

Can we ever finish the paperwork?

Documenting software development can be a time and cost-consuming process.

Brian Hambling, technical director of software acceptance at ImagoQA, challenges the role of software documentation standards and asks whether they are really needed

Software engineering, by its very nature, is a confused animal. A long and painful history of high profile failures has left the community in a schizophrenic state of mind: one persona wants to wrap itself in process while the other wants to stride out into a Brave New World, armed only with the latest method that will enable more rapid solutions to more complex problems than ever before. It is the difficult function of the quality practitioner to resolve these conflicts and while they naturally align themselves with the pro-process group, the daily challenges they face have encouraged them to take a more pragmatic view. Software documentation, therefore, represents the fulcrum of two schools of thought; the ultimate symbol of process for those who see it as pure bureaucracy and the most tangible evidence of due process for those who see it as essential to success.

The debate is further complicated by the uncomfortable reality that the production of software documentation is usually an uninspiring part of a project that you have already given your heart and soul to deliver. Indeed, documentation is probably the only part of the software engineering discipline that is less popular than testing, because it can be an expensive and time-consuming activity. However, one of the main purposes of software documentation is to enable effective testing to take place and it remains an important part of effective testing practice, with new standards appearing regularly.

So, what is software documentation? It can be defined as anything that records information about a software product or service. The areas that are most focused on are requirements documentation, design documentation, test documentation and user manuals but there are a number of other areas that might need documentation for a particular system. All of the key areas need to be documented and, to be effective, all of this documentation needs to be consistent and accurate.

Software documentation has three important roles in the software development process:

- 1 To capture the requirements and the emerging design so that the development process can be controlled
- 2 To provide a basis for quality assurance and, in particular, for the design of the testing activities
- 3 As a baseline for future maintenance activities.

If this approach is implemented in detail it will be expensive in both time and resources. However, good documentation can reduce development costs and timescales by reducing the number of mistakes, omissions and repetitions that lead to rework. It can also reduce the cost of maintenance by making it easier to identify the impact of changes and easier to implement and test the changes. Furthermore, it can enhance quality by reducing the number of errors and the impact of changes made to correct those errors.

Documentation is also essential to enable the effective use of reviews and inspections which, ironically, are used to improve the quality of documentation. This demonstrates that there is a built-in assumption that better documentation means better code and better software products, but is that justified? Does good process make good product? If we get all the documentation right, will we get the right end product? Not necessarily so. In fact there is a long chain of logic that leads inexorably from the decision to deploy a structured development life cycle. An approach like the V life cycle, for example, demands that a sequence of development and test phases be defined, leading to the necessity to verify each phase, and thus the need of a baseline for the verification activities. This baseline is software documentation. The real irony is that few organisations that adopt this stance actually implement the life cycle effectively enough to reap any real rewards.

Of course there is a counter argument to this. This challenges whether the right end product could be achieved without going through the exercises of collating information, checking assumptions and demonstrating traceability and conformance to requirements. Put simply, does the process we go through to create documentation actually add any value? Yes, it certainly does, and the thought processes are essential even if the documentation is not produced, with the proviso that the thinking is done at the right time and by the right people. For example, if software documentation is written after the event it will have no real value. Equally, a technical writer who was not involved in the development process at all cannot be expected to write accurate documentation. It is true that using a technical writer to produce documentation after the event might minimise the cost of producing the software documentation in the short term, but there is a real risk that what is being produced has no practical value in the long term.

Standards are, therefore, necessary to avoid 'reinventing the wheel'. If we know what needs to be documented and we have identified a way of making documentation clear and useful, we should use that knowledge wherever we can. This not only saves time in production, but it also makes it easier to check that documentation is complete and at the right level of detail. A standard can also provide guidelines or templates to simplify the task for anyone new to the project.

It is also clear that there are several practical problems associated with using standards. However well written, standards cannot be 'all things to all people'. If they address one area particularly well they will inevitably be less well suited to other areas. A life cycle process standard, for example, cannot deal with development or testing issues in enough depth to satisfy the specialists. Similarly, a standard on structured testing may not offer much help to a tester

operating in an Extreme Programming (XP) environment. Often a standard can imply a particular world view that is not relevant to all its users; the author may be an advocate of a particular structured method or an evangelist for iterative development, and that flavour will be visible in the standard. As a result, many users find standards hard to work with because they do not fit either their technical culture, their business environment or their specific requirements. Conversely, the standard may require a level of detail in the documentation that is simply inappropriate in a particular case and the result is often that critically important information gets only superficial coverage because the documentation author tries to 'fill in all the boxes'. Quality checks based on standards, too, can be superficial because the checker is drawn into checking the 'fit' of the document rather than evaluating the content.

In the testing sphere, IEEE 829 provides an example of the issues that standards can raise. To begin with, IEEE 829 is entitled 'Standard for Test Documentation' but mostly addresses the specific content of a test plan. It is generally accepted that test plans are driven by test strategies and test strategies are driven by test policies. That is certainly the view of the BCS, as documented in both the ISEB Foundation and ISEB Practitioner Certificate syllabuses. Yet IEEE 829 embeds the heading 'approach' within its test plan outline. Does that mean test strategy? If not, what does it mean? Are the authors of IEEE 829 really sending a message about the relative significance of test strategies and test plans? The point here is not so much to question IEEE 829, a standard that many have found extremely helpful, but to recognise that standards can convey unintended messages that generate uncertainty.

Then there is the issue of unit testing. There are a variety of unit (component) test techniques with associated measures for test coverage documented in BS7925. Are developers expected to use these techniques and deliver code tested with these techniques to a defined level of coverage? While this might be an ideal scenario, it is not reality and even if it were so, would our products be better? Not necessarily. What purpose is the standard serving here? Is it setting a standard to which we should seek to aspire? Is it providing an arbitrary or academic baseline for good practice? Is it documenting impractical techniques for the sake of completeness? Like IEEE 829, BS 7925 is a very valuable standard, but its very completeness and comprehensiveness can intimidate the newcomer to testing and mislead the unwary into believing that the practices it documents are typical of the real world.

Another powerful example is the ubiquitous V life cycle. For decades, the testing community has based much of its theory, reflected into a variety of standards, on the principle that good development is based on good requirements and that testing based on those requirements should begin in parallel with the requirements documentation activity. Yet practitioners will be aware that 'good' requirements are as rare as rocking horse manure and requirements invariably change during the life cycle, causing massive rework to any test design based on them. They will also recognise that any life cycle that depends totally on the completeness, correctness and stability of initial requirements is fundamentally and fatally flawed. Given that the main requirement for software documentation and the standards that support it is the nature of this flawed life cycle (or rather the nature of any sequential life cycle), why do we continue to build standards around that model? Meanwhile, the really valuable lessons we have learned from the model – the approach we call risk-based testing – appears nowhere in a standard.

Perhaps the approach to software documentation just needs to be more creative. For example, writing a user guide before we write the application code could help to ensure that developers focus on what the user expects at a practical level. We could, perhaps, rely on software development tools to generate a database from which software documentation could be extracted on demand rather than writing it in case someone may wish to read it. Given that a professional system testing team will discover how a software product actually functions and will also know how its behaviour differs from the behaviour anticipated by the requirements, it seems sensible that the outcome of system testing should be used as the basis for system documentation. Perhaps system testers should be the generators of the 'as built' documentation. Perhaps

the test suite itself is the best representation of what the system does and the standard we need is a standard for generating a systematic test suite; that, at least, would have a value after the system is delivered.

Even more radical is the notion of testing before development, as advocated by adherents of XP and other Agile development methods. Agile development asks hard questions of conventional software documentation. Indeed, the Agile community disagrees with some of the fundamental principles that underlie the strong process approach of conventional methods, and one of their justifications is the large volume of documentation that gets created but never gets used.

Whatever path documentation standards take in the future, there is no question that they will remain integral to software development in order to maintain system quality. Test practitioners are faced with the balancing act of adopting best practice procedure for current standards while constantly questioning their relevance and benefit to the project in hand. This ensures that documentation supports practical development without becoming a theoretical burden. **PT**

Automated Testing Just Got Easier

Seapine
QA Wizard™

REGRESSION TESTING TOOL

- ▶ Powerful capturing capabilities
- ▶ Easy database connectivity
- ▶ Powerful checkpoint engine
- ▶ Browser-based toolbars
- ▶ User Friendly Graphical Interface
- ▶ Multiple filtration systems
- ▶ Organisational tree
- ▶ Highlight Indicators

Rapidly create automated test scripts for Windows and Web applications

CONTEMPORARY
Contemporary plc

Seapine Software™

Call now on 01344 297 613
Download evaluation at: www.seapine.co.uk