Keeping Development Standards Healthy and Current

An essay on the effectiveness of standards used for software development

Ask a roomful of software engineers what they consider important to be included in a definition of standards for software development and you are likely to get a variety of responses. Every developer has an opinion about what constitutes good, well-designed and well-written software and it is commonplace in the data processing industry for an organization to have a published standard available encapsulating the important aspects of software design all developers are expected to follow. Enabling these professionals to contribute to forging the standard through a process of continual improvement makes for a very healthy and current standard.

Companies at which I have worked over the years range from those with no software development standards to those requiring one hundred percent inspection of software for compliance with voluminous documentation describing their software development standards. Some regard a development standard with the perspective of "less is more" while others have embraced a "more is better" approach. Many of those in the latter camp tend to prefer a standard describing in detail how developers should write code, with elaborate guidelines for naming conventions, often overlooking why a standard exists in the first place and ignoring whether it provides any benefit to the organization.

The emergence in the late 1980s of entities such as ISO-9000 and Capability Maturity Model Integration (CMMI) has motivated some organizations lacking a software development standard to produce one. Some companies adopting such quality guidelines have rushed to define a software development standard where one previously did not exist. During one of my assignments the company was undergoing a certification for ISO 9000, for which a highly detailed development process was hurriedly compiled.

To be effective, a standard must provide a foundation for the quality of software engineering, and needs to evolve with architectural changes in the software industry. Standards defined primarily for marketing purposes or in response to requirements for organizations seeking to be awarded a higher level of maturity compliance will stagnate quickly once the award has been won or the marketing goal achieved.

Tom Gilb and Dorothy Graham, in their book Software Inspection, offer this depressing observation about documentation describing a software development standard:

> "You can expect the following situation regarding rules:
> (1) There are existing software engineering standards. Nobody uses them. They are not in useful shape. Too voluminous, too much trivia.
> (2) There are several areas needing rules for which there are no standards whatsoever."[1]

and caution against instant creation of missing artifacts:

> "There are usually no checklists, at least no really useful and significant ones available to begin with. Do not get somebody to draft checklists all at once. This is invariably wasted effort, because:
> - Too much trivia gets on the lists;
> - The lists are too voluminous;
> - Such lists have no credibility;
> - There tends to be no way to get rid of them, except to ignore them"[2]

Despite their best efforts, many organizations succumb to these traps.

In some cases software development standards are created in isolation by only a few developers who define the standard to be followed by all other developers.

Worse, the resulting software development standard often is nothing more than a collection of stylistic preferences of the authors.

1 Gilb & Graham, *Software Inspection*, 1994, p. 259.
2 Ibid, p. 261.

Worse yet, management often considers the final definition of the software development standard to be the end result, expecting the standard to require no further changes.

The inevitable result of this perfect storm of circumstances is a development standard that is not embraced by those who are expected to comply with it, that does little toward insuring the quality of software, and that atrophies over time due to the inattention by management of implementing policies for keeping the standard current. Software development standards following this path to existence eventually become impediments to quality, raising costs rather than lowering them.

On one of my contracting assignments I encountered a software development standard that described the state of the software environment at the moment in time the standard was written. Over a period of nearly ten years it had not changed to keep up with the subsequent architectural advances of the environment, yet all updates and new software development still was subject to review against this obsolete standard. Demoralized developers would share their despondence and frustration over how this standard was choking their productivity, to each other and to management, yet efforts undertaken to change the process to enable the development standard to reflect the current architecture and to adapt to environmental changes were largely ineffective.

In his book The Fifth Discipline, Peter M. Senge lists possible attitudes toward a vision:[3]

Commitment	descriptive of those who want it, creating whatever laws and structures are needed;
Formal compliance	descriptive of those who see the benefits, doing what is expected and no more;
Grudging compliance	descriptive of those who question the benefits but, not wanting to lose their jobs, do enough of what is expected without concealing the fact that they are not on board;

When it comes to software development standards, most developers I have met fall into the category of Formal compliance. Those interested enough to become part of a process of maintaining development standards fall into the category of Commitment or of Grudging compliance, depending on whether the organization provides a way to facilitate their involvement and tap the potential of their contributions.

Important as development standards may be to an organization, it is equally important that there be a process by which the standard remains current with changes in the industry. A static standard may be applicable within a few months of its definition but over time it becomes less and less useful. The need for an evolving standard is raised by Mary and Tom Poppendieck in their book Leading Lean Software Development:

> "Standards are the baseline of current good practice in an organization that everyone is expected to achieve. They are sort of like learning the scales in music. One of the roles of experts is to make sure that standards are in place. Those who have learned through hard experience the best ways of doing things in their environment should make sure that others don't have to reinvent the wheel. But standards are never perfect and need to be constantly challenged and improved. So once someone has mastered a standard practice, a good learning exercise would be for a mentor to help that person challenge aspects of the standard that need improvement and come up with a better way of doing things."[4]

When I hear developers complain about the software development standard, it is primarily for these two reasons:

- The existing standard already has become obsolete and there is no formal process in place by which to keep it current
- Rank and file developers feel left out of the process of contributing to a standard by which their development efforts will be measured

Mary and Tom Poppendieck address both of these points in their book Lean Software Development – An Agile Toolkit:

3 Senge, *The Fifth Discipline,* 2nd edition, 2006, p. 203.
4 Poppendieck, *Leading Lean Software Development*, 2010 pp. 94-95.

> "Developers appreciate reasonable standards, especially if they have a hand in developing them and keeping them current."[5]

Organizations are apt to get better results from software development efforts when the developers are committed, as opposed to formally or grudgingly compliant. Instituting processes for keeping development standards healthy and current by those who are held to those standards is a simple solution for turning compliant developers into committed developers.

Jim McDonough – Febraury 20, 2012

5 Poppendieck, *Lean Software Development – An Agile Toolkit*, 2003, p. 122.