Agile Methods and Software Documentation

by Dr. David F. Rico, PMP, CSM

The subject of Agile Methods and software documentation continues to be a sore point and center of much controversy. This is primarily driven by the philosophy that Agile Methods are starkly different from Traditional methods. And, indeed Agile Methods are different from Traditional Methods. We know Agile Methods are different from Traditional Methods, because the creators of the Agile Methods told us so.

Let's examine the Agile Manifesto. It states in no uncertain terms that the four major values of Agile Methods are: (1) individuals and interactions over processes and tools, (2) working software over comprehensive documentation, (3) customer collaboration over contract negotiation, and (4) responding to change over following a plan. It's that old second value that not only seems to be the thorn-in-the-side of Agile Methods, but proponents of Traditional Methods as well. The reason for this schism that divides the communities of Agile and Traditional Methods, the Traditionalists in particular, is because thorough software documentation is one of the good old foundational values of Traditional Methods.

Proponents of Traditional Methods claim that good software documentation is the cornerstone of the Traditional Methods software life cycle, definition of requirements and design, software quality, and software maintenance. So without good software documentation, software cannot be specified, designed, or maintained, and ultimately results in low software quality. To add insult to injury, the creators of Agile Methods are flying in the face of good old conventional wisdom by defining a value that says "working software over comprehensive documentation." Therefore, in the minds of Traditionalists, Agile Methods are flat out wrong, misguided, and nothing more than simple hacking.

Let's step back and examine the history of Traditional Methods for a moment. MIL-STD-1521B required over 41 different documents to be prepared for software systems (U.S. Department of Defense, 1985). In fact, some of these had to be prepared for each major subsystem, making the total number of documents in the hundreds for each software system. DoD-STD-2167A required 33 major documents, which once again numbered in the hundreds when produced for each major subsystem (U.S. Department of Defense, 1988). MIL-STD-498 required 47 major documents, which numbered in the hundreds when repeated for major subsystems (U.S. Department of Defense, 1994). It's clear to see from these examples that hundreds of documents were necessary to document the requirements of large-scale military systems. Today, these are called Acquisition Category I programs, and are defined as those exceeding $300 million in research, design, development, and test dollars (e.g., pre-manufacturing, operation, and maintenance dollars).

Both domestic and international bodies sought to adapt these previously identified DoD software development and documentation standards for commercial use in the early 1990s. So, they adapted MIL-STD-498 to create ISO/IEC 12207 (also known as IEEE/EIA 12207). ISO/IEC 12207 recommended as many as 70 documents for commercial software systems, once again numbering in the hundreds when applied to major subsystems (International Organization for Standardization/International Electrotechnical Commission, 1995). The cost to prepare ISO/IEC

12207 documentation for a modest software system of 10,000 lines of code is around $3 million U.S. dollars.

Additionally, the Software Capability Maturity Model (SW-CMM), a de facto standard for "improving" the software process, required 20 policies, 52 procedures, 39 documents, 45 task orders, 81 records, 79 reports, 46 meetings, and 2,420 hours to implement all of these documents (Paulk, Weber, Curtis, & Chrissis, 1995). And, it's successor, Capability Maturity Model Integration (CMMI) required 25 policies, 489 procedures, 478 work products, and 21,579 hours to implement all of these documents (Chrissis, Konrad, & Shrum, 2003). SW-CMM's and CMMI's contemporaries, the Project Management Body of Knowledge (PMBoK), ISO 9001, and the DoD Acquisition Life Cycle continue this trend of requiring more and more software documentation to help cite customer requirements, plan projects, record architecture and design decisions, and improve software quality in order to reduce software maintenance costs (Defense Acquisition University. 2008; International Organization for Standardization, 2000; Project Management Institute, 2004).

This veritable "explosion" in software documentation standards requiring hundreds of software documents to improve the quality and maintainability of software products is exactly what caused the creators of Agile Methods to speak out (to put it mildly). In reality, they "banded" together to "lash out" at burdensome software documentation standards by creating the Agile Manifesto in 2001. The creators of Agile Methods had already been perfecting Agile Methods in the mid-1990s with approaches like Scrum. They were programmers and consultants themselves, who were tired of applying Traditional Methods to software projects to no avail. Initially, they too believed that document-driven software life cycles were the answer to the "software crisis" of the 1960s, until their projects started failing, time and time again. No software was being produced, projects were burdened with analysis-paralysis, years were passing without a single line-of-code being written, projects were being cancelled, and lawsuits were rampant for lack of performance.

Out of these "dark days" in the history of software development, Scrum emerged and then Extreme Programming quickly exploded on to the global scene in 1999 on the heels of Scrum. These methodologies promised something new, unique, and something not heard of for over 30 years. That is, they promised working software in 14 and 30 day increments. They had the "audacity" to demand "working software" instead of "comprehensive documentation." And, along with the worldwide acceptance of Extreme Programming, Scrum, and Agile Methods, they set off a firestorm of controversy. Why? Because, everyone was accustomed to spending years and years producing hundreds of software documents without a single line-of-code being written in order to ensure the highest possible software quality and maintainability of Acquisition Category I U.S. DoD programs (e.g., those exceeding $300 million in development costs).

And, there you have it, the fundamental misunderstanding that pits Traditional Methods against Agile Methods. Surely, according to traditionalists, the Agile Manifesto "bans" software documentation completely and thus must result in poor software quality and maintainability. Or, does it? First of all, the Agile Manifesto "doesn't" ban software documentation. In fact, the Agile Manifesto says, "while there is value in the items on the right, we value the items on the left more." What do you mean? What are these so-called "items on the right"? The "items on the

right" are: (1) processes and tools, (2) comprehensive documentation, (3) contract negotiation, and (4) following a plan. In other words, the creators of Agile Methods value documentation, but not more than: (1) individuals and interactions, (2) working software, (3) customer collaboration, and (4) responding to change. The creators of Agile Methods place an emphasis on high-quality interpersonal trust and communication, working software, customer interaction, and personal, technical, and organizational flexibility. They never said they don't value processes, tools, documentation, contracts, or plans.

We're going to examine a few case studies of how software documentation is actually used within Agile Methods. However, we must first introduce the software method called "Open Source Software Development." The term "Open Source Software" or OSS was coined in 1997, though the practice of Open Source Software started in 1970 (Bretthauer, 2002). Simply put, Open Source Software is a "set of computer instructions that may be used, copied, modified, and distributed by anyone, anywhere, and for any purpose whatsoever" (Fink, 2003). Another definition said that Open Source Software is "labeled with free source, fast evolution, and extensive user collaboration" (Zhao & Deek, 2004).

Why are we talking about Open Source Software Development? We're talking about it, because Open Source Software Development is another in a long line of specific types of Agile Methods. Open Source Software development also embodies the values of the Agile Manifesto: (1) individuals and interactions, (2) working software, (3) customer collaboration, and (4) responding to change. Like Agile Methods, the proponents of Traditional Methods have been relentless in their attack on Open Source Software Development for its lack of adequate software documentation. The software documentation practices of both Agile Methods and Open Source Software Development have been placed in the spotlight of public ridicule and the microscope of scientific scrutiny. So, we're going to look at a few case studies involving the documentation practices of both Agile Methods and Open Source Software Development in an attempt to shed some light on this subject.

One study of Open Source Software Development likened its software documentation practices to "user driven, just-in-time writing" (Berglund & Priestley, 2001). The study goes on to identify 25 techniques to document the development of Open Source Software. Some of the highlights are: (1) use traditional software document standards as "guidelines," (2) start off small and incrementally build documentation much like the software, (3) keep the documentation to a very minimum, (4) use electronic or online documentation instead of printed documents, (5) make it publicly available, (6) distribute the documentation via the Internet, (7) allow anyone to add to or update the documentation, (8) use responses to customer feedback and inquiries as a primary means of documentation, and (9) use static or dynamic video conferences as a form of software documentation. Below is a full list of 25 ways in which software documents are created in the Open Source Software Development model of Agile Methods:

- Allow as many people as possible to contribute to the documentation process.
- Turn technical debate in mailing lists and discussion forums into formal support for software products.
- Draw from an accumulated pool of resources and competence in text, multimedia, and live support.

- Create tutorials, standard documents, books, online reference manuals, and so forth as necessary.
- An absolute requirement for documentation is that it must be in electronic format.
- Use websites to logically organize, access, and distribute electronic documentation globally.
- Use a SourceForge server to distribute documentation
- Select and use an appropriate open source software license to distribute documentation.
- Accept the possibility of branching projects.
- Start out with small documents and allow them to incrementally evolve.
- Regulate the quality of the documentation.
- Create and enforce a social structure for creating and maintaining documentation.
- Use traditional styles such as user guides and how-to documents as much as possible.
- Adapt and synchronize document development to the use of short release cycles.
- Use live, people-to-people communication an integral part of the process (e.g., chats, webcams, video recordings, live transmissions, etc.)
- Use automated static analyzers to verify documentation (e.g., DTD validation, HTMLTidy reports, link checks, etc.)
- Use technical writers to improve language, correct errors, and identify gaps (rather than content generation).
- Use discussion forums, mailing lists, and FAQs as documentation.
- Use hyperlinked, electronic documentation as much as possible.
- Use XML-based annotation formats and systems.
- Use annotations to identify the source and reliability of the document changes.
- Define and utilize a documentation life cycle process to capture user questions and update documentation.
- Use topic-oriented writing and information typing to produce disciplined reusable information.
- Define and use specific documentation roles (e.g., collection-level editor, topic-level editor, etc.).
- Create and enforce document structure with markup languages and tools (e.g., HTML, XHTML, DocBook, DITA, etc.).

Three closely related studies describe some of the best forms of software documentation models in use today, specifically by software maintainers. What does this mean? Well, one of the tenets of Traditional Methods is that software developers should create as much software documentation as possible during the development cycle in order to make life easier for the software maintainers. In other words, conventional wisdom holds that software cannot be maintained without ample written documents from the development cycle like software requirements, architectures, designs, and so forth and so on. That's why the proponents of Traditional Methods created all of those standards such as MIL-STD-1521B, DoD-STD-2167A, MIL-STD-498, ISO/IEC 12207, SW-CMM, CMMI, ISO 9001, and the PMBoK. These standards advocated the creation of hundreds of software documents to "help" software maintainers.

Let's see what these so-called software maintainers say about their software documentation requirements. One study of 130 software maintainers showed that the two documents used most often during software maintenance were the software source code itself and the comments they

contained, and software architecture documents were considered the least important among software maintainers (de Souza, Anquetil, & de Oliveira, 2005, 2007). Another study of 96 programmers found that embedded design patterns in software source code improve code quality and reduce software maintenance time, rather than those captured in separate documents (Prechelt, Unger-Lamprecht, Philippsen, & Tichy, 2002). Yet another closely related study of 80 Open Source Software developers found that 98% use Eclipse TODO tags, 43% use Eclipse FIXME tags, and 15% use Eclipse XXX tags as a primary means of software documentation (Storey, Ryall, Bull, Myers, & Singer, 2008). Finally, a study of a study of 78 software maintenance personnel indicated that the presence of UML documents did not improve the speed and productivity of software maintenance tasks and the costs of producing UML documents outweighed the benefits of producing them for software maintenance tasks (Arisholm, Briand, Hove, & Labiche, 2006).

So, what's the bottom line? What have we learned here? We've learned that the proponents of "both" Agile "and" Traditional Methods "value" software documentation. They just value it in different ways. Proponents of Traditional Methods value hundreds of formally printed software documents prior to the software maintenance cycle at the cost of millions of dollars. The proponents of Agile Methods "first" value the production of working software releases and then the production of a variety of user-driven, just-in-time, electronic multi-media documents that embody tacit, interpersonal communication. We've also learned that not only do software maintainers use comments embedded in software source code as their primary means of software maintenance, but that these enhance the code quality more than written documents. And, I think we've learned that the proponents of Agile Methods have gotten a pretty bad, undeserved rap from the proponents of Traditional Methods who favor the production of hundreds of software documents before software maintenance (and probably deserve a pretty bad rap themselves).

REFERENCES

Arisholm, E., Briand, L. C., Hove, S. E., & Labiche, Y. (2006). The impact of uml documentation on software maintenance: An experimental evaluation. *IEEE Transactions on Software Engineering, 32*(6), 365-381.

Berglund, E., & Priestley, M. (2001). Open-source documentation: In search of user-driven, just-in-time writing. *Proceedings of the 19th Annual International Conference on Computer Documentation, Sante Fe, New Mexico, USA*, 132-141.

Bretthauer, D. (2002). Open source software: A history. *Information Technology and Libraries, 21*(1), 3-10.

Chrissis, M. B., Konrad, M., & Shrum, S. (2003). *CMMI*: *Guidelines for process integration and product improvement*. Boston, MA: Addison-Wesley.

de Souza, S. C., Anquetil, N., & de Oliveira, K. M. (2005). A study of the documentation essential to software maintenance. *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting and Designing for Pervasive Information (SIGDOC 2005), Coventry, UK*, 68-75.

de Souza, S. C., Anquetil, N., & de Oliveira, K. M. (2007). Which documentation for software maintenance? *Journal of the Brazilian Computer Society, 13*(2), 31-44.

Defense Acquisition University. (2008). *Integrated defense acquisition, technology, and logistics life cycle management framework*. Retrieved July 6, 2008, from https://acc.dau.mil/ifc

Fink, M. (2003). *The business and economics of linux and open source*. Upper Saddle River, NJ: Prentice Hall.

International Organization for Standardization. (2000). *Quality management systems: Requirements* (ISO 9001:2000). Geneva, Switzerland: Author.

International Organization for Standardization/International Electrotechnical Commission. (1995). *Standard for information Technology: Software life cycle processes* (*ISO/IEC 12207*). Geneva, Switzerland: Author.

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (1995). *The capability maturity model*: *Guidelines for improving the software process*. Reading, MA: Addison-Wesley.

Prechelt, L., Unger-Lamprecht, B., Philippsen, M., & Tichy, W. F. (2002). Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *IEEE Transactions on Software Engineering, 28*(6), 595-606.

Project Management Institute. (2004). *A guide to the project management body of knowledge (third edition): PMBoK Guide*. Newtown Square, PA: Author.

Storey, M. A. , Ryall, J., Bull, R. I. Myers, D., & Singer, J. (2008). Todo or to bug: Exploring how task annotations play a role in the work practices of software developers. *Proceedings of the 30th International Conference on Software engineering (ICSE 2008), Leipzig, Germany,* 251-260.

U.S. Department of Defense. (1985). *Military standard*: *Technical reviews and audits for systems, equipments, and computer software* (MIL-STD-1521B). Washington, DC: Air Force Systems Command (AFSC).

U.S. Department of Defense. (1988). *Military standard. Defense system software development* (DoD-STD-2167A). Washington, DC: Space and Naval Warfare Center (SPAWAR).

U.S. Department of Defense. (1994). *Military standard*: *Software development and documentation* (MIL-STD-498). Arlington, VA: Space and Naval Warfare Center (SPAWAR).

Zhao, L., & Deek, F. P. (2004). User collaboration in open source software development. *Electronic Markets, 14*(2), 89-103.