SDEVEN Software Development & Engineering Methodology

Version: 7.0.12
Release date: 230805

Project structure (SDEVEN.62-PSTR)

Table of Content

Preamble and goals

This procedure contains usual project structure and it is just a recommendation. The *Project Manager* will organize the project in the best possible mode in order to to be relevant in specific project situations. A common practice is to start with these recommendations and to add (or refine) elements that reflects project particular aspects.

Project basic backbone structure

First level of project backbone consists of:

- **830-DEV** - here will take place all system "active" development
- **880-RLSE** - here will be kept data for public releases - this directory is not be explained here, for details see procedure 60-RELM

All product system code is kept under `830-DEV` directory. The objective of its structuring is to assure as much as possible code reusability and its "after-release" maintainability. This directory contains:

- **<project root>/830-DEV/** directory go to section with *following structure*:
  - **doc_src/** go to section
  - **docs/** go to section
  - **pjm/** go to section
  - **setup/** go to section
  - **static_portal/** go to section
  - **syslnit/** go to section
  - **<system_module_A>/** - directory dedicated for <system module "A"> go to section
  - **<system_module_B>/** - directory dedicated for <system module "B"> go to section
  - ... <another system module>/ ... go to section
  - **Commons**/ go to section

Each of these directories will be explained in next sections.

> **Naming conventions**
>
> To avoid conflicts and misinterpretations at programming language level it is recommended that in FILES and DIRECTORY NAMES to avoid characters space ( ) and (-) and to replace them with underscores (_).

For a clear "picture" please refer the "Example of project full directory structure" section.

doc_src directory

- the technical documentation:
  - 110-SRE System Requirements
  - 120-CPTS System Concepts
  - 130-SKIT Sales Kit(s)
  - 810-DSGN System Design
- system manuals
  - euma
  - adma

> **system manuals**
>
> system manuals (adma & euma) will be assembled as deliverables in release packages for details see 60-RELM procedure

docs directory

This directory will accommodate the **FINAL** (**RELEASED**) documentation static portal that accompanies the developed system. This is part of what is known as "Help Center" of that system This is mandatory for products from category "*ENTERPRISE SYSTEMS*".

remarks

- this directory content is obtained from `static_portal` directory after tests passed and as preparation for a release (here go to static_portal directory section)
- this directory is subject to git repository as is part of a release

pjm directory

Here are kept project management items that could be necessary in software development [1], things like that:

- project contract
- project tests & acceptances procedures
- deliveries content and schedule
- ... etc

project management documents

the project management documents make subject of Project Management discipline and will not be explained here or in other SDEVEN section

setup directory

The aim of this directory is to keep code to install the system by this understanding the code that:

- create all directory structure required to accommodate and run developed system
- create all OS level users, groups or other administrative OS "items"
- install all required OS level dependencies and applications (for example a local particular database system, a system application used to manage the network components, etc)
- install the framework(s) components that are required to run developed system (for example JRE for Java components, PHP Laravel, Python Flask, etc)
- configure OS installed components in corresponding directories (for example on Linux some changes in directories `/etc`, `/var`, etc)

setup components language

A general practice is to make setup components in usual OS scripting language (Bash, Power Shell, etc) but is not mandatory to do like that. A good practice is to use a language that:

- can assure enough independence of OS specific commands and "formats" (for example the directory separation character, `\` vs `/`)
- can be executed on all known public OS-es (Linux, MacOS, Windows)
- one of the "perfect" candidates is *Python 3*

static_portal directory

This directory will accommodate the documentation static portal that accompanies the developed system. This is part of what is known as "Help Center" of that system This is mandatory for products from category "*ENTERPRISE SYSTEMS*".

how to create documentation static portal

- the company practice is to use mkdocs to build this portal
- this directory is used for testing and validation resulted portal - released portal is kept in `docs/` directory

sysInit directory

The `sysInit` directory accommodates code that initialize all system modules. This system initialization routine **SHOULD BE THE CENTRALIZED ONE** meaning:

- each system module / component must have its initialization code (as described in "system module X")
- the `sysInit` code centralize all modules initialization in **correct order**

The code of `sysInit` module should be called repeatedly without generate side effects except that determine system initialization and loosing all sessions in work data. But repeating calls should all system data is correctly flushed and persisted and no **UNEXPECTED *missing*** (of course others that "unsaved data") or other files, configurations ***damage*** is happening.

<system_module_X> directory

The system must be designed following the next principles:

- must be structured in "independent **modules**" (see the next explanation)
- **modules** should interact between them ONLY:
  - using parameters and returns
  - using defined interfaces (as recommended in OOP guides)
  - using a external shared - common - data component

- interactions or communication between **modules** that require global variables should use the **Commons** component (see Common section)
- should have their own *initialization* code callable from `sysInit`
- should have their own `README_moduleX.md` file containing specific technical specs and info (will become technical documentation)

> **What means an independent module?**
>
> A software module can be considered independent enough when it can be "transformed" into a distinct library with an *acceptable work around* effort meaning without change or alter its functional code but only the required code to make it separated "package or library" (ie, the code that define its library definition)

Commons component

This component is a specialized module used to replace direct usage of ***global variables***. It usually is implemented as a `class` object and take care of global variables by meaning:

- assure their consistency such as they are critical regions
- prevent circular references when using them (everybody import only `Commons` module)

`Commons` component (if is present) should have data initialized by each module that post any global data and in `sysInit` module should be among the first created, if not the very first.

> **Commons component code-name**
>
> The `Commons` component has the name starting with *uppercase* especially to avoid confusions with `commons` name which can be used in more other contexts being an usual and general term. So, the idea is **to use in clear** `Commons` instead of `commons` and to potentially get some warnings at least in stating / initializing phases...

<project_root>/830-DEV/ directory

In the project root directory will be at least these files:

- `README.md` which contains a kind of product data sheet with project information
- `project.toml` which contains project information like:
  - *name* - product / system / project code-name / short-name as known in organization
  - *description* a short description of the project (just emphasizes the essence or "reason to ve" of product because more detailed information is offered through README)
  - *version* is the product version (the product in that package !) and must conform all SDEVEN versioning specifications
  - *license* type

    - ... more information, usually this file being also required by PACKAGING AND DEPENDENCY MANAGEMENT used solution ...
- `requirements.txt` which contain product / system internal and libraries dependencies (just system level not OS level)

Example of project full directory structure

Here is shown an example of project directory structure starting from a `PROJECT-ROOT-DIRECTORY`.

```
<PROJECT-ROOT-DIRECTORY>
├── 830-DEV/
│   ├── doc_src/
│   │   ├── 110-SRE/
│   │   ├── 120-CPTS/
│   │   ├── 130-SKIT/
│   │   ├── 810-DSGN/
│   │   └── other_project_docs...
│   ├── docs/
│   ├── pjm/        # organization specific project management and contractual docs ...
│   ├── setup/
│   ├── static_portal/
│   ├── <sys_module_A...dir>/
│   ├── <sys_module_B...dir>/
│   ├── <sys_module_X...dir>/
│   ├── Commons/
│   ├── SysInit/
│   ├── project.toml
│   ├── README.md
│   └── requirements.txt
└── 880-RLSE/       # specific organization (see procedure 60-RELM) ...
```

---

1. The reason that project management documents are kept "in development repository" is to be available for the whole team. This is not mandatory and in special cases this directory can be moved out of development repository. ↵

Last update: August 5, 2023