W3C use of git

The W3C uses Github, as described in https://w3c.github.io/. Their use is not universal. Each project can be different, and not all use *git*. They all comply with a governance policy described in https://www.w3.org/2021/Process-20211102/ with the standards writing, review, and approval process covered in https://www.w3.org/2021/Process-20211102/#rec-advance

The documents are prepared in *ReSpec* which is a constrained subset of HTML, and they are validated automatically to stay within the constraints of *ReSpec*. There are a variety of details about the process that can be found if you follow links in https://www.w3.org/Guide/. It looks very similar in concept to the DICOM process, although W3C is structured as many independent recommendation documents rather than one huge standard.

IHE use of *git*

IHE ITI is using Github. If Steve Nichols is there at the meeting he can probably explain the details better than I can.

I did get one immediate reaction from Neal Tenenbaum. We were working with Radiology on the enhancements to SOLE and then we started looking at IHE ITI as the better home. He immediately was happy that at last there was a group using modern technology (*git*) and modern methods. This makes him much more eager to work with them.

Considerations when switching to *git* or a *git* based service

git is agnostic when with regards to the change governance process. This can be confusing.

The dominant form of change governance process used by the Github community is a Continuous Integration / Continuous Deployment process (CI/CD). That process is not what SDO's use, nor is it one that I've seen used for Class Two or Three regulated medical devices. SDOs do not approve of allowing any team to publish a new version of the standard without further review.

This will result in confusion because many of the articles about *git*, GitHub, and similar services assume a CI/CD process, and many of the tools are intended for a CI/CD process.

For example, there is a huge effort and discussion around how to automatically detect, prevent, and reconcile situations where two groups or people have modified the same code without coordinating their efforts, and as a result they release a defective product to the public. That's very common in CI/CD if not managed properly. It's quite rare in the SDO structure where modifications are approved in advance, and supervision of development acts to discover and resolve overlapping efforts before the changes are committed and distributed to the public.

The *git* community is quite emphatic that establishing a clear change governance process must be first if you want your project to succeed. It's also clear that using *git* is independent of the choice of document structure.

The important first choice is deciding what *git* workflow will be used. There are already common workflows defined that can be used without change, or with modifications. These include:

- Centralized workflow, providing a setup very similar to the old CVS.
- Feature branching, https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow, where each feature is fully developed on its own branch and then merged. (This is much like our CPs and Supplements)
- GitFlow, https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow, which extends feature branching to incorporate multiple independent releases and versions. This addresses concepts like back ports of bug fixes into earlier versions.
- Forking workflow, https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow, where there are independent public and private *git* repositories that operate internally as each group prefers, but do so in a manner that makes subsequent later merging straightforward. This is analogous to our Working Group structure.

Github, GitLab, and others have extended the basic *git* with their own GUIs and proprietary extensions. As long as they continue to preserve *git* compatibility this is not a problem. I use my local *git* repository and local *git* tools to manage software and document changes. I can synchronize with GitHub, GitLab, and several other systems without problems. For some administrative tasks I need to use the GUI, but this does not interfere with my regular non-administrative uses.

Documenation formats

Github supports following markdown languages within their GUI. It will render them automatically as part of the web interface.

```
.markdown, .mdown, .mkdn, .md -- Github flavored Markdown
.textile -- Another markup language
.rdoc -- ruby documentation
.org -- emacs org-mode
.creole -- another wiki markdown
.mediawiki, .wiki -- Mediawiki markdown (wikipedia)
.rst -- ReStructured Text
.asciidoc, .adoc, .asc -- asciidoctor
.pod -- Plain old documentation (perl)
```

Github clearly favors the use of *Github Flavored Markdown*.

I've been using *Asciidoctor* for historical (O'Reilly books) and functional (complex tables) reasons, and it is also rendered by Github. The biggest issue has been that my local Asciidoctor software also understands the inclusion and editing of *plantuml* graphics. GitHub Asciidoc rendering does not understand *plantuml*. To keep Github GUI happy I keep my plantuml documents separately as puml files, render them to svgs, put both the puml and svgs onto Github, and in the Asciidoc file I refer to the SVG files for inclusion. That is sufficient for the GitHub GUI tools. (Keeping plantuml diagrams as separate files, etc., is not a serious problem.)

We can stay with our present mix of Word and Docbook files if we also stay with our present policy of adding new files for each new version. *git* will have no problem with that, and it will not conflict with policies of GitHub, GitLab, etc.

A problem will arise if we decide to use *git* to track internal changes within files. The *git* delta processing assumes that the files are line oriented text, i.e., programming languages, markdown of various sorts, CSV, or plain text. When given binary or XML they can behave badly. Not only will the deltas be poorly chosen, the various GUIs do not understand Word or XML, so the display of the deltas will be hard or impossible to use.