SDEVEN Software Development & Engineering Methodology

Version: 7.0.14
Release date: 230812

Practices (SDEVEN.20-RENPRA)

Table of Content

This procedure recommend "day-by-day" practices in software development work targeting the following objectives:

- to assure a good level of communication and exchange information
- to create premises for a good synchronization of activities and work
- to allow for clear presentations of work results and obtain best benefits

Git usage

Recommendations are more easily applicable when using standard `git` systems (*practice strongly recommended by SDEVEN methodology*).

General common and frequent aspects

Here is a list with the most common and frequent situations:

- *never change anything for a closed version* or *issue*. Normal way is to create a new issue instead of changing the existing one.
- organize development issues in *sprints* as small chunks of changes that have clear objective, specs (following Agile principle) and a short enough deadline to remain "*useful & valuable*" at finish

- when work for an issue always create a dedicated branch, and make STRICTLY WHAT IS INTENDED, EXPECTED and REQUIRED TO DO (otherwise could be difficult to reverse work, for example in case of something goes wrong with unlikely impact to the quality of result and the deadline term). Respect the principle that states "*when you do something, do ONLY THAT THING and do it WELL*".

Branches and repository

- always make a branch for each change / sprint, even is a short one (will allow you to quickly rollback work) - this branch should be locally on your development machine but is not mandatory, it could be remote and devops engineer should be notified
- try to *avoid mixing with other branches* even if they're still yours (as work in progress)

Releases check list

Here is a check list regarding most important issues that need attention before closing a release:

- check for still open, in progress sections; look for specific words like `wip`, `...` (ellipses), `todo`, `fixme`, `bug`, `need review`, etc. Ignore case when search for these words !
- check release notes: if exists as separated file or there are marked in a clear way, not mixed with things intended or in work for other versions
- check for version code (at least major, minor, patch) to be in according with roadmap
- check technical documentation: specs for usage, notes for developers
- check for end user documentation: updates, references that released features are available from version x, "how to use features", etc
- check the language used in end user intended documents to be as most as possible IMPARTIAL and avoid misinterpretations
- check for other elements with impact on branding, such as logos, colors, fonts, etc

Technical issues regarding syncing and distributed execution

- ref sync subject objects it is recommended to be accompany them with useful metadata at least with info ref to *last sync* date time stamp
- in multi systems sync (more than 2 involved in sync process), every system should have its own list with targeted systems to be synced; this list itself is subject to sync
- generally ref syncing it is recommended to use standard components and technologies, like `rsync` or derivate but largely enough used and maintained by producer; clearly should be avoided solutions that are available only on few systems (and in this case this should be explicitly documented)
- ref distributed execution of processes it is recommended to use already known components that have enough support as community and are dependent only of other known components, for example for queues and pub /

sub systems, Rabbit MQ, Redis, AMPQ, can be used; proprietary closed systems should be avoided (can be used only in custom / dedicated / turn key systems if the beneficiary want strictly a component)

- in file names intended for code modules / parts / chunks AVOID the character dash (`-`). Replace it with underscore (`_`). In many languages the inclusion of the other files in code is made using some pre-processor directives (as `include`, `import`, `require`, and so on) and these directives does not always accept strings but directly filenames and often the dash character is treated as ***minus arithmetical operator*** which can lead to many *"hard to detect"* problems.

More information, techniques and practices can be find in template of Software Design document.

Tool stacks components versions

- new (not enough tested in market) version of a toolstack component must be avoided, especially when is a core one for system where work is done
- if a feature intended to be used is not backward compatible, before using or updating it must check for:
  - the impact to already developed or in development code by any member
  - adoption of this version in standard operating systems

Functions signature parameters

> **Optional chapter**
>
> This chapter makes subject of *optional* recommended practices

- always make a local (in function code) copy of received parameters. This is to minimizing risk of generating unwanted side effects, *except* you really want to change their values and this change to be "seen" by caller
- *PAY ATTENTION that making a copy of parameters is not enough to avoid side effects*; if they are `mutable` (ie pointer, address) its update will alter the original value with unwanted side effect (unless is made intentionally)

Last update: August 13, 2023