Test Principles and Guidelines

V1.0

22/04/2022

Table of Contents

[Principles 1](#_Toc101539491)

[1. Testing shows the presence of defects 1](#_Toc101539492)

[2. Exhaustive testing is not possible 1](#_Toc101539493)

[3. Early testing 1](#_Toc101539494)

[4. Defect clustering 1](#_Toc101539495)

[5. Pesticide paradox 1](#_Toc101539496)

[6. Testing is context-dependent 1](#_Toc101539497)

[7. Absence of errors fallacy 2](#_Toc101539498)

[Guidelines 2](#_Toc101539499)

[1. Development team should avoid testing the software 2](#_Toc101539500)

[2. Software can never be 100% bug-free 2](#_Toc101539501)

[3. Start as early as possible 2](#_Toc101539502)

[4. Prioritize sections 2](#_Toc101539503)

[5. The time available is limited 2](#_Toc101539504)

[6. Testing must be done with unexpected and negative inputs: 2](#_Toc101539505)

[6. Inspecting test results properly 3](#_Toc101539506)

[7. Validating assumptions 3](#_Toc101539507)

The following testing principles guidelines should be followed while testing the software.

# Principles

## 1. Testing shows the presence of defects

The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn’t talk about the absence of defects. Software testing can ensure that defects are present but it can not prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects

## 2. Exhaustive testing is not possible

It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.

## 3. Early testing

To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

## 4. Defect clustering

In a project, a small number of modules can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.

## 5. Pesticide paradox

Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

## 6. Testing is context-dependent

The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

## 7. Absence of errors fallacy

If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

# Guidelines

## 1. Development team should avoid testing the software

Testing should always be performed by the testing team. The developer team should never test the software themselves. This is because after spending several hours building the software, it might unconsciously become too proprietorial and that might prevent seeing any flaws in the system. The testers should have a destructive approach towards the product. Developers can perform unit testing and integration testing but software testing should be done by the testing team.

## 2. Software can never be 100% bug-free

Testing can never prove the software to 100% bug-free. In other words, there is no way to prove that the software is free of errors even after making a number of test cases.

## 3. Start as early as possible

Testing should always starts parallelly alongside the requirement analysis process. This is crucial in order to avoid the problem of defect migration. It is important to determine the test objects and scope as early as possible.

## 4. Prioritize sections

If there are certain critical sections, then it should be ensured that these sections are tested with the highest priority and as early as possible.

## 5. The time available is limited

Testing time for software is limited. It must be kept in mind that the time available for testing is not unlimited and that an effective test plan is very crucial before starting the process of testing. There should be some criteria to decide when to terminate the process of testing. This criterion needs to be decided beforehand. For instance, when the system is left with an acceptable level of risk or according to timelines or budget constraints.

## 6. Testing must be done with unexpected and negative inputs:

Testing should be done with correct data and test cases as well as with flawed test cases to make sure the system is leak proof. Test cases must be well documented to ensure future reuse for testing at later stages. This means that the test cases must be enlisted with proper definitions and descriptions of inputs passed and respective outputs expected. Testing should be done for functional as well as the non-functional requirements of the software product.

## 6. Inspecting test results properly

Quantitative assessment of tests and their results must be done. The documentation should be referred to properly while validating the results of the test cases to ensure proper testing. Testing must be supported by automated tools and techniques as much as possible. Besides ensuring that the system does what all it is supposed to do, testers also need to ensure that the system does not perform operations which it isn’t supposed to do.

## 7. Validating assumptions

The test cases should never be developed on the basis of assumptions or hypothesis. They must always be validated properly. For instance, assuming that the software product is free from any bugs while designing test cases may result in extremely weak test cases.