the Availability Digest

www.availabilitydigest.com
@availabilitydig

Software Documentation

February 2016

I'll start coding.
You go find out what the customer wants.

With those words, the software for many legacy applications began decades ago. Little wonder that many legacy systems today are unmaintainable.

Back in the 1970s through the 1990s, I ran a software development company, The Sombers Group. We developed real-time, mission-critical software for many large companies including The New York Stock Exchange, Time, McGraw Hill, General Electric, and the Chicago Transit Authority, among many others.

Back then, real-time systems were by and large programmed in assembly language to get the maximum performance out of the underlying hardware. Unlike today's languages such as Java and C#, these languages were hardly self-documenting. Therefore, a major part of our effort was in software documentation. In fact, it was our practice to thoroughly document the software *before* we began coding. We than coded according to the documentation, which could be hundreds of pages long for a major project.

We supported many of these systems for decades. In doing so, we kept the documentation up-to-date. There was never a maintainability problem because of a lack of knowledge of how the software was organized. We did not face the legacy software problem that so many older systems face today, such as in the banking and finance industries.[1]

I can remember well-publicized programming contests in which programmers were given a task to complete. The programmer that finished the fastest won the contest. Never mind the maintainability of the result. Never mind the level of the documentation. If the code worked, that was all that mattered.

And that was the mindset of many software projects of the time. Major systems were built without documentation. The languages that were used were not self-documenting or were marginally human-readable such as Cobol, PL/1, and Fortran. The developers moved on, retired, passed away. The skill sets to deal with the older languages became scarce. Old legacy applications became unmaintainable.

Things have changed. With self-documenting languages such as Java and with advanced programming techniques, the need for software documentation has taken a different tack. Today, there are two common software development models – Waterfall and Agile. Both take a different approach to documentation.

[1] RBS – A Poster Child for Outages, *Availability Digest*; January 2016.
http://www.availabilitydigest.com/public_articles/1101/rbs.pdf

© 2016 Sombers Associates, Inc., and W. H. Highleyman
www.availabilitydigest.com
For discussion, contact editor@availabilitydigest.com

The Waterfall Model

The Waterfall model is really a throwback to how we handled documentation decades ago. It is a sequential design process flowing through several formal phases, from requirements specification to the design of the software, its implementation (coding), testing, and maintenance. The Waterfall model requires that a new phase be entered only after the previous phase has been completed and verified. In the 1980s, the Waterfall model was the standard used by the U.S. Department of Defense to work with software contractors.

The Waterfall model places emphasis on documentation such as the requirements documentation, design documentation, and test procedures. In a typical Waterfall project, about a third of the time is spent on the initial requirements and design documentation, about a third of the time is spent on coding, and about a third of the time is spent on testing and implementation.

One argument for the Waterfall model is that it places emphasis on documentation. In less thoroughly designed and documented methodologies, knowledge may be lost if team members leave before the project is completed; and it may be difficult for the project to recover from such a loss. If a complete design document exists, new team members can familiarize themselves with the project via the documentation.

It is also noted that time spent in the early phases of the software development cycle can reduce costs at later stages. For example, studies have shown that a problem found in the early stages of the project, such as during the creation of the requirements specification, is cheaper to fix than the same problem found later during the testing process, perhaps by a factor of 50 to 200.

Critics of the Waterfall model argue that the potential users of the software product may not know exactly what their requirements are before they see working software. This leads to changes in requirements, requiring redesign, redevelopment, retesting, and increased costs. It is better to be flexible during the project and to deliver useful results in small increments that can be vetted by the end users. In this way, changes can be made quickly and efficiently.

The U.S. Department of Defense now has a stated preference against Waterfall-type methodologies. Rather, it encourages *iterative and incremental development*, such as the Agile software development methodology described next.

Agile Software Development

Compared to traditional software engineering such as the Waterfall method, Agile software development targets complex projects with dynamic characteristics, in which accurate estimates, stable plans, and predictions are often hard to get in the early stages. Big up-front specifications and designs would probably be wrong, cause a lot of wasted time, and would not be economically practical.

With Agile software development, requirements and solutions evolve through collaboration between cross-functional teams including management, marketing, the end users, the developers, and the maintainers. It promotes adaptive planning, evolutionary development, early incremental delivery, continuous improvement, and rapid and flexible response to change. Agile software development minimizes the need to prepare a priori documentation.

Proponents of Agile software development argue that there are often better ways to achieve design goals than by writing static documentation. Documentation should be “just good enough.” Too much documentation can be a waste of developers’ time and is subject to rapid change, which often is not incorporated into the documentation. Therefore, the documentation rapidly becomes out of sync with the code. However, too little documentation may cause problems for later maintenance, user manuals, and marketing communications.

© 2016 Sombers Associates, Inc., and W. H. Highleyman
www.availabilitydigest.com
For discussion, contact editor@availabilitydigest.com

Therefore, a main goal of Agile software development is to reduce the amount of documentation created during a project and focus on the actual code. This is in contrast to Waterfall models where the process is often highly controlled, and minor changes to the system require significant revision of supporting documentation. One of the key features of Agile development is that it is an iterative process. When done correctly, deliveries of small increments of the project occur frequently (weeks rather than months or years) and provide immediate feedback for correction. Agile methods are focused on quick responses to change and continuous development. Working software is more useful to clients than documentation for managing the software development project.

The Agile development method breaks tasks into small increments with minimal planning. Iterations are short time frames that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions: planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration, a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. Working software is the primary measure of progress.

One unique method used by Agile software development is the use of *pair programming*. This is a technique in which two programmers work as a pair together on one workstation. One programmer writes the code while the other reviews each line of code as it is typed in. The two programmers switch roles frequently. Pair programming results in a small amount of additional effort to create the code, but the incidence of coding errors is remarkably reduced.

Agile software development covers many of the purported advantages of the Waterfall model. For instance, pair programming means that knowledge will not be lost if a team member leaves the project. Furthermore, the delivery of the software in frequent incremental stages allows errors to be detected early in the project and corrected with little expense

Documentation

There is a massive difference between the Waterfall model and the Agile method in terms of the amount of documentation that is developed. However, the documents can take on the same form even though one set (Waterfall model) is voluminous and the other set (Agile method) is minimal.

Technical Documentation

The first step in writing technical documentation is to determine what information needs to be included. This documentation will be used by the designers of the interfaces, by the developers of the code, by the database administrators building the project database, and by the testers who verify that the software works as intended. This information will probably include:

- The data to be maintained in the database.
- A description of each function.
- The overall program structure. For a disk-based application, this may include the individual program modules. For a Web application, it may mean describing which pages use which backend files or tables.

Specify how much documentation should be within the program code and how much should be in a separate document. For the Waterfall model, most of the documentation will be in separate documents (though it may be included in the code as well). For the Agile method, most of the documentation will be in the code, with perhaps a general guideline in a separate document. The modern programming languages used by the Agile method, such as Java, C++, C#, and PHP, have self-documentation facilities that should be used.

The technical documentation should describe the purpose of the functions, subroutines, variables, and constants used in the program. It should provide a mapping of which functions call which functions.

© 2016 Sombers Associates, Inc., and W. H. Highleyman
www.availabilitydigest.com
For discussion, contact editor@availabilitydigest.com

The documentation should be indexed or searchable so appropriate functions can be rapidly accessed. For legacy systems that have outdated technical documentation, there are several tools available to help in recreating up-to-date documentation, such as those from CGI (www.cgi.com).

User Documentation

The primary reason for providing user documentation is to help users understand how to use the application. In addition, it can serve as a valuable resource for marketing communications.

User documentation should not assume any familiarity with computer concepts on the part of the users. Some users such as system administrators will be highly knowledgeable. Other users, such as data-entry clerks, will most likely only know the application and how to enter data.

There are several formats that can be used for user documentation. A reference manual explains the individual data-entry features of an application, such as buttons, tabs, fields, and dialog boxes. This format can be useful to support context-sensitive help files that display a relevant topic whenever the user clicks the help button on a particular screen or hovers over a particular field or button.

A user guide explains how to use the application for a given task. User guides often take the form of tutorials, with a summary of the tasks to be performed in the introduction and instructions given in numbered steps.

User documentation can take many forms, including printed manuals, online manuals, or help files. Help files and online manuals should be indexed and keyword-searchable to allow users to quickly find the information they are looking for.

The creation of user manuals is an area of specialty for those of us at the Availability Digest. We have created user manuals for companies such as the New York Stock Exchange, McGraw-Hill, and the Chicago Transit Authority. Please contact us if you have a need for user manuals.

Summary

The role of software documentation has changed a great deal over the last several decades. Moving from the extensive upfront documentation of the Waterfall model to the minimal documentation of the Agile method, software documentation in now more of a matter of including it in the code than it is of writing large documents that rapidly age and become useless.

The one area that has not changed is user documentation. Though user documentation is more likely to be an online function than it was years ago, it is still needed in all of its detail to guide users through the use of the application.

Acknowledgements

Information for this article was taken from the following sources:

How to Write Software Documentation, w*ikiHow*.
Waterfall model, *Wikipedia.*
Agile software development, *Wikipedia*.
Pair programming, *Wikipedia*.

© 2016 Sombers Associates, Inc., and W. H. Highleyman
www.availabilitydigest.com
For discussion, contact editor@availabilitydigest.com