Practical Guidelines for Documenting Internally-Developed Software Applications

Ron Bass

South Carolina Department of Public Safety

February 2, 2009

Problem Statement

The South Carolina Department of Public Safety (SCDPS), like most other medium-to-large State agencies, has a staff of analysts and programmers that develop and maintain a number of software programs that support the basic data operations of the Agency. In SCDPS' case (and I would guess in the case of most agencies), most of these internally-developed applications have very little supporting documentation. What documentation that does exist suffers from one or more of the following conditions: 1) it is not readily available to anyone other than its author, 2) the content and the layout of the information is inconsistent between documents (even between documents created by the same person), 3) it is misaligned to its audience, 4) it is redundant, 5) it is stored in an inappropriate media for its audience, and/or 6) the information that is contained within it is sparse, incomplete, and/or out-of-date.

In an application development environment, quality documentation is critical. It is necessary for timely and accurate customer support, knowledge transfer among staff and users, knowledge preservation, and, in large part, the overall quality, effectiveness, and sustainability of the application itself. However, despite an academic acknowledgement of the value of documentation or an intuitive sense that documentation is simply a good idea, in practice, documentation rarely rates more than an afterthought in the software development lifecycle. Consequently, we *achieve* the results mentioned earlier.

In the enthusiasm to improve the existing documentation efforts, we must guard against "documenting for documenting sake"[1]. The mere existence of documentation does not guarantee its worth. It has to be valued by its audience. Value, in terms of documentation, is best assessed by its actual usage. If documentation is created but never referenced, it is probably worse than not documenting at all. Not only do you have the wasted effort of producing the documents, but you have also reinforced the TAGRI (They Aren't Gonna Read It) principle[2] in the minds of your developers.

Similarly, sheer volume does not ensure a document's significance. In fact, there are arguments that voluminous documentation actually deters use[3]. The value of documentation to the user is gained, instead, through its pertinence, availability, presentation, and accuracy.

Ultimately, "poor" documentation (which includes non-existent documentation as well as documentation that is not referenced) increases the Total Cost of Ownership of a software application. Consequently, by improving our documentation practices and our documentation products, we will become more efficient in the delivery and support of our software programs. This is critical given the ever-increasing application development demands of the Agency.

Purpose

The purpose of this research effort is to identify the best practices, conventions, standards, procedures, tools, and delivery methods for adequately and efficiently documenting internally-developed software applications. Identifying this information upfront, prior to initiating a full-fledged documentation overhaul, will hopefully allow me and my staff to avoid wasting time on missteps and false starts which could further disillusion their opinion of documentation.

Scope

This research effort will focus only on the documentation that is used by the application developers to describe and detail the software application itself. It is not concerned with end-user instructional documentation* or project management documentation†.

---

* There is a whole body of work that concentrates on developing end-user documentation known as the Minimalist Principle. It is predicated on the work of IBM's John Carroll who wrote "The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill" in 1990.

† Similarly, the Project Management Book of Knowledge (PMBOK) covers the entire project management discipline.

Data Collection Methodology

One of my techniques was to locate books and articles on the most current trends in software development documentation. I used a combination of Internet searches and traditional printed material.

As mentioned earlier, the value of documentation is best assessed by its actual use. Accordingly, another technique that I employed was staff observation. It is my belief that if you can find the documents that the developers have created on their own (without being told), you will have a good indicator as to what they deem is important (and what they are willing to put effort into). Similarly, if you identify documents that the developers are regularly maintaining (again, of their own free will) or are regularly referencing, you will have a sense of what they value. In contrast, if you find documents that are out-of-date or rarely opened, you *may* have a feel for information that is considered not useful.[‡]

Data Analysis

Of the traditional research that I have conducted, the approach that most closely addresses my documentation concerns is a concept known as Agile Documentation. This concept owes its name and its philosophy to Agile Software Development.

Agile Software Development is a set of methodologies that are designed for the rapid delivery of high-quality software. Agile Software Development distances itself from traditional programming processes by developing software in small increments with minimal planning rather than long-term planning. These methodologies contend that the software can be developed faster and with a higher level of customer satisfaction by promoting development iterations and constant customer collaboration.

[‡] In some cases, a document may contain useful information, but there may be other issues such as poor organization that make it difficult to use and, thus, avoided.

Similarly, the Agile Documentation also diverges significantly from traditional documentation processes. The Agile Documentation concept promotes the following principles:

- Documentation should focus on communication.
- Documentation should be lightweight or "just enough".
- To be truly useful, documentation must be accurate, up-to-date, highly readable, concise, and well-structured.
- Documentation should be developed "just in time".[4]

The *Focus on communication* principle concerns itself with who the document is being created for, what will they do with it, and what do they require from it. Without this information, there is a tendency to "over" document, creating documentation that is either not referenced or whose sheer volume is an impediment to the intended audience. Consequently, the Agile Documentation approach advocates clearly defining the documentation audience and involving them in its creation.[5]

Equally important to the *Focus on communication* principle is the emphasis on transferring understanding to the intended audience. In other words, the goal should not be to create a requirements document (for example) but rather the goal should be to ensure that the readers clearly and unambiguously understand the requirements so that they can be fully implemented.[6] While this may at first seem like a distinction of semantics, the mere existence of a document does not ensure the latter goal. Instead, the latter goal of understanding is only achieved by collaborating with the audience.

The principle of *"Just Enough" documentation* finds its footing in the truism that comprehensive documentation does not necessarily ensure project success. While traditional processes define which documents need to be created, they do not guarantee that the documents will be read, understood, or even acted upon[7].

The Agile Documentation approach, in contrast, attempts to address these weaknesses by stressing quality over quantity.

Although the principle of *"Just Enough" documentation* implies less documentation (in terms of page count), it does not mean that less time is required to produce these documents. Instead, the effort that is saved by producing less documentation is often re-directed and spent on ensuring that the documents that are created are on-point and sufficient to the task at hand.[8] This premise of the Agile documentation approach falls into the widely-accepted speechwriting paradigm of "if I had had more time, I could have made it shorter". The idea is not to reduce the effort spent documenting but rather to improve the results of the documentation effort.

Naturally, the concept of quality is also important to the principle of *Useful documentation*. However, in addition to the obvious need for accuracy and currency of information, other factors that can impact a document's usefulness are its level of overlap between other documents and where it is ultimately housed.

To minimize the risk of outdated or inconsistent documentation, you should strive to define information in one place and one place only.[9] Having data that overlaps in several documents is akin to creating multiple links on web pages. While it provides the user with the convenience of a variety of retrieval methods, it introduces a maintenance concern which often outweighs the original user benefit.

Putting documentation in the most appropriate place and in the most appropriate medium is important for subsequent ease of access. This particular documentation decision will vary depending on the intended audience. For example, if the intended audience only includes the application programmers then the most appropriate place to document a design decision might be in the

source code itself. If the intended audience includes application programmers and management, then the most appropriate place for the design decisions might be a separate Word document.[10]

The *"Just in time"* principle states that documentation should be delayed as long as possible and created only just before it is needed. The idea is based on the premise that you should wait until what you are attempting to document has stabilized. For example, a system overview is best written right before a product's release because, by that time, you know exactly what you've actually built and there is little likelihood that it will change.[11] While documentation updates are inevitable, this principle attempts to minimize the update effort.

Obviously, there is a balance that must be struck between postponing documentation and providing updated information to customers and team members throughout the development lifecycle.[12] Supplemental documents such as meeting minutes may have to serve as the repository for project information while the technical documentation is completed.

In addition to the guiding principles, below is a list of other interesting concepts related to the Agile Documentation method that have distinct utility in any documentation effort.

- Include design decisions and their supporting rationale in your documentation. Similarly, include the alternatives that were considered but not chosen.[13] The one bit of information that is usually missing from documentation is "why" something was done a particular way. This information can be very useful when reviewing an application years after it has been developed and some or all of the original developers are no longer on staff.
- Templates should be used to promote a consistent and readable layout among the numerous authors who may be contributing to the Agency's documentation effort. In addition, templates allow the authors to focus on their forte, which is the document's content.[14] However, be cognizant that

not all templates are applicable in all cases. Accordingly, be careful that the use of templates does not evolve into a bureaucratic, prescriptive process.[15]

- There is no definitive template for documenting software applications. While certain types of information are recommended to be documented, there is no list of prescribed data elements that fit every situation or every environment.
- Introduce some level of file organization for the resulting documents that is consistent and serviceable. The reason for a file structure is to 1) allow the intended audience to easily locate a document and, 2) prevent redundancy or version-control issues between documents.[16]
- To the degree possible, use source code comments for documenting the application code. Because the source code comments are co-located with the actual application code, the application developers do not have to refer to a separate document. Similarly, being in such close proximity to each other, it is more likely that the application developers will maintain the code comments as functional changes occur to the application code.[17]
- Documents other than the source code should be used for documenting system overviews, requirements, design, and architecture.[18]
- Within the source code, use variable names that are descriptive. Avoid generic variable names such as Variable1. Instead, variable names, such as BirthDate, are far more descript and are often "self-documenting". Thus, requiring little or no additional source code comments.[19]
- When choosing a storage medium, consider the document's typical usage. The rule of thumb is that a printed medium is good for reading while an online medium is good for searching.[20]

In an effort to establish a baseline for my own section's current level of documentation AND to determine what types of document my employees are generating "own their own", I requested samples of their source code and "any

other documents that they had created related to performing their jobs". Below is a summary of those findings.

Almost predictably, all three staff members have introduced some level of comments into their source code. The commonality that I observed between the staff is that 1) their comments are used primarily to describe the various functions within the code, 2) their comments are relatively terse, and 3) the variable names that they have created are descriptive and, thus, have no associated comments. Given the sparse use of comments, the underlying assumption is that a person must have a working knowledge of the programming language to effectively troubleshoot or modify the existing applications.

The other types of documents that my staff members have generated include the following:

- procedural / instructional documents
- recurring maintenance documents
- user acceptance forms
- system description documents

What it interesting about this list is that some of the documentation is actually intended for people other than the developers themselves. At first glance, this occurrence appears to be in direct contrast with the theory that everyone will act in their own self-interest unless otherwise instructed. However, after talking with the application development staff, I learned that the procedural / instructional documents are created so that the developers do not have to be directly involved in the resolution of common application issues. Thus, the developers ultimately benefit from creating this type of documentation by deferring some of their more routine tasks.

As another part of my observation process, I also considered the types of questions that the application development team often pose. These questions

are good indicators of what type of information (ergo, documentation) is missing in the environment. The most common questions are "What is the purpose of a particular application?" and "What does a particular database field mean?". These two questions address concerns at two different levels of granularity. The "application" question is concerned with high-level, overview information. The "database field" question is concerned with low-level, detailed information. Both questions represent the different strata of documentation that is necessary to adequately support the needs of the application developers.

Implementation Plan

Based on my research and observations, I have identified four (4) core document types that need to exist. The first is an application overview document. This overview document will provide information about the purpose, scope, and background of an application. It will also include detailed information such as the application's file location, its associated database, and the primary and secondary programmers who are responsible for the application. Given that this document can be used by the application developers as well as other IT staff members, it needs to be created in a document separate from the source code. Similarly, given that it potentially could be viewed and queried in a variety of ways, it needs to be created as a database. NOTE: A database inherently introduces a level of access among its users that addresses the issues of availability and version control that are typically problematic for Word or Excel documents.

The second document that is needed is a data dictionary. The data dictionary will provide attribute, business logic, and relationship information about each of the data elements that comprise an application. For the same reasons given for the application overview document, the data dictionary needs to be created as a database separate from the source code.

In conjunction with the introduction of the data dictionary, a naming convention for all fields in all applications needs to be established. In our current environment, database fields are not named consistently across applications. This is proving problematic in our efforts to consolidate and integrate applications.

The third document is the source code. The source code, along with containing the actual programming code, provides variable definitions and descriptive information about the logic flow and the various functions performed by the programming code. Given that the information within the source code document is pertinent only to the application developers, a separate document will not be necessary.

In conjunction with the source code, a set of standards and conventions needs to be established. These standards and conventions will ensure that each source code document is consistent and contains at least the minimum level of information to be usable by staff programmers other than the original author.

The fourth document type would be for the procedural / instructional documents, the recurring maintenance documents, and the user acceptance forms. These documents will contain how-to information related either to global processes or to specific applications. These documents will be separate from the source code and will be developed as Word documents. Initially, these documents will be stored in folders based on an established folder hierarchy. Ultimately, these documents would be best presented as hyperlinks on a web page so that their content can be easily searched.

It is anticipated that the most significant development time will be associated with creating the database formats for the application overview document and the data dictionary. The primary obstacle will be that the database formats will

require actual application development effort and will need to be coordinated with other on-going development efforts to which my staff is currently assigned.

It is also anticipated that developing the standards and conventions for the data dictionary and the source code comments will require additional time to solicit staff input and perform research. I envision that the standards and conventions will initially be introduced as broad guidelines and will evolve over time with additional details.

Evaluation Method

Since the purpose of this effort was solely to perform research, the implementation of most of the findings will not occur until after this research paper is submitted. To date, only one of the document types (the application overview document) has been implemented.

The application overview document was developed as a browser-based system with a supporting SQL Server database. The application is accessed using Internet Explorer and the interface has the look-and-feel of a website. Currently, the application overview document is being used primarily by the application development team and by IT management.[§] Given its broad appeal, the document is also being used to document third-party software that we have purchased and deployed within the Agency. We are still in the process of populating the database with software titles. Of the records that have been entered, only a small fraction have been fully documented.

I propose to evaluate the long-term success of the application overview document (and the other document types) in much the same way that I approached the research. I plan to observe the natural behaviors of the programmers and the other users of the documents. I will review the file dates

[§] Appendix A has a list of the data elements that are collected and stored in the Application Overview Document.

and/or transaction logs to see what documents are being updated and viewed. If it appears that the documents are falling into disuse, the staff and I will look for any contributing circumstances, and, if need be, re-assess the value of producing the document.

Summary and Recommendations

Agile Documentation is a dynamic, evolving, and iterative process. Accordingly, we must be willing to continuously evaluate and improve our documentation in order to keep it viable, useful, and valuable within our specific environments. While the model does not offer definitive, static documents, its premise is based on practicality. It strives to produce sufficient documentation while minimizing unnecessary effort.

Based on the research and the degree of success / user acceptance that we have had with the application overview document, I am encouraged that this is the correct approach to achieving functional maturity within our software development processes.

Similarly, I fully envision that we will deploy this same methodology when defining the documentation needed in our other information technology disciplines such as server administration, network administration, and web development.

Endnotes

[1] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 3.

[2] Scott W. Ambler, http://www.agilemodeling.com/essays/tagri.htm, p. 1.

[3] Scott W. Ambler, http://www.agilemodeling.com/essays/agiledocumentation.htm, p.9.

[4] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. xii.

[5] Scott W. Ambler, http://www.agilemodeling.com/essays/tagri.htm, p. 2.

[6] Scott W. Ambler, http://www.ddj.com/architect/197003363, p. 1.

[7] Scott W. Ambler, http://www.ddj.com/architect/197003363, p. 1.

[8] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 4.

[9] Scott W. Ambler, http://www.ddj.com/architect/184415786?cid=Ambysoft, p. 1.

[10] Scott W. Ambler, http://www.ddj.com/architect/184415786?cid=Ambysoft, p. 1.

[11] Scott W. Ambler, http://www.ddj.com/architect/184415786?cid=Ambysoft, p. 1.

[12] Scott W. Ambler, http://www.agilemodeling.com/essays/agileDocumentation.htm, p. 8.

[13] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 20, p. 39.

[14] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 97.

[15] Scott W. Ambler, http://www.ddj.com/architect/197003363, p. 2.

[16] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 120.

[17] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 126.

[18] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 127.

[19] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 127.

[20] Andreas Ruping, Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects, p. 129.