The Documentation Myth

It appears that software developers, and their managers, have a love-hate relationship with documentation. On the one hand, we all seem to be able to agree that documentation is *a good thing* and should write more of it. On the other hand, we are the first to acknowledge that don't produce enough documentation. Frequently I talk to developers who say they don't have the time to do it, while managers often don't see the point "shouldn't you have written that before you started?"

Still, when a new recruit joins an existing project we're quite likely to sit them at a desk with a pile of documents and somehow expect that by reading them they will learn the system. Chances are the poor guy will just end up being bored and have difficulty keeping his eyes open by lunch.

Then, when you actually give him some code to work on he'll find it was never documented, or, the document wasn't finished, or, its out of date, or, it's just plain wrong. How often have we heard a new developer ask for documents, even though he knows they won't exist, one has to go through the ritual of asking - after all, this is a professional outfit isn't it?

Now at this point in the article, writers like me are expected to launch into exhortations on why documentation is important, why managers and developers should take it seriously, and maybe even recommend an amazing new tool which will help you do this. Surprisingly this tool is available from my company for just £199 (plus VAT).

However, I'm going to save my breath. To my mind it stands to reason that if documentation was as important as we say it is then we would take it seriously, we would do it, and it would be available. End of argument.

This is not to say documentation is useless. Particularly when we are planning and designing our system the writing of papers, drawing of diagrams and discussions based around such documents are an essential way of analysing the problem and creating a shared understanding between team members.

Specification documents are somewhat more troublesome. They may be essential in order to start the project, indeed they may form part of a legal contract between customer and supplier, but we all know that specifications change. In fact, I'd argue that the process of writing the specification changes the specification. By documenting the problem we come to a better understand of the domain, this leads to new insights for all concerned and often leads us to view the original problem in a new way which also leads to new requirements.

OK, I'll accept that documentation has a role to play in helping new staff understand what is going on. However, diminishing returns are at work, the bigger the document the less information will be absorbed. The first few pages add more to understanding than the second hundred. Bigger documents delay the time when new staff actually does anything. Voluminous documentation can even discourage people from problem solving when they believe it is just a case of find the right page with a working solution (and how often have we each searched MSDN or Google for such a page?)

Even when there is documentation available we don't really trust it - and with good reason. We expect there to be a gap between what is says in the document and what is actually the case. The gap arises because things change over time, and because English and Java express different ideas.

In the worst case writing documentation becomes goal deferment - why complete the code when you can complete the document?

I worked on one project that followed a rigours ISO-9000 certified process. Despite very tight deadlines the documentation had to be kept up to date, as the documents grew the development work slowed, moral fell and the documents become more and more inaccurate. Even when they where proof read by others there was no check to make sure the document actually described what was in the code. Increasingly the documents said what the managers wanted to hear and the programmers wrote the code they way wanted to.

So what are we to do about the documentation myth? Which tool will solve my problem?

There is no technical fix for this problem. We need to rethink our view of documents. They are themselves a tool which allows us to discuss the problem domain, explore solutions and share information. However, they are a product of their environment. Inevitably they will address the issues at the time they are written, not the issues we find two years later. Documents will contain the information considered important by the people involved when they where writing. Other people, with different backgrounds and at different times will consider different information important.

Documents only contain explicit information that we choose to express. Much of the important information on a project is actually tacit and held in people's heads without recognising it. Information is also held in the practices and processes adopted by a development team and duplicating the processes won't necessarily replicate the information.

While our code base and deployed systems will always be the definitive source of information we can supplement these sources if we value other forms of communication - particularly verbal and cultural. This means we need to look to our people, we need to encourage them to communicate and share what they know. New staff shouldn't ask "Where can I find the documentation?" but "Who can I ask?"

Postscript

The Documentation Myth was written a couple of years ago - January 2004 according to the file. For some reason I never got around to finishing it, I still stand by everything I've said: documentation can be useful, but its probably not as useful as we often think it is. And the proof is: if it was that important we'd do more of it.

The reason for dusting it off and publishing it now is down to Peter Sommerlad's presentation at ACCU conference: *Only the Code Tells the Truth.*

Peter suggested that at the end of the day the only definitive documentation of a running system is the code itself. Not the system documentation, not the specification, not the UML and not even the comments in the system.

Then he went further. He suggested that many of the practises we consider important in software development may be myths. For example, comments in the code, design before coding, test after coding and strongly typed languages to name a few.

Some of these things were never true, we just thought they were a good idea at the time - well intentioned but not actually helpful. Others were useful once upon a time but things changes and maybe they aren't so useful today - for example strongly typed languages. And others were just down-right wrong, ISO-9000 for one.

I've written about this before in a way: its the need to unlearn somethings. Some ideas have passed their sell by date and we need to let go.

But this goes further, we need to constantly look at what we are doing and ask: does this add value? Does it contribute?

These are hard questions to ask and the results can be even harder to accept but we have to do it if our profession is to move forward and not get caught in sub-optimal positions.

However, I believe, like Peter that the time has come for some *Myth Busting*. Its a new century, our young profession is entering middle age, the time has come to look at what passes for conventional wisdom and question it. So, do you know any Myth's we should be exposing?