Is "agile documentation" an oxymoron?

Understanding the role of documentation in an agile development environment

Kurt Solarte

January 23, 2012

Does the term "documentation" have any place in an agile environment? The goal on agile projects is to keep documentation as simple as possible, relying on roadmaps, overviews and concepts rather than enterprise-focused details. But what happens when using an agile approach on more complex projects? For example, what if the team that writes the software is different from the team that must maintain it? Or what if auditors come calling? In these instances, basic agile documentation based on user stories alone may come up short. This article provides insights into how teams can take an agile approach to documentation in more complex environments.

Is "agile documentation" an oxymoron?

For many agile development teams, "documentation" is considered a dirty word. After all, agile teams work under the premise that uncertainty is so common and change so rapid that spending a lot of time on upfront architecture or design will be largely incorrect or irrelevant further down the line. Instead, agile developers opt for short, frequent iterations, and take a refactoring approach to architecture and design.

In a nutshell, agile development teams start with an idea, break it down into features and functions, write high-level requirements and then get started. The objectives are to document just enough to proceed with construction, complete a select set of the highest-priority requirements in each iteration, collect requirements in a work item list and produce documentation that anyone can use when they are working with the software.

Having documents that anyone can use is a laudable goal; however, in some cases, a document that is supposed to be fit for everyone ends up being fit for no one. As agile teams scale and fit into larger enterprise environments, the agilest must devise more mature, but equally agile, documentation strategies.

The maintenance documentation gap

Many business-critical projects, such as commerce applications or portals, are now adopting agile practices on their projects. In these projects, it's common to have an outside agile development
shop creating the application that will be handed over in 12-18 months either to the hiring company itself or to another third party vendor for ongoing maintenance.

Although lightweight stories and requirement artifacts work really well for construction, they may not be sufficient for ongoing maintenance. These stories and artifacts do not have the information needed to maintain the application for a number of reasons.

Often, when a team gets in the middle of a big project, things change and those changes might not make it back into the documents. The attitude is that the software has already been delivered, and it's time to move on to the next iteration. So, maintenance ends up with out-of-date documentation. In addition, agile documents are often very minimalist without the overview often needed by a maintenance team.

The audit documentation gap

In some industries such as financial services and healthcare, there are outside rules and regulations that applications must comply with. Because traceability is important in large projects, large agile teams often use tools to link artifacts for traceability, but at times, these are not sufficient.

For example, I worked on a big portal implementation for a pharmaceutical company. It was an internal portal that involved data from drug studies. In that situation, not only was the data access subject to audit, but also the construction of the data access portal itself. Although we linked artifacts, we were very focused on developing the application and website, and there was a scramble for documents and required traceability when the auditors showed up.

Like maintenance documentation, the information auditors need is not necessarily found in "one-size-fits-all" requirements documentation. Attaching traceability to the project is also not sufficient. Auditors need documentation that demonstrates compliance.

Agile documentation

When agile teams are faced with these two scaling factors, separate maintenance teams and external audit requirements, the Disciplined Agile Delivery framework and a practice such as Document Continuously are good places to start. Closing the gap between the documentation produced for agile projects and the needs of maintenance teams and auditors can be achieved by taking the same approach to delivering documentation as that taken for delivering code. Instead of producing documentation that everyone should be able to use but can't, think about providing maintenance and audit documentation as separate deliverables produced as part of the iterations.

In the projects I work on, we detach requirements, stories and artifacts from one another and create other deliverables that are part of the iterations. For maintenance, we build some frameworks based on what maintenance wants. We harvest things out of agile requirements and then look at what gaps exist. Part of the work for each iteration involves filling in the gaps.

For audit documentation, we get the full requirements and make them part of the story. In fact, the story is not complete until we have run the report and compliance matrices and connected them
to the story. We recognize that we might have to add more traceability, that compliance and audit regulations are continually evolving and emerging, and that we need to keep up.

A plan for coexistence

Requirements, artifacts, maintenance documentation and audit documents can all coexist, not necessarily in the same location, but as part of different iterations. And when you take the just-in-time and fit-for-purpose approaches, all of which include taking advantage of reusable parts when possible, generating documentation is easier.

After all, when you look at maintenance manuals or administration guides, you can see that they contain some elements found in design documents. Some of the traceability required by auditors is similar to what agile project teams want, such as links between business need, development work item and acceptance tests.

In the agile projects I work on, we identify these similarities, publish out a shell and build a template with an outline that includes an extract from the business process flow or architectural diagram. Some of the objects we create in other tools can be inserted in the manual. The result of this approach is the creation of "mini-documents." Our mini-documents for maintenance have been more popular than even the older large design documents because maintenance teams can move right to the space where a defect is detected. They no longer have to deal with huge documents that are difficult to navigate.

Conclusion

Documentation for applications built by agile development teams that are based on requirements, artifacts and stories can come up short when faced with the scaling factors of separate maintenance teams or regulatory audits. Providing documentation that suits different purposes is possible if you take an agile approach to how you scale. Determine documentation needs as you progress and include the different documentation builds in your iterations and you will provide the right documentation solution for all stakeholders, including maintenance and audits.