SDEVEN Software Development & Engineering Methodology

Version: 7.0.13
Release date: 230810

System Software Testing (SDEVEN.25-SYTEST)

Table of Content

- System Software Testing (SDEVEN.25-SYTEST)
  - Preliminaries
    - Why
    - When
    - Vocabulary
    - Test types
  - Testing and working environments
    - Environments and Information flow
    - Development environment
    - Test environment
    - QA environment
    - Production environment
  - Test deliverables
    - Test Plan
    - Test Scenarios
    - System hardware requirements
    - Proof of Concept

Preliminaries

The testing is one of the most important activity in software development as long as a piece of software is NOT written for own purposes.

Why

The testing may *assure* you that a piece of software *do what was intended to do*. This is "one face" but testing must also assure the *owner* of the software for the same things. And finally must assure the *end users* (generally the customer) for same things, sometimes more things.

When

The testing should be done (conducted) first "internally" (ie, not in the presence of customer's people). Then some more complex, elaborated tests should be done in the customer presence (for customer confidence).

Vocabulary

The testing process will involve some specific terms and concepts like: *compliance*, *bug*, *acceptable*, *workaround solution*, ... These terms are not necessarily new terms but they will make more sense, will get a more clear meaning "if are seen" from testing perspective.

Test types

In testing process more test types will be conducted. The *type of tests* can be seen from more perspectives, but those that relevant in this context are:

- access to code perspective:
  - *white box tests* - apply when code is known and can be accessed and test that code by expecting some behavior in known conditions
  - *black box tests* - apply without knowing the code but expecting some results for some given input - because they address functionalities these are called **functional tests**
- scope / range of code impact perspective:
  - *unit, unitary* tests - these tests apply on small code blocks (for example a function or a procedure)
  - *integration* tests - these test address more complex portions of code and are mainly "looking for" their good working when interact one with others - in most cases these test are simply known as **acceptance tests**
- performance and compliance perspective:
  - *standards conformity* - these ones seek to demonstrate the system conformity to some standards or practices
  - *performance* - these ones seek to demonstrate the system performance in some given "stress" conditions, also from these tests results what is known as **System hardware requirements**

More details regarding test types is out this document scope. They are fundamental learning units in software engineering theory. The reason why some of them were listed here is to be aware that:

- they are used and applied in current job operations
- the other members of the team expect you to know what are them about

Testing and working environments

The basic assumption of testing theory is: **the final produced system must be able run on different machines** than those where it was produced.

Using more than one environment is a *must* because:

- anyway you use at least two environments, the one **where you develop the software** and another one **where the system application will be installed** to be used
- these two environments are *not guaranteed to be identical* and the one where the system application will be installed you even do *not know "how it looks like"* - the only thing you can do is to make some recommendations but that's all
- so, at least an environment where you'll test the system is absolutely necessary and this should be different that the one where you developed (or still developing) the system

Environments and Information flow

Production environment

Production environment is a **real & live** environment, where our customers work and operate their current business. IT IS COMPLETELY **FORBIDDEN TO ACCESS** THAT ENVIRONMENT FOR NOT AUTHORIZED PERSONNEL.

The next diagram shows the most basic flow of testing without extending it after delivery of product.

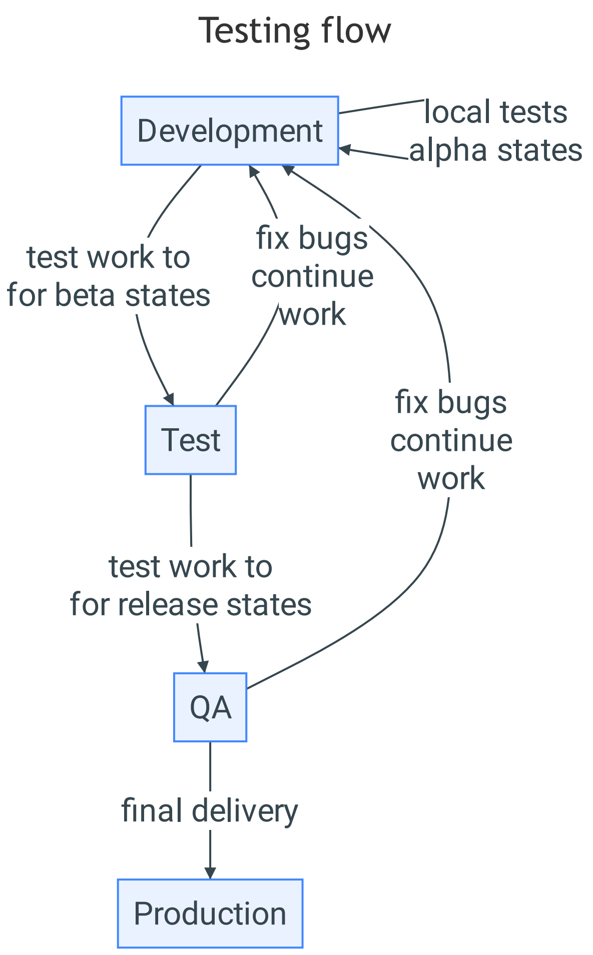


Diagram reveal the following environments:

- **development** aka **dev**
- **test**
- **qa**
- **production** aka **prod**

test & qa combined

In some projects `test` and `qa` environments are combined in a single one usually called `qa-test` or simply `test`, environment who takes on the role of both.

Each environment will be treated in details in next sections.

Development environment

The *development environment* means *all systems and tools you use to develop the software system* (application, product, etc). These could be on more than one device (for example use a phone or tablet to edit some files, a git repository to store them and a laptop to make some compilation, a git client to manipulate its data, a IDE tool to edit code, a compiler to compile code, etc) the idea being that development environment does not means necessarily one device.

Could be situation when some simple devices are not enough to finalize a development step and a more powerful machine, a server is needed in that process. In this situations, dedicated servers are used for development and they are called *development servers*.

The development environment is very *tight and dedicated* to a project and is not recommend to be reused from one project to another. Development environment is also very specific to a person, each developer having his affinity, preferences and productivity by using different tools, and *AS LONG AS THIS DOES NOT CREATE INTERFACE PROBLEMS* with the other team members or *LICENSING ISSUES*, its perfectly to use them (this is frequently happen for code IDEs and editors).

> So the development environment life is limited to one project or even only to a phase of a project. Development environment can contain all things that developer (or the team if use a development server) consider necessary to use. Especially when using development servers *it is very useful if the development language / framework allow for some instruments to isolate environments* and clearly the should be used (examples are: `Poetry` or `venv` for Pyrhon, `composer` for PHP and Laravel, `cargo` for Rust, etc).

Resulted version quality

Software versions resulted from development environment ***cannot be "graded" more than*** `alpha`.

Test environment

The *test environment* has the role to test the system on other completely different environment than the one in which development was made.

Doing so, any software components, libraries, code parts, text files characteristics, date or time stamps, user environment data, operating system configuration, or *other kind of system particular configuration* **WILL BE DETECTED** by making this kind of testing. Remember the basic objective of testing process: "*the final produced system must be able run on different machines*".

The ideal `test` environment is obtained by cloning an existing `production` environment and if necessary (in case the production machine is a "huge resources" one) make **only "quantitative"** adjustments, not qualitative ones (ie, downsize not downgrade).

> The test environment is MANDATORY to be limited to one project and one test phase. Other test phases will need another test environment. (The test process can alter enough the environment so other tests to be irrelevant).

Resulted version quality

Software versions resulted from test environment are ***usually "graded" as*** `beta`. But this depends more on type of tests conducted, ie, integration, functional, acceptance, etc.

Testing team

Testing conducted in `test` are executed by and in **presence of producer team**. This is done exclusively in all cases is possible this not being a matter of confidence but a matter of fucus on "doing what you have to do and only this and now !" - see also the section ref `qa` environment.

QA environment

The *qa environment* is absolutely identical with `test` environment and all things from `test` must be applied for `qa`. The only difference is regarding the presence of customer team.

Testing team

Testing conducted in `qa` are executed in **presence of customer team** and this is mandatory. For particular / producer only tests, see the section ref `test` environment.

Production environment

The *production environment* is the place where the customer business reflected by the (through) system is happening. Live, real, with real data and critical as functioning (at least from the provider perspective).

Production environment

Production environment is a **real & live** environment, where our customers work and operate their current business. IT IS COMPLETELY **FORBIDDEN TO** ACCESS THAT ENVIRONMENT FOR NOT AUTHORIZED PERSONNEL.

There are no more things to say about production environment except the warning, production environment should not be **accessed, modified, queried, etc, generally no operation**.

Any intervention required in production environment must be done **ONLY BY AUTHORIZED PERSONNEL AND ONLY WITH CUSTOMER WRITTEN PERMISSION.** Credentials for any component from production environment are subject of **customer strict confidential data and "secrets"**. The customer must be instructed **to change all credentials** used in environment setup phase.

All other operations regarding production environment (for example backup or update) are ONLY *customer responsibility*.

Any **copies of production environment** can be made ONLY by customer authorized personnel and obtained ONLY with customer representative consent.

Accepted version grades

In `production` environment only **`release`** graded versions are allowed to be installed. Only as exceptions and:

- for ***critical business*** reasons
- from trusted sources versions `beta` graded will be allowed

Test deliverables

In order to be consistently applied and to be a proof of functioning, the testing process includes a series of *activities* and a set of *deliverables* that will be explained in next sections.

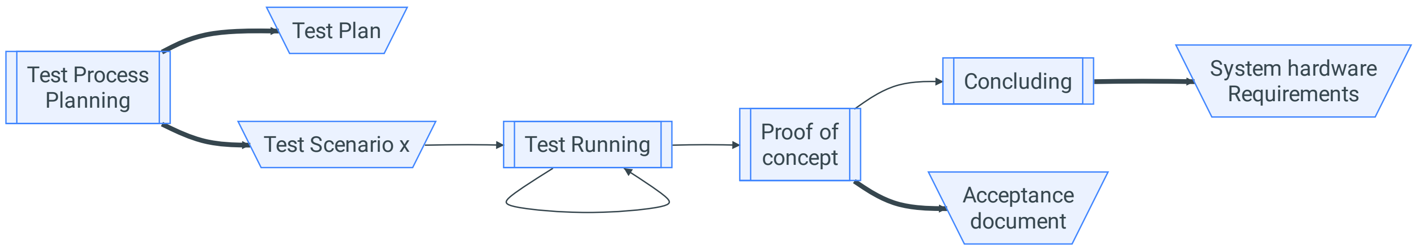
Deliverables that must be created are:

- **Test Plan**
- **Test Scenarios**
- **System hardware requirements**
- **Proof of Concept** document (aka **PoC** or **Acceptance** document)

All these deliverables must be formally agreed by customer.

The following diagram summarizes the testing process.

Testing process and Deliverables



The following sections will discuss each deliverable focusing on its content and purpose. Those aspects that are not always in the sphere of *perfect (with zero deviation)* but have a level of approximation and tolerance that *must be kept in a zone of comfort, trust and functional acceptability to not alter business operations*.

Test Plan

The test plan is mainly a planning of test scenarios:

- a summary list with all scenarios expected to be executed
- a time frame in which each scenario execution will take place
- a general objective (*plan objective*) that establish the *goal* of test plan execution, more exactly what **acceptance** type is targeted

- for each scenario which functionality(ies) will be demonstrated (not detailed because these are written in scenario test)
- required team for each scenario (at "mandatory / optional" level)
- who approve the scenario resolution (pass or fail)
- a pre-requisites list and different other requirements, IT, logistics, rooms, etc
- who will execute each test scenario

The test plan should be considered a contractual and an official document, so its change is subject of *change control procedure*.

An electronic test plan template can be found here or use document `Appendix_F1_TestPlan_template`.

Test Scenarios

The test scenario is a *form of verification* of a punctual, concrete functionality, which is *completely defined*, by this understanding that its finality is known exactly.

Through the **test scenario, we aim to achieve a desired result for a series of known conditions**. In other words, for a known set of input data, it is verified if the results are the expected ones. Thus, for each aspect that needs to be checked, a test scenario will have to be created.

A test scenario must have (and guarantee) some *qualities* (properties, characteristics) which gives confidence to the person who decides whether the test result is *CONFORM (PASS)* or *NON-CONFORM (FAIL)*. This can be done by:

- establishing what means *acceptable tolerance* when comparing obtained result with the expected one - this tolerance should be quantified in any rational-measurable way
- if test scenario has *more than two or three particular cases* (ie, if-cases), cases leading to results of different natures, then the test scenario must be divided, separated in more test scenario one for each expected result nature
- tested case should be as *smaller as possible but enough relevant* for customer, more precisely, to avoid falling into the trivial, irrelevant, useless

Well defined test scenario

The scenario must have a series of small steps each one being clear ref:

- what and how should be operated in system
- what is the expected system behavior
- what are the expected results

Each test scenario must be closed with a clear resolution **PASS** or **FAIL**. In case of fail some short recommendations or explanation ref what should be done (these are subject to future ***bug fix issues***).

An electronic test scenario template can be found here or use document `Appendix_F2_TestScen_template`.

System hardware requirements

This deliverable must show the necessary hardware requirements for the system (product, application) to run in *normal operating conditions* and this is tight correlated with concept of ***system load***.

To be able to estimate and to acceptable fulfill this requirement, some *volume metrics* should be identified, but these ones **must be relevant for system** regarding *loading situations*. *Loading* is clearly a pure technical aspect and should be FIRST established and defined (as system / application relevant metrics and units of measures) by system architects and designers.

"Conversion" to hardware resources required to allow execution under those loading conditions is another pure technical thing that should be defined (in terms of equivalence) by infrastructure specialized people.

Normally *System hardware requirements* deliverable should present the **minimum** requirements the system to run. Is optional (but recommended) to present also the requirements for an **optimal** system run, or to offer a "way" to determine how to calculate them when loading conditions are changing.

Level of details in *System hardware requirements* deliverable should be minimal regarding resource types (as these change very rapidly in time...) but enough for a customer to be able to determine what to buy (or to make available) in a reasonable way (for example regarding storage do not idicate the type of disks but only required capacity as operating system can directly access, or for processing capacity do not indicate the CPU type but indicate the number of unit of processing units, some necessary features like hardware virtualization, and so on).

Proof of Concept

This deliverable is the **acceptance agreement** *as a formal confirmation of test objectives* (ie, from *Test plan* document).

Referred documents

- If the *test plan* was "well done" and formally agreed the it just will be referred in this document.
- Also, is recommended that *test scenarios* (resulted after their execution) to be referred

Finally must remember that this document will become **part of contract** and will be the *fundament of future financial* documents (for example *invoice*) and operations, so it must **respect all legal requirements** stated contractual agreement between parts (customer and supplier). At least a brief review from legal perspective of this document is strongly recommended.

The concrete form of this document is subject of contractual terms and cannot be generalized here.

Last update: August 13, 2023