SDEVEN Software Development & Engineering Methodology

Version: 7.0.12
Release date: 230805

Appendix B System Design Document Template (810 DSGN)

Table of Content

- Appendix B System Design Document Template (810 DSGN)
  - Preliminaries
    - Structure of document
    - Structure & content of deliverable
  - 100-ANA Analysis
    - 110-SRE System Requirements (REQUIRED always)
    - 120-CPTS System Concepts (REQUIRED in almost all cases)
    - 130-SKIT Sales Kit(s)
      - 130.01 Product Datasheet (REQUIRED always)
      - 130.02 Product Overview (REQUIRED if not present in Datasheet)
      - 130.03 SPIN documents (OPTIONAL)
      - 130.04 Licensing Editions & Pricing (OPTIONAL)
      - 130.05 Service & Training Programmes (OPTIONAL as RECOMMENDED)
    - 190-SKTD Sketches & Technical Diagrams (OPTIONAL)
  - 800-SWD Software Development
    - 810-DSGN System Design
      - 810.00 Overview - update of 130.02 (REQUIRED)
      - 810.01 System Requirements - update of 110-SRE (OPTIONAL)
      - 810.02 System Landscape (REQUIRED)
      - 810.03 System Data & Objects (REQUIRED)
      - 810.04 System UI (OPTIONAL RECOMMENDED)
      - 810.05a System Processes (REQUIRED)
      - 810.05b Application Software Organization (OPTIONAL)
      - 810.06 System API & Interfaces (OPTIONAL)

- 810.40 System Concepts in Detail - update of 120-CPTS (OPTIONAL)
- 810.45 Licensing Model - update of 130.04 (OPTIONAL)
- 810.46 Product Features - update of 130.01 (REQUIRED)
- 810.50 to 810.79 Appendices (OPTIONAL)
- 810.80 to 810.99 Design Sketches (OPTIONAL)

- 820-SINT System Internals
  - 820.01 System States (OPTIONAL)
  - 820.02 System Interfaces & BUS-es (OPTIONAL)
  - 820.03 Synchronization & Clocks (OPTIONAL)
  - 820.04 Algorithms & Strategies (OPTIONAL)
  - 820.90 Toolstack notes (OPTIONAL)
- 830-DEV Development (MANDATORY)
- 840-TEST System Test
  - 840.01 Test plan (REQUIRED)
  - 840.02 Test cases / scenarios (REQUIRED)
  - 840.04 Test Report (REQUIRED)
  - 840.90 Release Notes (REQUIRED)
- 880-RLSE System Release
  - 880.10 FEAT Product Features - update of 130.01 (REQUIRED)
  - 880.20 ELPRI Editions, Licensing & Pricing - update of 130.04 (REQUIRED)
  - 880.30 EUMA End User Manuals
    - 880.30 EUMA.QG End User Quick Guide (REQUIRED)
    - 880.30 EUMA.WPnn Work Procedure nn (OPTIONAL)
  - 880.30 ADMA Administration Manuals
    - 880.30 ADMA.SI System Installation & Configuration (REQUIRED)
    - 880.30 ADMA.SA System Administration (OPTIONAL)
  - 880.40 SKITs update - update all 130.nn (OPTIONAL)
  - 880.50 TKIT Training Kits & Programmes - update of 130.05 (OPTIONAL)
    - List of courses
    - Schedule
    - Courses curriculum
  - 880.60 SRVC Service - update of 130.05 (OPTIONAL)
  - 880.90 SCA Source Code Archives
- 890-MNT System Maintenance

- 900-OPS Operations
  - 920-TLE Prepare temporary live environments (REQUIRED)
  - 990-PMSP Project Management Support (REQUIRED)

Preliminaries

Here you'll find a recommended structure of 810-DSGN document containing ALL information that it can contain.

This structure is enough /comprehensive for most of the projects but it can bw extended in all situations where this brings more recant aspects in a project.

Structure of document

It follows the methodology phases as described in RENBLU document. For each section there is specified if is: MANDATORY, OPTIONAL or RECOMMENDED. Also, should be noted that those sections which have "a full code" (not just a simple number) are intended to be released as separated documents and here make reference to them.

Structure & content of deliverable

Its important to say that some sections are **mandatory** (and marked accordingly) and should **exists AS IS in all cases**, regardless other information or the same but in other structure presented.

100-ANA Analysis

110-SRE System Requirements (REQUIRED always)

Here are the requirements in most raw form, meaning "exactly as collected". It seems that sometimes is optional but:

- if requirements are already in a bid documentation - a reference to this document should be made
- if this is a new version with just some issues, too few and clear - even so, after some time should be not SO CLEAR as it was...

If section a better approach is to make some short notes following “MoSCoW" taxonomy, if possible or if no then let some margins for future hand wrote annotations for a reviewer.

120-CPTS System Concepts (REQUIRED in almost all cases)

Here should be at least a detailed description of system specific terms, objects, information domains, paradigms, environment (business or technical), similitude, different relevant taxonomies, etc, whatever is considered important to a good understanding of system.

This information will be BASIS for a future system design.

When is not necessary? So, in cases when it already exists and system intended modifications are nor of nature to alter these concepts.

130-SKIT Sales Kit(s)

130.01 Product Datasheet (REQUIRED always)

The datasheet is an element that describe the product itself, mostly from technical point of view. It is very common as engineering practice and its structure is not "something standard" but in practice the following information should appear here:

- what the product is good for
- technical specifications, mainly the standards which are compliant
- interfaces (if are), shortly and technically how to use them
- the product ecosystem: ie if there are more editions available, trainings available, etc

Basically a datasheet contains all that is considered useful for a person "at first look". Then, if really interested, will search for other detailed documents. This document is required being considered "*one of the first contacts with customer*". Many sites (search engines) are looking for `motors` section in HTML and what is considered as datasheet is one of the first elements to be displayed.

There are many examples (different forms and kind of products) of this document for RENNware products, not being a standard form (but changes with market trends) so here will not be included one.

130.02 Product Overview (REQUIRED if not present in *Datasheet*)

This section is mandatory if it is not present in datasheet (see previous section) and it should contain exactly what its name say, an overview of system / product. Many companies name this document as "flyer", "products brochure", and so on, but mainly is an overview of product ref *what is good for*.

130.03 SPIN documents (OPTIONAL)

This section is absolutely optional as not any developer knows what it is. It contains (if present) useful information for sales department to build cheat sheets ref problems that could be addressed at a customer in business context of this product.

Just for as curiosity, SPIN is a sales technique and the acronym is derived from Situation, Problems, Increase (problems) and Needs.

130.04 Licensing Editions & Pricing (OPTIONAL)

This section is optional if the system is "one edition - one license" model.

If there are more options for licensing or the product has been elaborated with "more editions in mind", so these should be written here as for sales / product manager references.

There is no need for elaborated or "sophisticated" text / content. Just enumerated options, what exactly would be the metric for licensing and some information useful to have an ideea of how could be established a price (if there are known). The document will be reviewed and (re)written by somebody else from product management area.

130.05 Service & Training Programmes (OPTIONAL as RECOMMENDED)

This section should be present in all cases as it describe what other services (installation, configuring, migration, etc) and training programs are available and how can be accessed / requested.

As in the licensing section, there is no need for "sophisticated" text. Anybody else would process latter this document but need to understand some basics and to have a future reference where to "come back" to check for missing ideas or useful things.

190-SKTD Sketches & Technical Diagrams (OPTIONAL)

This section(s) contains the diagrams and pictures used in other parts in their raw form un order to be latter easily converted in other formats needed.

800-SWD Software Development

810-DSGN System Design

The system design document is a collection of more chapters / sections / volumes, and these can be outlined in one or more documents, function of information volume. In that case please keep the convention of file names in order that other people to "recognize" a document first by its name.

810.00 Overview - update of *130.02* (REQUIRED)

Updates the overview of the system (`130.02` document). This update is "how things are seen by *system designer*". Anyway at least a reference should be made and a summary of them will bring "more clarity" for software development.

810.01 System Requirements - update of *110-SRE* (OPTIONAL)

This is the nature to make things more clear for system development by aligning (mapping) technical taxonomies to existing business ones.

Essentially this is an update of `110-SRE` document ref its requirements specification part "as seen by system designer".

810.02 System Landscape (REQUIRED)

The system landscape is anything else that different kinds of architectures, such as logical, functional, physical, interfaces, etc. Some of them are "the same thing" in most cases, but there are situations when small subtle information makes difference.

The basic idea is to depict the whole system in smaller parts un order to be manageable from all points of view.

Another important aspect is that the **architectures must be coherent between them**. They represent the same system from different perspectives and clearly **must exists a map between them**, not necessarily 1 to 1 (it would be impossible) but in any cardinality it must be and **must be a deterministic one**.

As architectures, at least a **logical** one should be, describing those logical components and their relationships.

Must be clear that all next sections will be *details* of the landscape and to avoid complexity you should **focus here on WHAT** system has to do and NOT on how system will do. For "how" there are next sections and references to them are clearly useful.

As architectures here are mentioned the most used and common ones:

- *logical* architecture - this depict the system logic for its coherent functionality
- *functional* architecture - in almost cases is just a breakdown of system functionalities
- *physical* architecture - describe the physical environment on which the components will operate (machines, devices, etc)
- *interfaces* - describe those system interfaces designed to be used by various other systems to communicate with this one

Important things in architectures design:

- start with an "overall" picture; this contains just the system as bubble and all its external environment, meaning data (generally named messages at this level)
- do not concentrate at first level for what is / will be inside the "system bubble", just on its external environment
- *name* each message outlined in system environment; associate some codes with these names to assure that is unique
- use a future dot (`.`) scheme for hierarchy and do not concentrate on ordering on messages here - there should be no relevant ordering at this level
- give a *direction* to these messages; direction should be as "seen" from system perspective and can be: IN, OUT, BIDIRECTIONAL
- each message should be triggered by "something" and at this level is important to mark the triggers; these could be: an event, an user action / manual, another system automate action, etc
- do not mix messages with their triggers; latter you will need to have them as separate
- when ready (must establish a "ready", latter you can and will come back and review these) **break the system bubble** in smaller parts / bubbles and repeat the same schema with messages
- at this level consider any sub-bubble as an independent system itself and treat it like did for first system bubble
- the whole process should be repeated for each new bubble
- when to stop with breaking down? the most frequent signal is when **you need to talk about how not only about WHAT**

810.03 System Data & Objects (REQUIRED)

For data objects there are more useful levels and taxonomies ref how and what to describe:

- **conceptual schema** - this level focus on list (inventory) of objects and their relationships. **THIS IS THE LEVEL WHERE YOU MUST START**. this model is. mostly known as "conceptual ER diagram"
- **behavioral schema** - this model focus on how data *react* at various events / triggers and how it transform depending on context

From other point of view (taxonomy) there could be:

- **physical schema** - that describe data from database server (whatever it would be) perspective; the biggest problem with this is the *model dependency of database server*; at this moment and in most cases we do not want to create *so hard* dependencies such a database server - our intention is to let these aspects as "large / open" as possible but continuing development
- **ORM schemes** - this kind of schemas describe data as *classes* (in their true sense); ***this is the most preferred model*** as being in almost cases agnostic to database server (but in a way depends of programming language); it should be able to model both the *conceptual* and *behavioral* aspects of data (lifecycle)

Here are some guidelines in designing system data (objects):

- use underscore (`_`) to prefix those attributes that are not necessarily / directly related to business; in most languages, variables beginning with `_` are private or protected by default
- prefix names with `_pk` for primary keys; this is preferred instead of `id` which could create unwanted dependencies being so large practiced
- prefix names with `_fk` for foreign keys; keep a "clean" name for business relations instead (usable only in ORM schemas)
- keep a largely supported, best know and with enough language libraries available ORM; ***SQLAlchemy*** is the most preferred, but could be others except those that limit usability strictly to one language and / or, directly / indirectly to one operating system for a real production deployment (which must be "something" stable and robust enough by definition)
- think relatively agnostic ref database server nature (SQL or noSQL); using ORMs you should be able to work with JSON formats in all cases without too much issues
- let *detailed diagrams* as much as possible for `820-SINT System Internals` document where are specialized sections for these; the other people expect find them there not here

810.04 System UI (OPTIONAL RECOMMENDED)

Things here should be "obviously", but here are some guidelines:

- keep a "clean" Ui interface, with simple and usual widgets
- be clear with user information ref widgets - for example for a "Browse..." button it should be clear what files are visible: from its local system or from server...
- think for usage of simple template frameworks, use a simple one (like *Bootstrap* or *UI Kit*) and try to avoid as much as possible to mix them, or assure they have good enough disseminated properties in order to be mixed (for example *UI Kit)*
- use any kind of mockups you think give enough clarity for (the other) developers in coding `HTML` , `CSS` , `JavaScript` , etc sections
- be *EXPLICITLY* in specifying all things; avoid to suppose that anyone should know defaults for a framework; even in this case make enough references and put links where defaults ca be found
- use (wherever is applicable) hierarchical diagrams to show "things"; they offer more "visibility" for auditors

810.05a System Processes (REQUIRED)

Here you must define the system processes and functionalities.

Start with a functionalities list. Organize it hierarchically having in mind a business criteria relevant for system informational domain. As a good check, any functionality is outlined here should be assured by at least one component from logical / functional architecture. So a *compliance matrix "functionality by component"* will be clearly very helpful. If there are functionalities assured by more than 1 logical component, instead a simple "x" in matrix cells, use more symbols, for example "s" for start here, "e" for ends here, "n" for notification, etc. DO NOT FORGET TO ANNOTATE MATRIX WITH A **LEGEND**. Even if seems that abbreviations are / should be very clear, there is not necessarily true. Remember basic rules:

- Do not be IMPLICIT, but EXPLICIT.
- Do not suppose anything; do not make assumptions

When the functionalities list is ready or for those which are clear, a (one for start) process should be outlined for it.

The processes are "coming" not only directly from functionalities but can be derived from other "places". Even so, these places will be addressed from some functionalities, maybe not necessarily as first level but from a functionality.

This functionality is not necessarily to be one directly "visible" for end users.

So, after outlining a process iy should be "decorated" with:

- a code and a human understandable name
- a description of what will do this process (ie what is good for)
- a list of what data will be required in this process
- the results will produce

Identically, all things should be **coherent and consistent**. Some compliance matrices up to architecture will be a very good check for designer but for developer also.

810.05b Application Software Organization (OPTIONAL)

Here are 2 aspects (at least) tha need to be addressed:

- toolstack used / proposed to be used
- application structure / organization

From the ***Toolstack*** perspective there must be said:

- the programming languages
- scripting languages
- operating systems required compatibility
- frameworks and libraries proposed

From the ***Application Structure / Organization*** perspective it is important:

- to design a simple directory structure; make use as much as possible of known practices, for example anybody will expect that in a directory named `doc` to find out some documentation
- for data as physical one, if applicable (not only / necessarily database files, even JSON files for example) think a `/data` directory directly from application root
- for data model think a `/data_models` directory directly from application root as it could be used by many system components
- always outline a `/test` directory
- always outline a `static` directory (preferable at application root level) for any kind of static files (style files, different data, etc) - doing so will be much easier to protect it on production system and to give "right security fio right persons
- in most situations a `libutils` directory is good; it can accommodate all project specific general purpose modules
- not as last case, have in mind that application is better to be put on an automated backup system, being this just a simple script, and who does that should have a simple and clear scheme of *what to backup*; also when ned to restore data from an old backup, people should be able to find the right places without making *"special analysis"* for that
- be consistent, avoid changing directories and file names as this could have a ***severe*** impact on code

For small projects it couldn't be necessary this section as probably "everything" are located in one to three directories.

810.06 System API & Interfaces (OPTIONAL)

Here is the section which describe the system **interfaces**. Of course these should come from (or go to update) *architecture*.

For each system interface should be outlined:

- an unique **code** (name)
- a **description** ref what does and what is good for
- the interface **protocol** often aka handshaking model
- the interface **API** in terms of methods, arguments, returns, usability, etc
- any other relevant applicable conditions or contexts

810.40 System Concepts in Detail - update of *120-CPTS* (OPTIONAL)

This section role is to update (clarify or add details) ref to concepts presented in `120-CTPS` document as result of different things new or changed along development.

Also, here could appear new concepts but, normally, this indicates a poor analysis process (or incomplete understanding of what system should do). Anyway, **new concepts must be clear and visible marked** and should be **notified** to project management area as they could be subject of change management.

810.45 Licensing Model - update of *130.04* (OPTIONAL)

This is a dedicated section where licensing model can be explained / detailed. Usual here are information regarding volumetric usable for licensing (as system is aware of and can count them) additionally to `130.04` document. The information provided is used by product manager and commercial departments to derive final *product licensing models*.

Essentially the information provided here will update `130.04 Licensing Editions & Pricing` document, as this kind of information become clear along development process.

Guidelines and licensing metrics examples:

- keep a dedicated section in `130.04` document this being too technical for commercial use; therefore will be easier to clean commercial information
- metric ex: named users - the registered enabled users in system
- metric ex: size of uploaded files
- metric ex: number of days for subscriptions
- metric ex: number of tokens / certificates given
- metric ex: number of days to allow downloading some information
- metric combinations

810.46 Product Features - update of *130.01* (REQUIRED)

This is a dedicated section where product features can be explained more clear and detailed. Therefore is an update to `130.01 Datasheet` document.

For smaller projects only `130.02 Overview` will be updated, but if 130.01 exists, this should be updated, the other (130.02) being a subset of it.

Anyway, this information is also marked (as more raw form) in CHANGELOG, RELNOTE, README files.

These information will definitely contribute to `130.01 Product Datasheet` and `130.02 Product Overview` documents, as being known that it becomes more clear and defined in development process.

810.50 to 810.79 Appendices (OPTIONAL)

This sections reserves numbers / codes starting with `50` up to `79` for different appendices. Normally, appendices are used:

- if is more useful to separate the information as being large in most cases, and put it on a separate file
- the information does not need attention when reading all document, but is containing *details* that are useful for "advanced readings" or a deeper understanding of information domain in subject

810.80 to 810.99 Design Sketches (OPTIONAL)

This sections reserves numbers / codes starting with `50` up to `79` for different appendices. Design sketches are sometimes (often) very useful to be as separated files in their raw formats in order to be used latter in other materials (for example by marketing / branding departments).

Pay attention that these sketches could be duplicates of `190-SKTD Sketches & Technical Diagrams`, so be aware and try avoid as much as possible duplicates.

Remarks to 810 DSGN section

Sometimes (especially in case of small / med projects) it is easier or required to have one single file - in this case just name the file using `ALL` instead of a volume / section code and "put all together" in that file as "volumes" / sections

820-SINT System Internals

These are very technical and specialized sections ref system (internals). They can be simple references to *notorious / known algorithms* or detailed technical notes about *how is implemented*, for example (and most often) for states.

820.01 System States (OPTIONAL)

Here are described various **system states** used in development process. *This section applies to systems where there are states*. Preferred forms of diagramming follow UML notation standards (they are not excluding one each other):

- ***state diagrams*** (ALWAYS **REQUIRED**) - which relieves all the states (in subject) and *transitions* between them
- ***sequence diagrams*** (OPTIONAL) - which add the *temporal* information to state diagrams

820.02 System Interfaces & BUS-es (OPTIONAL)

Here should be described each system interface (following the guidelines from section *810-DSGN System Design 06 - System API & Interfaces*). The `BUS` term means in this document almost the same thing, but is just a more hardware term. It is quoted here because it is so common that, often, this term will be found.

820.03 Synchronization & Clocks (OPTIONAL)

Here is clearly a very specialized section regarding *clocks*. This is in almost cases REQUIRED if the system is (or part of) a distributed one, where clock synchronization is "something" required.

If present, it should be present both **logical** and **physical** clocks, how are they implemented, if use some "known" algorithms (and a reference to them), sync issues, context conditions for good operations, etc.

820.04 Algorithms & Strategies (OPTIONAL)

This section is "self descriptive" as name. Clearly, if there are *known algorithms*, they shouldn't be described here in detail, but a few words (ref how and why are used) must be written and a reference made (or internet, ie Wikipedia, or book).

820.90 Toolstack notes (OPTIONAL)

Here should be various notes relevant for toolstack and "outside" normal things:

- special remarks, ie how to, which parts / components, etc
- where a framework, library, etc can be found
- warnings and other known issues (the designer should know them or the most important of them)

830-DEV Development (MANDATORY)

This is not a document section. In this folder it is supposed to keep all system code and it is organized as project require.

840-TEST System Test

840.01 Test plan (REQUIRED)

Here it will reside the plan for testing. Normally this should be as simple as possible, being "just for your information" at development level. What is really important (especially for own time planning perspective) is what in Project Management discipline is named "***WHO - DOES -WHAT - WHEN***".

So a simply "agenda like" will be enough. Not details, not why, not specificities, etc. The WHAT part should be exactly what respective person wil have to do, in fact indicating a *test case number / id*.

Also it will be very good to specify the nature of test:

- "white" test - ie with "entry" to code level
- "black" test - aka functional testing, just "looking & exercising" from an end user perspective

Most (almost all) tests of here will be in "black" category, as long as code-in testing is the developer's responsibility for "alpha" releases testing.

Of course there are many other test that should be done (integration, acceptance) but they are not in scope of this methodology, but more in Project Management one.

840.02 Test cases / scenarios (REQUIRED)

A test scenario should be outlined for every system functionality (see *810-DSGN System Design* section *05a - System Processes*). "Test cases" is a list with all test scenarios, ie the table of content or index of them. The difference is so subtle that in most cases they are uses as synonyms for the same thing: test scenario.

Every test scenario should contain:

- an **unique code** / number / identifier so will give us the possibility to reference it latter
- a brief **description** of what the functionality under test should (is expected) do
- the (ordered) *NORMAL* **steps** that should be done for functionality to fulfill it; pay attention, these steps should reflect the end user manual or help or working procedure as they should be here exactly how an end user would do
- **expected results** (for a *PASS*) the system must "produce" - here a good practice if these are the nature of "real numbers" or "too fine grained dates, as timestamps" is to specify the accepted tolerances (if they are and if appear in documentation)
- any **condition** that must be met / fulfilled by system as in documentation

840.04 Test Report (REQUIRED)

The test report should be a mix between *test plan*, *test cases* and *test scenarios*. In essence this will be a **list** with all **tests** (test cases), **who** executed the test (test plan) for further questions and **result as PASS / FAIL** (test scenarios).

Guidelines:

- keep it simple
- it will be just an input for *Release Notes*; not subject for any organizational regulations
- no details (as reasons, motivational facts, etc)
- just PASS / FAIL; the customer is not interested about why and if, then are other persons who should address that issues

840.90 Release Notes (REQUIRED)

This deliverable result as a "humanize" compilation (clean, choose words, etc) of *820-SINT System Internals* section *04 Test Report* and it will ***accompany the released packaged*** as required part of it.

880-RLSE System Release

All these parts / sections aim to have a **PACKAGE** (ie, required for a system *product*) that accompany a release. They will (finally) be "what a client see" ref a system update / upgrade package. **It will be "its primary / main source of information" in deciding that will get this package or not**.

Being deliverables with commercial impact they have a *literal code* not only a numerical one. This code is also unique so it can be used as Alternate Key.

880.10 FEAT Product Features - update of *130.01* (REQUIRED)

It will be get / obtained by a (re)compilation (following mainly a commercial facelift) of *840-TEST System Test* section *90 Release Notes*.

Basically in (after) this step the `130.01 Product Datasheet` document should be updated, or to create a new one based on it. Idea is that here, this document will **BE A FULLY COMMERCIAL** one.

The same mentions (please read the previous sections) apply here ref to *130.01* & *130.02* documents. If you're here probably the size / complexity of the system is of nature that require for `130.01 Product Datasheet`.

880.20 ELPRI Editions, Licensing & Pricing - update of *130.04* (REQUIRED)

This should be the "commercial form" (as more elaborated and with a commercial facelift) of *130-SKIT Sales Kit(s)* section *130.04-SKIT Licensing Editions Pricing*. Essentially this is the update and "commercial facelift" of `130.04 Licensing Editions & Pricing` document.

The same mentions (please read the previous sections) apply here ref to *130.01* & *130.02* documents. If you're here probably the size / complexity of the system is of nature that require for `130.01 Product Datasheet`.

880.30 EUMA End User Manuals

Here are the manuals that will mainly address *end users* (business area) of product. These are usual of two kinds:

- **full** / complete manuals - these are "monolithic" documents containing all information needed for end users, doesn't matter what kind of end user is reading it (a very common example is "*Quick Reference Guide*")
- **partial** manuals - these are addressed to a more target segment of end users; one manual for a category of end users (typical example are "*Work Procedure*" documents)

So, regardless the enumerated type, our recommendation is to specify the TARGET AUDIENCE for each manual as being a very useful practice.

880.30 EUMA.QG END USER QUICK GUIDE (REQUIRED)

This kind of manual is intended as a very quick reference where users will find easily what they need or where to look for details. It is a ALL IN ONE manual style containing "nothing about all" as style of information.

As guidelines, it should contain:

- a brief overview referring functional aspects, not technical ones
- a quick installation procedure or "how to make it usable first time" - no details, no options; just a simple, default installation
- a brief tutorial ref "how to use it first time" or "first steps"

880.30 EUMA.WP*NN* WORK PROCEDURE *NN* (OPTIONAL)

As general rule a "work procedure" is a description of how to do a specific, concrete and atomic activity.

> **Specific** means that activity is specific to business that system is implementing
>
> **Concrete** means that activity is not a generic one, but a very concrete, one of the known activity for targeted audience
>
> **Atomic** means the respective activity is not normally breakable in smaller parts with the risk of inconsistencies and normally is executed in full up to finish, ie, technically speaking not let open states(!)

As good and best practice, the work procedures package should be accompanied by a *catalog* document which is an index with all work procedures, a table at least code and name foe convenience ordered by procedure name.

Procedure content guidelines:

- **code** the activity and mark this code; remark that usually is a number, those *nn* from document title; this code is subject of project conventions
- **name** the activity just procedured as it is known in business; try to keep a "human name"
- specify the intended **audience** as business roles (*attn not system, but business role if they are not the same*)
- specify the **navigation path** to achieve this activity in system
- specify all **steps** that must be followed to successfully fulfill activity
- for steps give an aid using **screen captures** and annotate them

- give indications ref every **field** that appear on that step; be clear, make references where from that information is coming, etc
- do not suppose anything; just explain everything even it seems simple or annoying
- **number steps** as can be referred easier
- for each step give **hints** ref expected results and system behavior (surprises are not something good at job), identically even it seems obvious for you

As mandatory work procedures (applicable for any system) are:

- ***System navigation*** - this contains specific instructions ref how access a functionality in system, usual the menu path and other relevant info
- ***Authentication in system*** - explain how to login, logout, change user, change password, etc
- ***System indicators*** - explain what are good for and how to use all system icons, widgets iconic butons, status indicators, etc - if it is small enough this procedure is included in navigation

Also, keep in mind that many users will print these procedures, so keep a standard paper format and try to include them in **system help** module.

> ***Where to find out what are those activities that need to have a work procedure?*** Remember, there is a document with system processes (*810.05a System Processes*) - this is the place where to start.

880.30 ADMA Administration Manuals

Here are the manuals that will mainly address *administrators* (IT area) of product. These kind of manuals should address at least the following content:

- current system maintenance
- installation and first configuration
- initial system credentials
- some disclaimers regarding initial credentials clean
- how to make some safety copies

> These manuals should not contain detailed information (ar advanced information) regarding administration as this is usually subject of payment. The same thing apply for service(ing) procedures / diagrams and so on.

880.30 ADMA.SI SYSTEM INSTALLATION & CONFIGURATION (REQUIRED)

This manual is about:

- how to install the system in various deployment scenarios
- deployment scenarios that system is prepared for
- how to change initial sensitive data
- basic system configuration (make it to run in in its simplest mode
- more ref advanced configuration parameters

880.30 ADMA.SA SYSTEM ADMINISTRATION (OPTIONAL)

This manual is about:

- current maintenance operations
- backup the system and system data (if applicable)
- enable / disable users
- enable / disable other system objects
- starting / stopping different interface / communication / messaging components
- clone system for testing purposes

Please be aware to general guidelines regarding administration manials.

880.40 SKITs update - update all *130.nn* (OPTIONAL)

This process, is anything else that **ALL SALES KITS must be reviewed and updated** where applicable. The process is here just a checklist / remember point. Basically all *103.nn* documents should be REVIEWED and give the required "commercial facelift".

880.50 TKIT Training Kits & Programmes - update of *130.05* (OPTIONAL)

Ref "Training Programmes" some basic things are required:

- a list of courses
- a schedule for these courses
- a curriculum for each course

In next sections, some recommendations and guidelines can be found.

LIST OF COURSES

This is "just a table" with all courses relevant for the system. It will be helpful, often required to specify some taxonomies for courses, so:

- a course ***code*** to allow latter references
- ***category of auditorium*** as: beginners, advanced, etc
- type of available ***materials*** as: video, powerpoint, pdf, none, etc
- ***average duration*** as: under 1h, 1 - 3h, over 3h, etc
- some prerequisites: course A, knowledge of, etc
- ***target auditorium*** as: administrators, end users, etc
- what will be the benefit after this course, ie "***what you'll learn***"

SCHEDULE

- Schedule should be a relative on, so no exact calendar dates here, but *periods*, *duration*, a *min*, *max*, *recommended* number of participants.

- at least an indication of *lectors* or quality of project persons intended to keep that course (ie, analyst, sw engineer, etc)

COURSES CURRICULUM

This should follow a "standard" but simple curse curricula. In its simplest (but very relevant) is a "**COURSE OBJECTIVES**" type of document.

These items will be put in a commercial form (update `130.05 Service & Training Programs`).

880.60 SRVC Service - update of *130.05* (OPTIONAL)

This section refer to services that can be done for that system in production. This is just a list and **not** HOW WILL BE DONE but **WHAT CAN BE DONE** and is intended to update `130.05 Service and Training Programmes`.

Guidelines of list can contain:

- a customer specific maintenance plan, based on its infrastructure
- a med / long term update and support plan
- assurance of extended warranty

880.90 SCA Source Code Archives

Here are just (normally as archives) the tags saved from git repository with aim of "backup copies". No other processing is required if they are saved AS EXPORTED from git in a standard archives format from: *zip*, *tar.gz*, *tar.bz2*.

890-MNT System Maintenance

This section address those things related to system maintenance as it would be in PRODUCTION, ie "*what should be done to keep system alive after migrate*" it in production environment. It is important to say that, here, things are from software development perspective more exactly, here are those things that results as concrete facts in / after development, what exactly should be done and when to have a "clean - always ready to run & usable" system.

These are some guidelines with what this section should contain:

- ref **Maintenance Plan** - a frequency of operations should be specified or if exists a proposed one, to be revised and reviewed
- **type of incident**s that could appear and their **severity** levels - normally type of incidents are almost known but their *impact* should be described very concrete with potential consequences (ref data loss, data damage, unrecoverable facts, etc)
- the **response** & **resolve** times - these should be revisited and reviewed
- few words about **Hot Fixes**, **Security Updates** & **Critical Patches** - what exactly they means for this system, if and how are applicable, etc

900-OPS Operations

This section outlines activities and deliverables that are not necessarily / totally related to engineering of software development, but **directly related** and at its boundary with other company operations (mainly with *Project Management*). They are designed as helpers in software development with more content specific to software "manipulation".

920-TLE Prepare temporary live environments (REQUIRED)

ATTN: new in SDEVEN methodology

This activity is intended to support the software development process and activity. It is described (as informational content) in *90 RENBLU* document and in other methodology places as targeted systems (ie what need to be prepared).

Here it is expected to find a list with all these systems and, if needed, some more specific and detailed information, for example ref operating system, external environment where from system need to be accessed, users, etc.

990-PMSP Project Management Support (REQUIRED)

ATTN: new in SDEVEN methodology

Here, the software engineer should depict some elements useful in this project management operations:

- a list of deliverables that is expected to be issued in 800 Software Development phase. This list should be **concrete** and **clear** as being addressed to a non-software-practicing person as its normal job.
- some useful quality factors which will support for deliverables measuring and ***PoC*** calculation
- a list with quantify the resources need for each deliverable
- a duration for each deliverable
- any other relevant things resulted from a discussion with the project manager or technical project manager, such as: discussion minutes, various action plans, resource contacts, etc

---

Last update: August 5, 2023