized in a table, preceded by an introduction. Here is how he handled this situation:

> Based on the results from all of the filtering methods, it can be determined that the EFPS filtering operations perform better than Regular filtering operations when applied to an image randomly populated with uncertainty. More specifically, all of the EFPS filtering methods reduced the number of corrupt pixels in the resulting image while retaining the intended functionality of the specific filtering operation. The EFPS Mean filter proved to be the most effective, and successfully removed all uncertainty from the originally corrupted "Lena" image (Figure 4.4b). Conversely, all of the Regular filtering methods resulted in images containing more corrupt pixels as compared to the original image.
>
> Table 4.4 summarizes the performance of all of the Regular and EFPS filtering techniques. The numbers and percentages displayed correspond to the amount of uncertainty present in the resulting image before and after it was filtered:

TABLE 4.4

Summary of Filter Performance

	Uncertain Pixels: Pre-Filtering			Uncertain Pixels: Post-Filtering		
Filtering Technique	**–3**	**–2**	**–1**	**–3**	**–2**	**–1**
Regular Median (8-Neighborhood)	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	42892 (11.621%)	36654 (13.982%)	30864 (16.362%)
EFPS Median (8-Neighborhood)	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	0	3(0.001%)	20 (0.008%)
Regular Mean (8-Neighborhood)	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	31009 (11.829%)	37266 (14.216%)	42793 (16.324%)
EFPS Mean (8-Neighborhood)	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	0	0	0
Regular Morphological Filter	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	14163 (5.403%)	14769 (5.634%)	14984 (5.716%)
EFPS Morphological Filter	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	1(~0.0%)	14(0.005%)	34 (0.017%)
Regular Morphological Dilation	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	13738 (5.241%)	14848 (5.664%)	15369 (5.863%)
EFPS Morphological Dilation	5231 (1.991%)	5242 (2.00%)	5220 (1.995%)	0	18(0.007%)	44(0.017%)

Finally, he interprets the results:

> Many recommendations arise from the strong assumptions made in this investigation. All uncertainty generated randomly was given the values of −1, −2, and −3 (or 2, 3, and 4 for the morphological erosion filters). The ability to classify these varying levels of corruption remains undefined and may lead to extremely subjective classifications. Based on this prospect, the filtering algorithms presented in this investigation are useless without extensive pre-filtering characterization of the possible types of uncertainty residing within the corresponding image data. Numerous techniques have been adopted that identify various types of image contamination. An extension of this investigation would be to consolidate past works in assorted areas of uncertainty classification and implement these results as the pre-filtering procedure to all of the EFPS techniques, specifically the population of the fuzzy set used to define the varying levels of image contamination.

These excerpts from my student's thesis are great because they showcase several elements: graphics, tables, code snippets, and experimental data. The Lena story is a good teaching moment, highlighting the challenges in procuring copyright permissions and focusing on working around setbacks.

4.6 Reviews of Books, Papers, and Reports

4.6.1 Reviews

Reviews of technical books, research papers, and reports take a form similar to reports of nontechnical works. You may be asked to write such reviews for your employer, for a client, or for a class. Therefore, it is worth understanding how to write good reviews.

I have served as an area editor for ACM's *Computing Reviews*, which is the leading forum for reviews of books, conference papers, journal papers, and other scholarly writing on the computational sciences. In this capacity, and as a professor, I have reviewed hundreds of other peoples' reviews that range from exquisite to sophomoric and even worse. I have also written and published a few reviews myself. Based on these experiences, here is some advice on writing good reviews for various types of publications. Should you have to write a review for a technical paper or book, I have included a template in Appendix B.

4.6.2 Journal and Conference Paper Reviews

Journal and conference paper reviews are straightforward: You want to capture the essence of the work and its importance, identify any shortcomings, and then make a recommendation as to the type of reader who would be most interested in the paper. Here is a detailed outline of a review. The first and last elements are essential; the other elements can be used or omitted as appropriate.

1. Grab the reader's attention in the first sentence.
2. Set the context for the work being reviewed:
   a. Why is this work important?
   b. How does this work fit with respect to related work?
   c. Is there something important about the authors to note?
3. List the salient findings or results of the work.
4. Discuss any shortcomings of the work, and suggest possible improvements.
5. Recommend areas for future research related to the work.
6. Recommend the type of reader who would most benefit from the work (e.g., researcher, practitioner, novice, or expert).
7. Recommend further readings.
8. Give your final assessment of the work and any salient conclusions.

Here are some excerpts from a review that I wrote for a paper about software improvement [Laplante 2006] (© 2006, ACM, Inc. Included here with permission). First, I try to get the reader's attention:

> An old saying goes, "When you don't know where you are going, all roads will take you there." The authors would have you reinterpret that saying as follows: "If you know where you are going, regardless of how much time and money you waste, how many accidents you have, and how much you hate the ride, all that matters is that you become a better driver"; In short, this piece seems intended to salvage the reputations of those who led a disastrous software process improvement (SPI) initiative.

I then proceed to describe the experiment that was conducted involving an anonymous company and a particular process improvement framework called the Capability Maturity Model (CMM). I recount particular details of the company in question, and note that certain details were omitted. For example:

> Although employee surveys were used extensively during the initiative to solicit feedback, we don't know much about this feedback (though we are told there were "no major problems with the methodology or with reluctance by the developers in using it"). But it is hard to believe that the response was overwhelmingly positive. In any case, it seems that a serious flaw in this approach was that no appropriate buy-in or input was obtained from developers prior to the mandate—only after its pursuit was a fait accompli.

I wrap up by recounting the nine lessons that the authors believe were learned from the experiment: I note that these lessons are already well known. I then give a rather scathing critique of the research:

> [The conclusions sound] like the CIO's damage control statement to the board and shareholders. And, although the authors allude to the benefits that were achieved by the attainment of level two (in the areas of customer satisfaction, number of days of deviation from scheduled delivery date, and the percentage of deviation from the proposed budget), we are given no data. Indeed, from the way the narrative is written, it is unclear that any of these measures improved at all.

Finally, I note that too much information is missing to repeat this research and that the results seemed to have been spun to support the foregone conclusions.

> The authors conclude that "software process improvement is a journey, not a destination." Although this sentiment is correct, it shouldn't be used as an excuse for a failed initiative, particularly when the cost of learning such obvious lessons is so high. By choosing to act as apologists for the CIO of AAC, the authors miss the real lesson of this case.

Clearly, my opinion of this paper was not very high, and the review is a negative one. But the structure of a positive review would be the same.

4.6.3 Book Reviews

Good book reviews are very hard to write—much harder than reviews of papers. The best reviews focus on themes that tie the book together. This is not always easy; for example, academic texts are less likely to have deep underlying themes. But the job of the reviewer is to uncover a hidden structure that goes beyond what one can get by reading the table of contents.

The review should weave the narrative around the book's central theme to reveal its essential concepts. The review should not consist of a litany of chapter summaries. For example, consider this snippet of a deliberately poorly written review, for the book: Frederick Brooks' *Design of Design: Essays from a Computer Scientist,* Addison-Wesley Professional, 2010.

> Brooks' *Design of Design* is a master work from one of the leading computer scientists in history. This 448 page book is an interesting read from cover to cover. The book's 28 chapters are organized into six sections….
>
> The book starts with Section 1: Models of Designing, consisting of five Chapters. Brooks gives us his list of issues and paradigms for design…. In Section 2: Collaboration and Telecollaboration, Brooks tackles the issue of the modern distributed workplace….
>
> In Section 6: Trips Through Design Spaces, Brooks provides us with seven case studies of design from his own experience, and concludes

> with "Recommended Reading," which provides a nice list of references for the interested reader.
>
> This is one of the best books that I have ever read. Everyone should have "Design of Design" on their bookshelf.

Why is this review so bad? It starts frivolously: "master work" is a meaningless platitude. The second sentence is worse. The length of the book is unimportant (you could look that up on Amazon.com) and the sentence ends with a cliché. Next, we are led through a dry rendition of the book's contents, each paragraph fitting the same template ("In section x"). The review concludes with a predictable and boring conclusion. In my capacity, as one of three software engineering editors for *Computing Reviews*, I have unfortunately read far too many book reviews by experienced professionals, senior scientists, and professors that were just as poorly written.

To contrast, here are some excerpts from the real review that I published for this book in *Computing Reviews* [Laplante 2010] (© 2010, ACM, Inc. Included here by permission):

> Best known for his seminal work on software engineering and project management, Frederick Brooks expands here to ubiquitous design. Having led the development of the IBM 360 computer and its operating system, I suspect that many do not know about his pioneering work in virtual reality. Brooks draws upon both domains, and more pedestrian endeavors such as the renovation of his own home, to seek universal meaning in physical design and the design of software.

In this opening paragraph, I am trying to draw the reader's interest. Then I use the fact that the book has a theme around "spirals" to lead the discussion.

> Brooks likes spirals. There is the Boehm spiral model, which is mentioned in several of the essays, a picture of a spiraling bridge, and even a spiral staircase in the author's beach house.

I then highlight a few of the chapters' contents, but not all of them. I conclude the review with the following:

> This book is many things, but mostly it is a look into the mind of one of the greatest designers of the automated age. If I were to make a movie based on this book, it would mimic *Being John Malkovich,* where the protagonist gets into the head of a famous American actor. Being Frederick Brooks would be a frenetic movie—thrilling at times, distracting at times, and puzzling at times—but there would be so much to be learned. I suggest that every computer scientist and software engineer buy this book.

Note how, despite my warnings about being funny, you can sometimes work humor into a review.

4.6.4 Blind Reviews

If you are an expert in your field, or simply a typical reader of some magazine, journal, or other periodical, you may be asked to review, for potential publication, the work of another author. In this case you are acting as a very special kind of reviewer or "referee." Your identity is protected from the writer (this is called a "blind") so that you may be honest without fear of retribution, argument, or harassment from the authors. In some periodicals, the identities of the authors are hidden from the reviewer during the review process so that any personal biases will be mitigated. In these cases, the review is known as "double-blind" because both the reviewer and the authors are unknown to each other. Only the editor knows the identities.

The editor of the periodical will take into consideration the reviews of the referees (usually at least two and sometimes as many as six) and render a consensus judgment. The usual judgments can be "accept without change," "accept with minor changes," "revise the paper and resubmit it for more review," and "reject." Papers can also be returned without review if they are deemed to be "out of scope," that is, inconsistent with the stated purposes of the periodical.

As a referee, your goal is to fairly, but critically, review the paper for the purposes of improvement. In the review, you are giving advice to both the authors and the editor. You need to put your personal feelings aside and be as objective as you possibly can be. In the worst cases, however, where the work is weak or very badly done, you may have to recommend rejection of the paper.

Here is an example of a blind review that I wrote for a short and informal paper that discusses a technique used in software engineering (and other kinds of engineering) called the "design interview." The paper had been submitted for a special issue of a magazine with the theme of "tools."[5]

> This paper is a little off-base from the tools theme issue. The paper introduces a methodology, or really, a micro-methodology. I like what the author is proposing, but I don't think we can call his proposed methodology a tool.
>
> This paper introduces a micro-methodology, the design interview. It's not a methodology in the sense that it is a fine-grain technique that can be used at every stage of the software life cycle. In my mind, the design interview does more than elicit information about progress. It's a team-building mechanism. It has to have a positive effect on the organization, team and ultimately the project, if the manager is caring and positive.
>
> When talking about embraceable techniques, I wonder what the effects of the lightweight methodologies (like XP or Scrum) have on embraceability? That is, it is reported that programmers tend to like to work this way as opposed to the traditional command and control, waterfall paradigm. Maybe a discussion on this might be fruitful.

I clearly liked this paper and could only offer a short recommendation for improvement. I left the decision to the editor as to whether this paper was within the scope of the special issue theme.

Some editors like to have two sets of comments from reviewers: one set for the authors and then a confidential recommendation to the editor. This approach is particularly convenient when the review has some negative elements. Here is an example of a review for a paper on project management that I wrote for another magazine. First, I provide comments to the author:

> I like this paper, and it addresses an important issue, but I found it somewhat lacking in specifics. In general, I think my main criticism is that it is a bit too abstract—while I understand the need to protect proprietary data, I think some more quantitative information is necessary. For example,
>
> What are some industry standards for "repeat business" and "strong referral" (mentioned on page 2). Then, perhaps you can talk about a percentage increase in these figures without giving away confidential information.
>
> The same goes for the Pursuits and Assessments and Implementation factors on page 3 (e.g., Win rate, project margins, client satisfaction, and timely payment). Talking about these in the abstract is less helpful than getting real numbers—even if they are simply industry standard numbers.
>
> On page 5, are "red projects" good or bad projects? In that same paragraph you mention a timesheet, a snippet example of one might be helpful.
>
> On page 6, the "Top 4–7 risks are examined in detail." It might be nice to have a table with a sampling of some of these.
>
> On page 6/7, "best practices are harvested…)," a short list of these might be nice.
>
> On page 8, maybe a few excerpts of sample audit responses would be helpful.
>
> On page 10, under "Did it Work?" I realize that the repeat business% is confidential, but perhaps the improvement% of repeat business would be nice. If this approach, e.g., led to a 10% increase in repeat business, companies would be scrambling to implement this!

Notice how I try to provide constructive criticism without attacking the author. Next, I give some advice:

> I don't know if the author got trapped by the word count limitations, but I think that more empirical data would give the paper much more credibility.

In the comments to the authors, I include very fine details as well as give my overall impression of the work. As a reviewer, you are expected to give both macro-level guidance and identify typographical errors, errors in equations, and errors in concept, as well as identify missing information.

4.6.5 Vignette: Scientific Proposal

To conclude the chapter, here's an excerpt from a proposal for a small grant project that I applied for many years ago. Proposals are discussed further in Chapter 6, but this vignette provides another example of scientific writing:

> A system Test Plan will be written that will incorporate hardware, software and integration testing. Hardware testing will cover the entire development life cycle including validation of system requirements using Requirements Reviews, model checking and consistency checking where appropriate; design validation using Design Reviews and simulations; hardware testing using best practice and final system integration testing via long term operational performance under load via fault injection.
>
> Typically, Field Programmable Gate Array (FPGA) testing consists of successively configuring the FPGE using a set of configuration inputs and then applying a test sequence for all configurations using the appropriate operational limit. After manufacturing, the complete FPGA will be tested independently of the application and again with the application installed using pre-defined test scenarios, which will include injected faults.
>
> The detectability of hardware faults in FPGAs depends on the circuit configuration. A given fault may be redundant (undetectable) in some configurations, while non-redundant in other configurations. Therefore, design for test requires that an appropriate configuration be identified, which will minimize the possibility of undetectable faults.
>
> Finally, built-in software testing specifications for the hardware will be generated so that ongoing software testing of the hardware subsystem can be performed.

The project was not funded, but the excerpt provides a nice example of scientific writing for discussion or editing (it can be improved).

4.7 Exercises

4.1 Write a short (no more than two pages) tutorial on your favorite hobby.

4.2 Write a product review for an electronic device of your choosing. The review can be similar in form to those found on Amazon.