I-SENSE Artifact Management

# Background

I-SENSE teams produce a variety of work products that are used to specify, design, build and test the ultimate sensor product. These work products typically undergo revisions as the project progresses.

These intermediate work products are referred to as *artifacts* and include things like use cases, designs, schematics, bills of material, quotes, PCB designs, requirements, technical feasibility analyses, source code, test plans and test results.

The file type for each artifact depends on what tool was used to produce it. For example, source code editors like Notepad++, Eclipse and Atom produce plain text files. Other common artifact file types include Word documents (.docx), Excel spreadsheets (.xlsx, .csv), Eagle schematics (.sch), Eagle PCB designs (.pcb) and Visio diagrams (.vsdx).

Engineering work is often iterative. Artifacts typically change as new information becomes available. For example, a software design might need to be revised when a chosen data structure no longer meets new performance requirements. Alternatively, a schematic may need to be revised when a selected capacitor is no longer suited for new environmental conditions. Older documents often need to be revisited and modified.

{ meaning that the current work reveals that earlier documents}

A simple, lightweight process is defined below that employs git technologies

# Objectives

A document management system will be deployed within I-SENSE to characterize and store documents. The document management system will meet the following objectives:

* provides one central repository for {all documents | project-related documents only}
* each project will be represented by a project portfolio (a repository)
* each project portfolio will contain one or more documents created by one or more people representing one or more engineering disciplines using one or more tools
* scalable to ten projects per year
* a local Wiki will be used to describe the project portfolios
* provides a mechanism to prevent against catastrophic loss (disk/server crash, hurricanes, etc.)
* project portfolio will help new team members come up to speed quickly
* project portfolios are loosely dependent on development process

XXX

|  |  |  |
| --- | --- | --- |
| **Tool/Toolset/Framework/IDE** | **File Formats** |  |
| Source code editors (Notepad++, Atom, etc.) | Plain text |  |
| Eagle | .sch, .brd |  |
| Visio | Visio diagram, PDF, JPG, PNG, and others |  |
| IDEs like Eclipse, IntelliJ | .java, .js, .h, .c, .cpp, .xml (plain text) |  |
| MS Word | .doc, docx | *git diff* does show diff’s even for a .doc(x) files |
|  |  |  |

# Quick & Dirty Document Management for I-SENSE

* project teams will use git technologies (GitHub, git bash, etc.) to store project-related documents
* documents include all engineering artifacts (designs, specifications, schematics, PCB designs, bills-of-material, database schemas, test plans, test results, etc.) of any file type
* documents *can be* checked-in anytime, for example, to snapshot a stable work-in-progress
* documents *must be* checked-in when they are associated with important milestones
  + - documents that are used to start a project
    - documents that specify or comprise a prototype sent to clients
    - artifacts that make up a proof-of-concept
* for each new project
  + a project name is selected
  + an I-SENSE project repository is created on GitHub() using the project name
  + each team member creates his/her local repository (git init)
  + for each document or group of related documents, the document is
    - added to the local index (git add)
    - committed to the local repository (git commit)
    - pushed up to the remote I-SENSE repository on GitHub()
* Security
  + General GitHub access credentials: <https://help.github.com/articles/updating-your-github-access-credentials/>
  + Use public key instead of password each time
    - Check for any existing keys by opening git bash and typing “ls -al ~/.ssh”

# Prototype: *theProject* repo

The prototype repository (repo) is called *theProject*. This repo contains high-level directories (folders) for requirements, architecture, hardware (hw), software (sw), mechanical (mech) and test-plans.

# Using *git subtree*

An existing GitHub repository can be merged into a directory of another GitHub repository. This is helpful, for example, when creating a master project repository from the repositories of one or more contributors.

|  |
| --- |
| $ // this labels a remote repository as a subtree to the existing repository  $ git remote add serial-tree https://github.com/SeaGuy/serialCommFun.git  $  $ // this creates a new subdirectory with the repository and copies the subtree here  $ git fetch serial-tree master  $ $ // even though git status shows nothing to be done, go ahead and add, commit and push  $ git add .  $ git commit -m "merged serial fun as subtree" $ git push -u origin master |

Figure 1 - Create git subtree

# Use Cases

1. Document Checkpointing
   1. *I’m getting ready to leave for the evening, so I’ll save the conference paper I’m working on and push it up to the project portfolio.*

The importance of this use case is proportional to the length of time since the last push. How many days work are you willing to lose? What if your disk crashes and it’s irrecoverable?

Pushing it up to the repository would be an example of “non-milestone” checkpointing. A better alternative (requiring much less work) would be to store non-project-related documents in-progress on the *isense\_research* shared network drive ([\\engvault05.eng.fau.edu\isense\_research](file:///\\engvault05.eng.fau.edu\isense_research)).

* 1. *I just finished the first draft of my conference paper. Since I’ll be sending it out to my colleagues for review, I’ll save the draft and push it up to the repository.*

This use case is more important than the previous one, because it associates a specific version of the document to a development process milestone.

In this case, there is no ambiguity about what version of the conference paper colleagues are reviewing.

In this case, pushing a copy of the document sent out for review u to the repository will guarantee no ambiguity about what version of the conference paper colleagues are reviewing. However, since this can also be achieved by a thoughtfully-chosen document numbering scheme, this use case is not recommended for document management.

# Candidate Technologies

* GitHub (branching, releases, tagging, etc.)
* Markup
* Semantic Versioning (<https://semver.org/>)
* Yet another bullet

# Issues/Questions/Risks

XXX

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | Do we want to include all documentation (research, trade studies, journal articles, conference papers, posters, etc.)? | Probably, yes.   * PRO: Why wouldn’t we want a central repository that holds all I-SENSE documents in one easily accessible location? * CON: Isn’t it overkill to be pushing conference papers up to GitHub? |
| 2 | Look into GitHub API |  |
| 3 | There are currently 30 I-SENSE repositories on GitHub already. How can we make it easy to locate and access a repository of interest? |  |
| 4 | What about data? |  |
| 5 | What documents will be required, suggested, etc.? |  |
| 6 | Use Semantic Versioning (<https://semver.org/>) to determine the GitHub tag for a new release? |  |
| 7 | One of the requirements for an document management system is that it scales to ten projects in the first year. | What about all the non-project related work? If non-project related work is not suitable for document management, then another scheme should be employed. Otherwise, where will all the theses, dissertations, journal articles, posters and conference papers produced by I-SENSE be archived? Or will they be remain solely on the authors’ computers? |
| 8 | Could get a little cumbersome. For example, in theProject prototype, we have three Eclipse projects that share one GitHub repository. |  |
| 9 | *git diff* is not useful on binary files like Visio files. For binary files, *git diff* merely states that the two files differ. |  |