SDEVEN Software Development & Engineering Methodology

Version: 7.0.14
Release date: 230813

Licenses and Products (SDEVEN.70-LIP)

Table of Content

This procedure refers to software licenses and products, how are used and what are good for.

Procedure present:

- what kind of licenses are practiced and
- how a software system becomes a product (or when can be considered product).

> This procedure is not mandatory for software development process but explain how a "program" becomes a "product".
>
> Also ref licensing aspect, this is clearly a commercial and legal one, but there are situations where licensing needs some *measurable metrics* which can be identified only in strong correlation with software as design and development.

Software licenses universe

Without any other clauses a software license is about the *legal rights to use* the respective product / system.

A software system can download, copied, installed, removed, etc in most cases free (without any payment required). But that's all. From legal perspective **you cannot use it (for any purpose) without a license**. And sometimes even the installation can be considered "out of law".

Another important fact that must be understood is that a **license DO NOT transfer you the intellectual property of the software code, even of the software product**, You cannot treat it as your own property, You CAN ONLY USE IT *for your own and in your name*. Also you can make safety copies, can backup it and restore it.

From legal perspective you **cannot use the software** without permission (given by license), Even if system does not enforce any restriction.

Software licenses

There are 3 kind of license models (types) used by company for software products:

`open licenses`

these are *free* (ie, there is no cost to pay for them) - for these licenses the company preserve the *intellectual property and copyright* and offer for free the rights of use - any other services *are not default covered by this license - cannot sale or resale* the system / product

`turn key licenses`

these are system products *made especially* for a customer, paid by him and the *intellectual property is transferred* to the customer - after finishing the system, *company has no right to sell / resell or use the code or parts of code AS IT WAS TRANSFERRED* - however, in most cases, two "evidence" `read only CDs` are made having a reference of the code for which the intellectual property is transferred

`commercial licenses`

these are strictly with *payment for usage* - quantification of payment is be made in various forms (and unit of measures), for example number of users, number of computers, number of processors, quantity of memory, and so on - the software can restrict usage (but is not mandatory, being a design specification) by "forcing" in a way these quantities, of course first thing being the software ability to "count" for them

IMPORTANT notice

If not otherwise specified, open licenses should be accompanied with text: "**This software is a copyright of company Systems (REN CONSULTING SOFT ACTIVITY SRL).**". Text should be put at start of license content as to not alter its original text which is usually published and can be referred AS IS.

System vs Product

Software targets of development cycle

For a software there are 3 major *targets of development cycle*:

- to become a product
- to become a system
- to become both.

As **product** must have an *usage documentation* (ie, work procedures), an *administration documentation* (ie, installation, configuration, maintenance), a *packaging* procedure.

As **system**, a software should have:

- an *installation procedure*, which could be just a documentation or other automation software. This procedure should be *clear*, *well defined* (ie, deterministic, without ambiguities) and *repeatable*
- a *logging mechanism* and some rotate policies. Using host operating system standards is recommend in order to be easy maintained by any system administrator

As **system product** inherit the requirements from both categories.

There are also other "things" that must be satisfied, especially from commercial point of view such as:

- a logo would be required for a product
- presentation materials, presentation views (ie, slides), a presentation site, some hints for sales and bid teams (ie, `130-SKIT` elements)

This methodology assures that the essential parts of both taxonomies will be covered, at least in raw forms creating the base for future / next refinement levels.

Software products / models

From this point of view relevant taxonomies are:

- by completeness
- by code level

By completeness taxonomy

- **full standalone** - products that contains everything to assure a complete functionality (aka full stack products, or in jargon "with batteries included", all in one, etc)
- **modules** - products that assure a single functionality usually useless only itself, but normally used in a large context, combined with other modules; examples: a database JSON transformer, a caching system, a queuing systems, etc
- **frameworks** - products aimed to be used as foundation to build other products over it; examples: python Flask, company CORE, etc
- **interfaces** - products aimed to "stay in front" of other systems / products and therefore assuring different kind of protection, translation, etc; often known as middleware products; examples: data APIs, proxies, guards, data translators, SQL Alchemy, etc

By code level taxonomy

- **low level / infrastructure** - these are systems that address low level operations, with intensive (ie, directly coded) use of operating system directives; examples: print utilities, file system watchers, system monitors, serializer, de-serializer, en(de)coders, etc
- **mid and high level** - these are systems that do not address directly operating system directives (just in rare cases for usual file operation), usually addressed to business or just to assets inventory (infrastructure systems for example); examples: ERPs, invoice makers, etc
- **UI / meta** - these are systems that assure some features for user interface (operations) by using different flavours of (tagging) languages specific to a device (for example VT100 terminals), to a software (for example HTML for browsers); sometimes these systems use "real" languages with empowerment of complex programming languages (for example JavaScript) or just simple "stylers" to assure a better readability (CSS is a good example, Markdown and PostScript are others, etc)

Last update: August 13, 2023